THE UNIVERSITY of TENNESSEE KNOXVILLE

Discontinuous Element Insertion Program

May 9, 2018

A program for inserting zero-thickness elements into a continuous finite element mesh in two and three dimensions. These elements are used for intrinsic cohesive zone modeling and for the Discontinuous Galerkin method.

By

T.J. Truster

Department of Civil and Environmental Engineering, University of Tennessee, Knoxville, 318 John D. Tickle Engineering Building, Knoxville, TN 37921

LICENSE

VERSION HISTORY

Version 1.3 (May 2018)

- Added content module for Gmsh mesh reader/writer
- Command file for execution of Code_Aster cohesive zone analysis with sample Gmsh file produced by DEIProgram
- Bug fixes for Octave compatibility
- Added workflow demonstration in Section 1.4 for mesh generation, coupler insertion, mechanical analysis, and results visualization
- Expanded element types: 5-node pyramid and 14-node pyramid

Version 1.2 (October 2017)

- Created data-structure of interfacial arrays and associated helper functions to streamline the I/O of the DEIProgram modules
- Created utility functions to query the output mesh and coupler topology, such as which couplers are attached to which regions
- Regrouped example files to highlight capabilities such as multi-material model examples
- Bug fixes for ABAQUS reader/writer
- Treatment of periodic boundary conditions (under development, in “dev” branch)
- Added more mesh conversion scripts along with example files

Version 1.1 (April 2016)

- Expanded content module examples for ABAQUS reader/writer
- Added mesh converter for 4-node quadrilateral to 3-node triangle

Version 1.0 (December 2015)

- Initial release
- Content modules: DEIProgram, FEA_Program, PostParaview, MATLAB plotting modules, ABAQUS reader/writer
- Element types included:
    - Two-dimensional: 3-node triangle, 4-node quadrilateral, 6-node triangle, 9 node quadrilateral
    - Three-dimensional: 4-node tetrahedra, 8-node brick, 10-node tetrahedra, 27-node brick
- Coupler types included:
    - Zero-thickness facet elements (cohesive zone); discontinuous Galerkin elements
    - Two-dimensional: 3-node triangle, 4-node quadrilateral, 6-node triangle, 9 node quadrilateral
    - Three-dimensional: 4-node tetrahedra, 8-node brick, 10-node tetrahedra, 27-node brick

INSTALLATION

The source code for DEIP can be obtained either by downloading a compressed folder from the Computational Laboratory for the Mechanics of Interfaces http://clmi.utk.edu/software/ or by cloning the repository from https://bitbucket.org/trusterresearchgroup/deiprogram.

Install the DEIP source code using the following steps:

1. Open MATLAB/OCTAVE terminal
2. Navigate the working directory to the folder containing the script InstallDEIP.m
3. Run the script InstallDEIP.m to load all source files into the search path

The source code can then be verified by executing the script ‘examples\TestAll.m’. All files show run without any error messages. Note that this process may take 1-5 minutes depending on the computer platform.

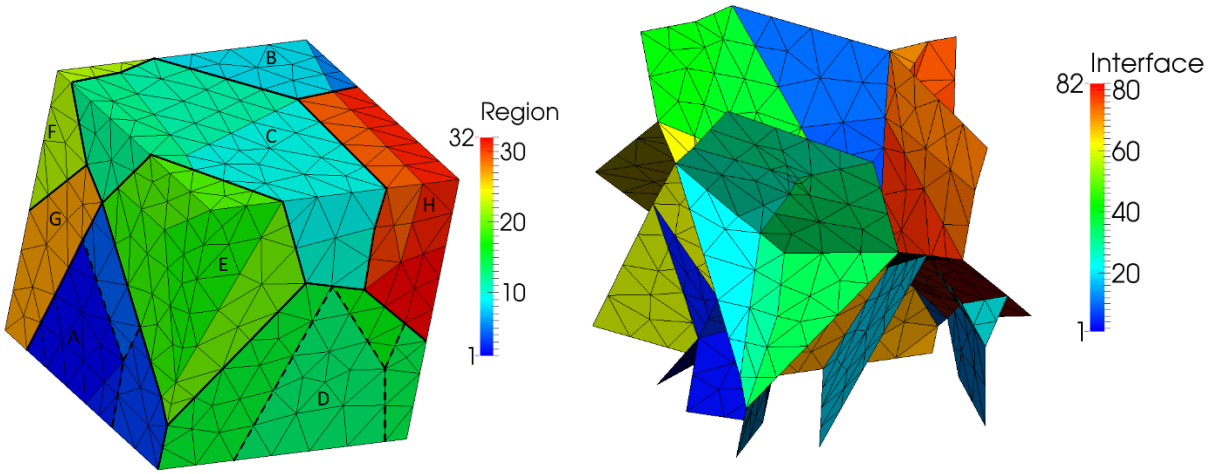
EXECUTIVE SUMMARY

The program Discontinuous Element Insertion Program (DEIP) is a set of MATLAB scripts designed for inserting zero-thickness interface elements, termed herein as “couplers”, into continuous finite element meshes in two and three dimensions. Insertion is governed solely by the mesh topology and is specified according to regions or subdomains within the overall analysis domain, a geometrically intuitive means to designate the coupler locations. The algorithm is self-contained and requires only nodal coordinates and element connectivity as input. A wide class of volume elements and interface couplers are treated within the framework. Interfaces of arbitrary complexity are naturally accommodated since the algorithm is topologically-based. Couplers can be selectively inserted within specific regions or along specific interfaces. Also, different types of couplers as well as different material properties may be directly assigned to these particular sets, providing an intuitive means to complete the description of the interfacial-modified mesh for analysis purposes.

The DEIP script files are also compatible with the open-source program OCTAVE. Additional modules are provided with DEIP to enable input/output of data for use in finite element analysis and visualization. These modules include a standalone linear finite element analysis code, mesh and contour plotting functions for MATLAB, and export functions for PARAVIEW.

Simple application programming interfaces (API) are provided for importing/exporting meshes generated by two popular platforms: the commercial finite element software ABAQUS and the open-source mesh generator Gmsh. Example command files are also provided for cohesive zone modeling in ABAQUS and the open-source finite element framework Code_Aster. These API are examples for how DEIP can be integrated into the workflow of other finite element codes.

Example files are provided to demonstrate the features of the code, including domains constructed from Voronoi tessellation with distinct material properties on each interface.



(a) (b)

Polycrystalline domain containing quadratic tetrahedral elements: (a) solid mesh; (b) inserted couplers between all grains and certain subgrains

ACKNOWLEDGEMENTS

The work described herein has been supported primarily by the project DE-AC05-000R22725 at Oak Ridge National Laboratory through Dr. Sam Sham. The interface couplers generated using this program were used in finite element models for investigating grain boundary cavitation and sliding within polycrystalline materials having hierarchical microstructural features.

We are grateful for the helpful feedback from Dr. Robert Dodds Jr., Dr. Kristine Cochran, Sunday Aduloju, Pinlei Chen, Wesley Hicks, Russell Hollman, and Omar Nassif prior to the general release of this software.

The following individuals have contributed ideas and implementation to later releases of the code: Dr. Mark Messner, Dr. David Parks, and Dr. Van Tung Phan.

TABLE OF CONTENTS

TABLE OF FIGURES

1. DISCONTINUOUS ELEMENT INSERTION PROGRAM

The `DEIProgram` is a general-purpose program in MATLAB for inserting zero-thickness interface elements, termed herein as "couplers", into specified regions of conforming finite element meshes. The duplication of nodes to accommodate the couplers is treated in a systematic fashion, employing only topological operations on the original element connectivity of the mesh to perform the coupler insertions. Therefore, it can be applied to general element types for two and three dimensional problems with minimal input from the user. Couplers can be selectively inserted within specific regions or along specific interfaces. Also, different types of couplers as well as different material properties may be directly assigned to these particular sets, providing an intuitive means to complete the description of the interfacial-modified mesh for analysis purposes.

The theory underlying the insertion of interface couplers is summarized in Section 1.1; additional details are contained in [1]. Usage of the program is described in Section 1.2 through the example files provided with the program, listed in Section 8. A list of currently implemented element and coupler types is provided in Section 6.

1.1. Theory

1.1.A. Topological Definitions

Consider a *domain* consisting of a conforming mesh of finite *elements* in two (2D) or three (3D) dimensional space. Throughout the following discussions, topological entities are identified by *italic* typeset. Each *element* in the mesh is defined by a set of *nodes* which are points within the *domain* associated with their particular coordinates; see the 2D example in Figure 1.1. In 2D, meshes containing a mixture of triangular and quadrilateral *elements* is considered, while in 3D, meshes of solely tetrahedral or hexahedral *elements* is assumed. In Figure 1.1, *nodes* are designated by numbers and *elements* are denoted by lower-case letters. For example, *element* a is composed of the three *nodes* 1, 4, and 5. The *edges* of *elements* in 2D are defined by the line segment connecting two *nodes*, and the *faces* of *elements* in 3D are defined by the set of *nodes* connected by *edges* which form a closed loop within a single plane in space. The term *facet* is applied to refer either to an *edge* in 2D or a *face* in 3D in the algorithmic descriptions that follow.

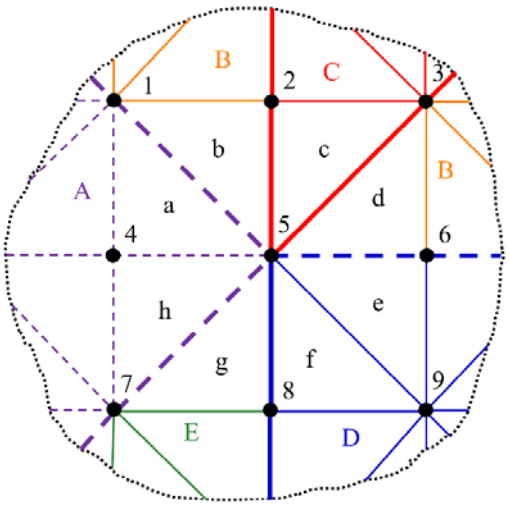


Figure 1.1. Conforming finite element mesh containing regions

In addition to the above standard features associated with finite element discretization, we define a *region* as a contiguous set of *elements* within the *domain*, which in general may form nonconvex subdomains and consist of spatially disjoint sets of *elements*. Each *element* in the *domain* is a member of exactly one *region*. In Figure 1.1, the *regions* are denoted by capital letters, and the *elements* belonging to each *region* share the same color. Examples of *regions* in the context of finite element modeling include the grains within a polycrystal, fibers and the surrounding matrix in composites, concrete and steel in reinforced concrete, and so forth, where each *region* is considered to have different material properties. However, herein a *region* is a purely geometrical construct to enable completely general applications. The role played by the *regions* is central to the algorithm for inserting the "interface elements".

The *facets* of all the *elements* in the *domain* can be separated into three disjoint sets. The first set are those *facets* which lie on the *domain* boundary, which are adjacent to exactly one *element*. The second set are those which lie between *elements* of two different *regions*, which are said to belong to *interfaces*. This set is further divided according to pairs of *regions*, such that *interface(A,B)* is the set of all *facets* between *region A* and *region B*. The third set are those which lie between *elements* of the same *region*. Such *facets* belonging to *region C* will be denoted as *intraface(C)*, and so forth. *Interfaces* are shown as thick line segments in Figure 1.1 while *intrafaces* are shown as thin line segments.

1.1.B. Coupler Insertion Algorithm

A topological-based algorithm is presented for inserting *couplers* along the *interfaces* or *intrafaces* in the *domain*. In the finite element literature, such computational entities are typically referred to as "zero-thickness elements" or "interface elements". Herein, we apply the term *coupler* to distinguish from the other topological definitions made in Section 1.1.A and to provide for broader types of computational entities. Thus, a *coupler* is defined as a topological unit consisting of *nodes* from exactly two *elements* which are adjacent across either an *interface* or *intraface*. The *coupler* is generated by duplicating the *nodes* lying on the *facet* shared by the two *elements* to effectively split the *mesh* along that *facet*. These *couplers* commonly appear as numerical realizations of discontinuous formulations for modeling PDEs. The treatment of such formulations is beyond the scope of this technical note. Regarding the intrinsic cohesive zone models, the reader may consult [2] for mathematical aspects and [3, 4] for notes on implementation. Similarly, the formulation [5] and implementation [6, 7] of the Discontinuous Galerkin method can be found elsewhere. The common feature of these and other discontinuous interpolation methods is that they require additional topological data beyond just the *nodes* and *elements* of the *mesh*.

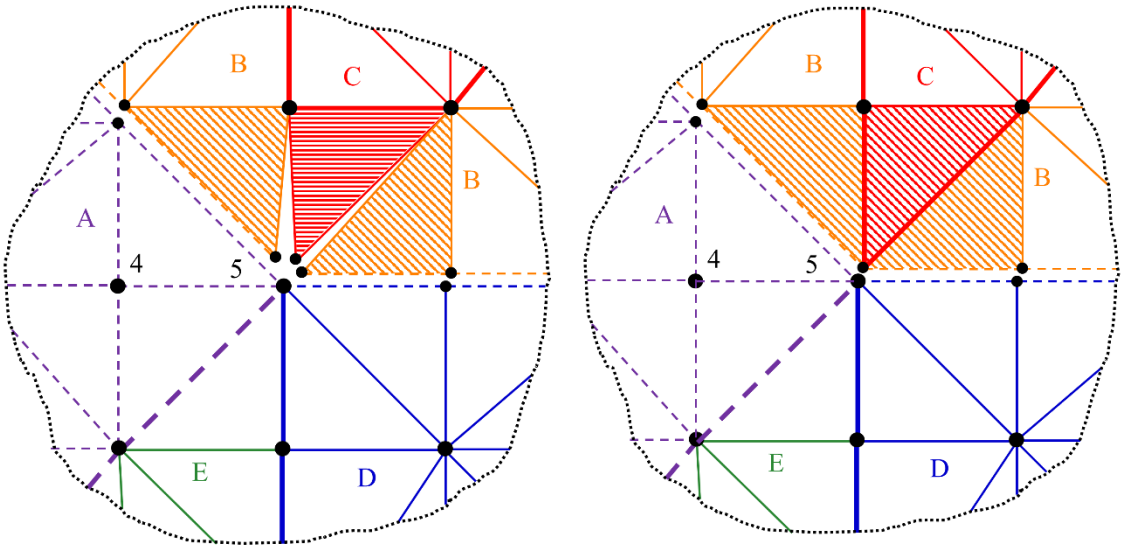
Starting from an initially conforming finite element mesh, the insertion of *couplers* is accomplished by reference to the *interfaces* and *intrafaces* between various *regions*. The *region* is a natural geometric entity for assigning the desired location of the *couplers*. For example, when modeling debonding in fibrous composites or cavitation along grain boundaries in polycrystals, the *interfaces* between the different material *regions* is the desired location. Similarly, *intrafaces* are the natural location for *couplers* when using Discontinuous Galerkin numerical methods or when simulating general crack propagation with cohesive zone models.

The input data for the algorithm is simply the spatial coordinates of the *nodes*, the list of *nodes* connected to each *element*, and the list of *elements* belonging to each *region*. This topological data is provided by almost every finite element mesh generation software package, meaning that

minimal data preparation is required by the user. Next, the list of *interfaces* and *intrafaces* is provided to specify which topological locations to insert the *couplers*. For example, the *interface* between a fiber *region A* and matrix *region B* would be indicated by flagging *interface(A,B)*; in Figure 1.1, this is indicated by the dashed lines between these *regions*. Herein, these *facets* where *couplers* are to be inserted are called "cut" *facets*. Similarly, the *intraface(A) facets* inside *region A* are dashed in Figure 1.1 to indicate that they will also be cut. The algorithm then determines the set of *couplers* to insert and the set of *nodes* to duplicate through an automated process based upon the topology of the mesh.

The crucial aspect of the insertion algorithm is the node duplication procedure, which relies upon the concept of *sectors* of *elements* surrounding a focus *node*. Two *elements* are defined to belong to a *sector* if the shared *facet* between them is not a cut *facet*, namely it is not designated for a *coupler*. A *sector* is then the largest set of *elements* satisfying this definition; a single *element* constitutes a *sector* if all of the *facets* of that *element* sharing the focus *node* are cut. Also, the union of all *sectors* is equal to the set of all *elements* surrounding the focus *node*. In general, a *sector* will consist of all *elements* for a single *region* only if all *interfaces* attached to that *region* are to be cut. Otherwise, a *sector* may consist of *elements* from multiple *regions*. This definition ensures that a single *sector* is obtained for the cases of a *node* attached to only one cut *edge* in 2D and a *node* with cut *faces* lying on a single (non-smooth) manifold in 3D. Thus, the *node* would not be duplicated, and the field interpolation remains continuous at that *node*.

Referring to the mesh in Figure 1.1, *elements b*, *c*, and *d* belong to one *sector*, since *couplers* are added along *interface(A,B)* and *interface(B,D)*. Within the finite element discretization, each of these *elements* is required to have a continuous interpolation of the solution field. Therefore, the nodal coefficient of the solution field associated with this corner *node* must be identical for each of the three *elements*, so that the *sector* remains "solid". *Elements* in other *sectors* will have a discontinuous functional interpolation, so the nodal value of the solution field should be independent or distinct between the *sectors*. For the example shown, the duplication of *node 5* onto each *region* does not satisfy the continuity requirement; this leads to an incorrect discretization shown in Figure 1.2 (a), with open gaps between *elements b* and *c*. Using such a discretization would lead to erroneous numerical results. Rather, a single copy of *node 5* should be assigned to these three *elements* which belong to the *sector*, as indicated by the single shading in Figure 1.2 (b). This situation arises because *regions B* and *C* are prescribed to remain connected without adding *couplers* along *interface(B,C)*. In contrast, within *region A*, *node 4* will be duplicated for all surrounding *elements*, since each *element* constitutes a *sector* due to the insertion of *intraface couplers*.



(a) (b)

Figure 1.2. Node duplication: (a) incorrect discontinuity obtained by using regions; (b) correct continuity obtained by using sectors

The *coupler* insertion algorithm consists of six phases, which are summarized below:

(a) Construct the set of *elements* attached to each *node*
(b) Categorize all *facets* in the mesh as *boundary*, *interface*, or *intraface*.
(c) Designate all cut *interface facets*, where *couplers* are to be inserted.
(d) Duplicate *nodes* according to *sectors* for *interfaces*
(e) Duplicate *nodes* for *intrafaces*
(f) Construct the connectivity of *nodes* to *couplers* according to specified templates

Each of these phases is described in detail within [1].

1.2. Usage of the Discontinuous Element Program (DEIP)

The insertion of couplers using `DEIProgram` is accomplished by writing MATLAB scripts which call the various modules of the program. The only required input is information related to the connectivity of the finite element mesh.

An overview of a typical sequence of steps using higher-level subroutines is as follows:

(a) Create an input file and generate or load the nodes and elements of the mesh; options include manual input, the block scripts in Section 2.4, the ABAQUS input file reader in Section 5, or the Gmsh mesh file reader in Section 6
(b) Call the function `DEIPFunction` to perform node duplication along interfaces and intrafaces
(c) Call the function `InterFunction` to define the new coupler connectivity along interfaces and intrafaces
(d) Call the script `UpdateNodeSet` to update lists involving nodal ID (such as boundary conditions) to reflect the duplicated nodes.

An overview of an advanced sequence of steps using lower-level subroutines is as follows:

(e) Create an input file and generate or load the nodes and elements of the mesh; options include manual input, the block scripts in Section 2.4, the ABAQUS input file reader in Section 5, or the Gmsh mesh file reader in Section 6
(f) Call the script `DEIProgram` to perform node duplication along interfaces and intrafaces
(g) Call the scripts `FormDG` or `FormCZ` to define the new coupler connectivity along interfaces and intrafaces
(h) Call the script `UpdateNodeSet` to update lists involving nodal ID (such as boundary conditions) to reflect the duplicated nodes.

Subsequently, other programs may be executed to perform other tasks, such as performing a patch test using the linear finite element analysis program in Section 2, plotting the mesh using functions in Section 3, exporting the mesh to Paraview in Section 4, writing an ABAQUS input file containing the inserted couplers in Section 5, or writing a Gmsh mesh file containing the inserted couplers in Section 6.

Discussion of the input relevant to the modules listed in the above four steps is provided in the context of the example file `DEIPex1`. The user is advised to consult this file alongside the descriptions below.

Since Version 1.2, much of the functionality of the insertion scripts `DEIProgram` and `FormDG` have been subsumed into the functions `DEIPFunction` and `InterFunction`, with simplified I/O listings. The descriptions of these functions below are associated with example file `DEIPex1_2`.

1.2.A. Mesh geometry format

First, the geometrical information pertinent to the conforming finite element mesh needs to be provided. Because the program `DEIProgram` is developed as a script file, specific variable names must be defined by the user which contain this relevant geometrical information. These arrays are listed below.

- `Coordinates(node,1:ndm)` = spatial coordinates of node `"node"`
- `NodesOnElement(elem,1:nen)` = list of nodes connected to element `"elem"`
- `RegionOnElement(elem)` = region ID of element `"elem"`
- `numnp` = total number of nodes in mesh = `size(Coordinates,1)`
- `numel` = total number of elements in mesh = `size(NodesOnElement,1)`
- `nummat` = number of different regions = `max(RegionOnElement)`
- `ndm` = number of spatial dimensions (2 or 3) = `size(Coordinates,2)`
- `nen` = maximum number of nodes per element = `size(NodesOnElement,2)`

The node ID and element ID are implicitly defined as the row number in the arrays `Coordinates` and `NodesOnElement`, respectively.

Example data is provided for a two dimensional mesh of linear triangular elements in `DEIPex1` as shown in Figure 1.3.

Figure 1.3. Geometrical data for 2D finite element mesh

The geometry formed by the input in Figure 1.3 is depicted in Figure 1.4, where the color corresponds to the region assigned to the element and the element ID is shown in the center of each triangular element.

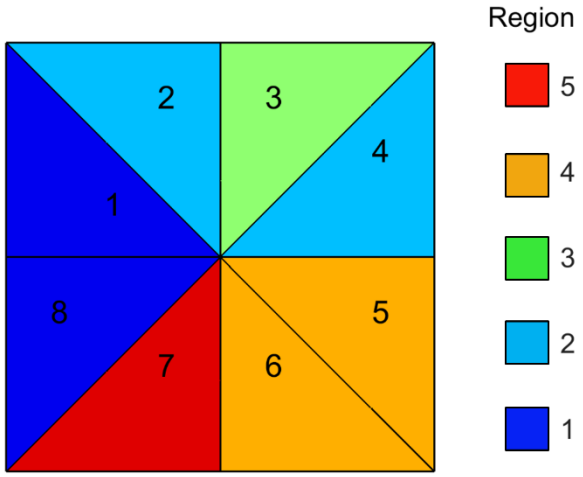


Figure 1.4. Finite element mesh for simple example

Next, the list of the region interfaces and intrafaces to be separated by couplers must be provided. This information is provided in the logical array `InterTypes`, which is a square array with dimensions equal to `nummat`. For the entry in `InterTypes(reg2,reg1)`, couplers are to be inserted along the interface between region "`reg1`" and region "`reg2`" if the entry is "1", and no insertion along the interface is performed if the value is "0". Intraface couplers are designated by a value of "1" on the diagonal, e.g. `InterTypes(3,3)=1` means that all elements inside region 3 are to be separated by couplers. Only the lower triangle and the diagonal of this array are accessed. An example is shown in Figure 1.5, which corresponds to the mesh described in Section 4.1 of [1].

Figure 1.5. Designation of coupler insertion using `InterTypes`

Subsequently, the identifier for each interface/intraface is given by the formula:

```
regI = reg2*(reg2-1)/2 + reg1
```

where `regI` is the identifier for the interface between regions `reg1` and `reg2`, or the intraface between the elements of region `reg1`. For example, if `reg1=2` and `reg2=4`, then `regI=4*3/2+2=8`. This interface identifier is used within the scripts of `DEIProgram`, and it also provides a convenient means to differentiate the groups of couplers inserted in the mesh so that distinct material properties can be assigned to them.

Recall from Section 1.1.B that the intraface couplers within regions are inserted after the interface couplers between regions. Also, if a region is designated for intraface couplers by having a "1" on the diagonal of `InterTypes`, then all interfaces adjoining that region must also be designated for couplers in order that the generated mesh is analysis suitable. Therefore, the `DEIProgram` script performs this check prior to the node duplication and provides a Warning to the user if there are missing "1" in the rows and columns of `InterTypes`. It will then assign the value of "1" into the missing entries for the user.

```
>> DEIPex1
Warning: some interfaces not requested in Intertypes and have been added
intraface: 1; additional interface flags have been added
>>
```

```
53 -    InterTypes = [1 0 0 0 0
54                   1 0 0 0 0
55                   0 0 0 0 0
56                   1 1 0 0 0
57                   1 0 0 0 0];
```

Figure 1.6. Warning message issued when interface couplers are not properly designated

1.2.B. Node duplication using `DEIPFunction` and `DEIProgram`

Then, the `DEIPFunction` can be called to perform the duplication of nodes along interfaces and intrafaces as described in Section 1.1 and within [1]. The syntax of the function is given in

Figure 1.7. The input mesh topology arrays `[InterTypes, NodesOnElement, RegionOnElement, Coordinates]` and sizing parameters `[numnp, numel, nummat, nen, ndm]` are defined in Section 1.2.A. The generated output arrays `[NodesOnElement, RegionOnElement, Coordinates]` have been updated to include the duplicated node numbers along intrafaces and interfaces, but only the domain elements are included; couplers are inserted in Section 1.2.C. Additionally, a data structure `Output_data` is created that contains all the intermediate arrays needed by subsequent scripts for coupler insertion. The class `@facet_data` is outlined in Section 1.3; the user does not have to understand the data in these arrays to use the functionality of `DEIProgram`.

Note that additional input and output arrays are supplied to `DEIPFunction` for models containing periodic boundary conditions. These features will be described in later versions of the code.

```
61 -    [NodesOnElement,RegionOnElement,Coordinates,numnp,Output_data] ...
62          = DEIPFunction(InterTypes,NodesOnElement,RegionOnElement,Coordinates,numnp,numel,nummat,nen,ndm);
```

Figure 1.7. Minimal input and output listing for function `DEIPFunction`

Alternatively, the `DEIProgram` can be executed to perform the duplication of nodes along interfaces and intrafaces as described in Section 1.1 and within [1]. Several output arrays are generated during this process. Particularly useful arrays are listed below; other arrays can be found by consulting the source code and the descriptions in [1]. These other arrays may be useful to the user for updating information concerning the finite element mesh, such as material properties, boundary conditions, lists of nodes in regions, etc.

- `Coordinates` array is updated with the duplicated nodes appended to the end
- `NodesOnElement` array is updated with the revised connectivity of the duplicated nodes
- `ElementsOnFacet(fac,1:4)` = `[elem1 elem2 fac1 fac2]` where "`elem1`" and "`elem2`" are the two elements on either side of facet "`fac`" in the mesh and "`fac1`" and "`fac2`" are the local facet numbers of the respective elements attached to "`fac`"; listing of the local facet numbering for different element types is provided in Section 7.2
- `FacetsOnInterface(1:numfac)` = identifiers for facets according to the interface/intraface identifier `regI`. The values are stored in compressed sparse row format according to the bounds in the indexing array `FacetsOnInterfaceNum`. See the example in Figure 1.10.
- `ElementsOnNode` = elements that are attached to each node, the inverse of the `NodesOnElement` table. This array references the original conforming mesh.
- `ElementsOnNodeNum` = number of elements that are attached to each node
- `ElementsOnNodeDup(locE,node)` = new ID of the node "`node`" that is attached to the element "`elem`" that is listed in `ElementsOnNode(locE,node)` after the duplication process.
- `numfac` = total number of interface/intraface facets in the mesh. Facets on the domain boundary are excluded and are placed in other arrays.

The output arrays generated by `DEIPFunction` or `DEIProgram` are in the format suitable for performing the last two steps of updating the mesh, namely the generation of the coupler connectivity and updating other lists involving nodal IDs, such as boundary conditions.

1.2.C. Generation of couplers using `InterFunction` or `FormCZ` and `FormDG`

The generation of couplers is accomplished using the function `InterFunction`. This function generate couplers according to the connectivity templates provided in Section 7.1. Current formats include cohesive zone (CZ) couplers, which include nodes only along the interface, and discontinuous Galerkin (DG) couplers, which include all nodes from each element adjacent to the coupler.

The minimal input and output arrays are shown in Figure 1.8 for the example file `DEIPex1_2` and summarized below:

- `couplertype` = integer designating the coupler type and insertion option; described below
- `InterTypes`
- `NodesOnElement`
- `RegionOnElement`
- `Coordinates`
- `numnp` = current number of nodes in mesh = `size(Coordinates,1)`
- `numel` = current number of elements in mesh = `size(NodesOnElement,1)`
- `nummat` = current number of different regions plus the number of distinct inserted coupler types = `max(RegionOnElement)`
- `nen` = the max number of nodes per element in the original contiguous mesh
- `ndm` = number of spatial dimensions (2 or 3) = `size(Coordinates,2)`
- `Input_Data` = data structure of class `@facet_data` generated from `DEIPFunction`

```
64 -    [NodesOnElement,RegionOnElement,Coordinates,numnp,nen,numel,nummat,RegionsOnInterface...
65      ] = InterFunction(2,InterTypes,NodesOnElement,RegionOnElement,Coordinates,numnp,numel,nummat,nen,ndm,Output_data);
66
```

Figure 1.8. Minimal input and output listing for function `InterFunction`

The insertion option parameter `couplertype` is used to designate several options for the type of couplers to insert into the connectivity array `NodesOnElement`. Currently implemented options are given by the integers listed below.

1. Insert cohesive zone couplers along all interfaces and intrafaces corresponding to the region pairs having a `1` listed in `InterTypes`
2. Insert Discontinuous Galerkin couplers along all interfaces and intrafaces corresponding to the region pairs having a `1` listed in `InterTypes`

Also, `InterTypes` can be modified after calling `DEIPFunction`; by replacing a `1` with `0`, an internal surface or notch is created since node duplication has occurred but couplers will not be used to tie the nodes back together. This idea is illustrated in the example file `PlateCOHNotch`.

Additional optional arguments can be provided, which are relevant to the linear FEA Program described in Section 2. These are shown in Figure 1.9 for the example file `NeperModel_2`.

- `ielNL` = `[iel nonlin]` (only the first integer is required, the other is assumed as 0)
- `iel` = element library identifier for the couplers to be generated.
- `nonlin` = 0 (designates linear PDE-type element)
- `CZprop` = list of material properties associated with this coupler type, such as the value of the penalty stiffness parameter
- `MatTypeTable` = current list of element library identifiers for materials in the mesh
- `MateT` = current list of material properties for the mesh

Note that the coupler property listing `CZprop` has two possible versions. It can be an array with a single row, which will be assigned as the properties for all couplers in the model regardless of the region pair. It can also be a cell array with a single dimension and each entry in the cell is an single-row array with the properties for the particular region pair. Namely, `CZprop{regI} = [stiff]` is the elastic stiffness associated to the couplers to be inserted between regions `reg1` and `reg2`. A detailed example that shows the user how to create this array starting from a desired set of material properties is given in the file `NeperModel_2`. As a physical example, these properties could be a misorientation-dependent viscosity for grain boundary sliding between grains (regions) that is computed as a function of the Euler angle of each grain (region).

```
97      % Insert DG couplers
98 -    [NodesOnElement,RegionOnElement,Coordinates,numnp,nen,numel,nummat,RegionsOnInterface,MateT,MatTypeTable...
99      ] = InterFunction(2,InterTypes,NodesOnElement,RegionOnElement,Coordinates,numnp,numel,nummat,nen,ndm,...
100                       Output_data,8,CZprop,MateT,MatTypeTable);
```

Figure 1.9. Extended input and output listing for function `InterFunction`; from `NeperModel_2`

The output of the script is appended to the arrays supplied as input. The new couplers will be added to the end of the list of elements in the mesh, `NodesOnElement`. Also, these couplers will be given a unique identifier in `RegionOnElement`. Also note that the number of elements `numel` and the number of materials/regions `nummat` will be increased, and the number of nodes per element `nen` will reflect the newly inserted couplers according to the templates in Section 7.1.

The array `RegionsOnInterface` is supplied so that the user is aware of which elements within `NodesOnElement` correspond to which interface type (region pair). Each row in this array corresponds to an interface type or region pair that is present in the model; note that even though a `1` may be listed in `InterTypes` (e.g. for `reg1=3` and `reg2=4`), these two regions may not be topologically adjacent in the model. The six columns in `RegionsOnInterface` correspond to:

- `matID` = material ID or row in `MateT` where the properties for this interface type are assigned
- `regA` = region on side 1 of the interface (lesser region ID)
- `regB` = region on side 2 of the interface (greater region ID)
- `regI` = `regB*(regB-1)/2 + regA` = interface identifier

- `coup1` = element ID of first coupler for this interface identifier
- `coup2` = element ID of last coupler for this interface identifier

Couplers of each interface type (region pair) are inserted concurrently so that their numbering in `NodesOnElement` is sequential.

Two optional outputs are also supplied:

- `MatTypeTable` = updated list of element library identifiers; populated by default values if input is not supplied
- `MateT` = `[reg1 reg2 nel1 nel2]` where region "`reg1`" is on the first side of the coupler and region "`reg2`" is on the second side of the coupler for all facets of that type, and "`nel2`" and "`nel2`" are the actual number of nodes per element of the solid elements on either side of the coupler. This output assures that the solid elements of a particular region are on one side of the interface consistently, which is usually critical for assigning proper material parameters within a subsequent DG finite element analysis.

The sample script provided in Figure 1.10 is quite general and applies to both 2D and 3D analyses and CZ and DG couplers. Other commands can be found throughout the examples provided in the program, listed in Section 8.

Alternatively, couplers can also be generated using the scripts `FormCZ` and `FormDG`. These scripts generate couplers according to the connectivity templates provided in Section 7.1. Current formats include cohesive zone (CZ) couplers, which include nodes only along the interface, and discontinuous Galerkin (DG) couplers, which include all nodes from each element adjacent to the coupler. The user has complete control to designate whether CZ or DG couplers are applied along particular interfaces alone, or if coupler connectivity is neglected so that a discrete crack or discontinuous surface can be introduced in the mesh.

An example set of commands for inserting all DG couplers along interfaces while distinct inter-region numbering is given in Figure 1.10.

Figure 1.10. Insertion of couplers along interfaces using `FormDG`

The format of the input to `FormCZ` and `FormDG` is identical, and is summarized below.

- `SurfacesIi` = selected rows from `ElementsOnFacet`, designating the particular facets to be assigned with coupler connectivity
- `NodesOnElement`
- `RegionOnElement`
- `Coordinates`
- `numSIi` = number of facets = `size(SurfacesIi,1)`
- `nen_bulk` = the max number of nodes per element in the original contiguous mesh
- `ndm` = number of spatial dimensions (2 or 3) = `size(Coordinates,2)`
- `numel` = current number of elements in mesh = `size(NodesOnElement,1)`
- `nummat` = current number of different regions plus the number of distinct inserted coupler types = `max(RegionOnElement)`
- `maxmat` = maximum number of different couplers combinations to insert; usually a value of 2 is sufficient. The module allows for elements with different number of nodes per element to be inserted during the same call to `FormDG`, and each separate pair (e.g. linear triangles next to linear quadrilaterals) will be assigned a unique "material" identifier.

Additional optional arguments can be provided, which are relevant to the linear FEA Program described in Section 2.

- `iel` = element library identifier for the couplers to be generated.
- `nonlin` = 0 (designates linear PDE-type element)
- `mateprop` = list of material properties associated with this coupler type, such as the value of the penalty stiffness parameter
- `MatTypeTable` = current list of element library identifiers for materials in the mesh
- `MateT` = current list of material properties for the mesh

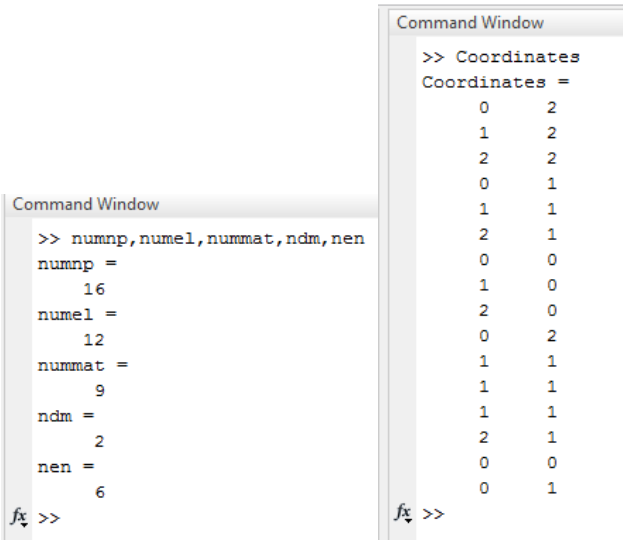
The output of the script is appended to the arrays supplied as input. The new couplers will be added to the end of the list of elements in the mesh, `NodesOnElement`. Also, these couplers will be given a unique identifier in `RegionOnElement`. Also note that the number of elements `numel` and the number of materials/regions `nummat` will be increased, and the number of nodes per element `nen` will reflect the newly inserted couplers according to the templates in Section 7.1.

Two optional outputs are also supplied:

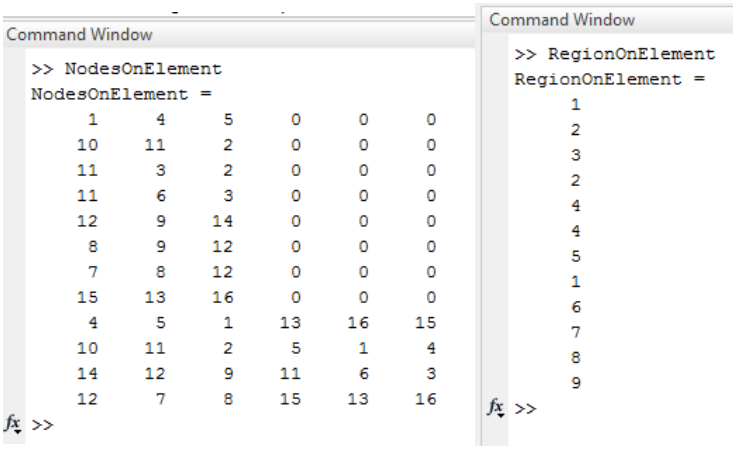
- `MatTypeTable` = updated list of element library identifiers; populated by default values if input is not supplied
- `MateT` = `[reg1 reg2 nel1 nel2]` where region "`reg1`" is on the first side of the coupler and region "`reg2`" is on the second side of the coupler for all facets in the list `SurfacesIi`, and "`nel2`" and "`nel2`" are the actual number of nodes per element of the solid elements on either side of the coupler. This output assures that the solid elements of a particular region are on one side of the interface consistently, which is usually critical for assigning proper material parameters within a subsequent DG finite element analysis.

The sample script provided in Figure 1.10 is quite general and applies to both 2D and 3D analyses and CZ and DG couplers. Other commands can be found throughout the examples provided in the program, listed in Section 8.

After running `FormDG`, the updated element connectivity and nodal coordinates for the example `DEIPex1` including the inserted couplers are provided in Figure 1.11. By comparing with Figure 1.3, the insertion process created new nodes 10 – 16 in `Coordinates` and new elements 9 – 12 in `NodesOnElements`. The original nodes 1 – 9 have remained unchanged. Also, the connectivity for elements 1 – 8 has been updated to reflect the duplicated nodes. Finally, the maximum number of nodes per element `nen` has been increased to 6 as a reflection of the 4 DG couplers that have been inserted. The original mesh contained 3-node linear triangular elements; as shown in Section 7.1, the DG coupler template for triangular elements is formed from the 3 nodes of the elements from each side of the interface facet.



(a)



(b)

Figure 1.11. Generated output from DEIP: (a) counting parameters and nodal coordinates; (b) element connectivity and region identifiers

1.2.D. Optional update to nodal sets

The last optional step is to update lists of node sets supplied by the user. Meshes generated from other programs such as ABAQUS and Gmsh will often have sets of nodes for various purposes, such as assigning boundary conditions. The duplication of nodes to insert the couplers, as accomplished in the previous 3 steps, leads to new nodes which may also be desirable to include in those lists. Therefore, a supplement script is provide, `UpdateNodeSet`, to accomplish this task. An example call to this module is provided in `DEIPex2`, which is shown in Figure 1.12.

```
167     % Update boundary conditions
168 -   NodeBCCG = NodeBC;
169 -   numBCCG = numBC;
170 -   reg = 0;
171 -   [NodeBC,numBC] = UpdateNodeSet(reg,RegionOnElement,ElementsOnNodeNum,...
172                                    ElementsOnNode,ElementsOnNodeDup,NodeBCCG,numBCCG);
173
```

Figure 1.12. Update to nodal identifier lists using `UpdateNodeSet`

The inputs to this script have already been defined above and are provided as outputs from `DEIProgram`, except for one. The parameter "`reg`" is a list containing the region ID for each version of a duplicated node which is to be assigned a copy of the entry from the input array `NodeBCCG`. The value of `reg` can be `0` to indicate all regions, as in Figure 1.12; otherwise, it can be any integers from `1` to `nummat`. Also, the array `NodeBCCG` can have any number of columns, but the first entry in each row must be the nodal ID of a node in the original contiguous mesh. The output array, `NodeBC`, is the updated list which only has entries for nodes belonging to regions provided in the list `reg`. For example, if node 5 is contained in the list `NodeBCCG`, and this node originally was attached to element 1 in region 1 and 4 in region 2, and the value of `reg` is `2`, then the output `NodeBC` will contain an entry for whatever the new nodal ID is for node 5 within region 2, which is 11 according to row (element) 4 of `NodesOnElement` in Figure 1.11 (b) since the node has been duplicated, in contrast to the original connectivity in Figure 1.3.

For the higher-level functions which produce the data structure `@facet_data`, the function `UpdateNodeSetFunction` accepts this argument and provides identical functionality. It is shown for example in `NeperModel_2`.

At the completion of these four steps, a finite element mesh containing discontinuous couplers has been generated that is suitable for analysis. Subsequently, other programs may be executed to perform other tasks, such as performing a patch test using the linear finite element analysis program in Section 2, plotting the mesh using functions in Section 3, exporting the mesh to Paraview in Section 4, writing an ABAQUS input file containing the inserted couplers in Section 5, or writing a Gmsh mesh file containing the inserted couplers in Section 6. Also, the user may simply write the generated data to text files or MATLAB .mat data files for use in external programs.

1.2.E. Utility routines to interrogate output mesh topology

Several utility routines are provided to help interrogate the lists of interface types and couplers that are generated by the `DEIProgram`.

- `[reg1, reg2] = GetIntRegions(regI, nummat)`: converts the interface identifier `regI` into the associated region pair `reg1, reg2`
- `regI = GetRegionsInt(reg1, reg2)`: converts the interface region pair `reg1, reg2`, or the intraface region `reg1=reg2`, into the interface identifier `regI`
- `[matID, reg1, reg2, regI, coup1, coup2] = GetCouplersIntRegions(reg1, reg2, RegionsOnInterface)`: determine whether the region pair `reg1, reg2` has any couplers present in the model associated with `RegionsOnInterface`. If no couplers are present, this function returns zeros for the variables. If couplers are present, then the function returns the pertinent values from `RegionsOnInterface` associated with that region pair.

Note that `GetCouplersIntRegions` is most helpful for assigning region-based material properties to the interface couplers as well as to determine the element IDs within `NodesOnElement` associated with a region pair.

1.3. Summary of input and output arrays

1.3.A. Higher-level functions

A summary of the inputs required for the node duplication function `DEIPFunction` is provided here; an example is shown in Figure 1.7. See example file `DEIPex1_2`.

- `Coordinates(node,1:ndm)` = spatial coordinates of node `"node"`
- `NodesOnElement(elem,1:nen)` = list of nodes connected to element `"elem"`
- `RegionOnElement(elem)` = region ID of element `"elem"`
- `InterTypes(reg2,reg1)`= couplers are to be inserted along the interface between region "`reg1`" and region "`reg2`" if the entry is "1", and no insertion along the interface is performed if the value is "0".
- `numnp` = total number of nodes in mesh = `size(Coordinates,1)`
- `numel` = total number of elements in mesh = `size(NodesOnElement,1)`
- `nummat` = number of different regions = `max(RegionOnElement)`
- `ndm` = number of spatial dimensions (2 or 3) = `size(Coordinates,2)`
- `nen` = maximum number of nodes per element = `size(NodesOnElement,2)`

The updated mesh arrays from the script are listed below:

- `Coordinates` array is updated with the duplicated nodes appended to the end
- `NodesOnElement` array is updated with the revised connectivity of the duplicated nodes
- `numnp` is increased to include the additional nodes that were duplicated
- `Output_data` is created that contains all the intermediate arrays needed by subsequent scripts for coupler insertion. Its class `@facet_data` is outlined in Section 1.3.

The generation of the connectivity for couplers is accomplished using `InterFunction`, with the input specified in Figure 1.8 for minimal usage or Figure 1.9 for modeling with `FEA_Program`:

- `couplertype` = integer designating the coupler type and insertion option; described below
- `InterTypes`
- `NodesOnElement`
- `RegionOnElement`
- `Coordinates`
- `numnp` = current number of nodes in mesh = `size(Coordinates,1)`
- `numel` = current number of elements in mesh = `size(NodesOnElement,1)`
- `nummat` = current number of different regions plus the number of distinct inserted coupler types = `max(RegionOnElement)`
- `nen` = the max number of nodes per element in the original contiguous mesh
- `ndm` = number of spatial dimensions (2 or 3) = `size(Coordinates,2)`
- `Input_Data` = data structure of class `@facet_data` generated from `DEIPFunction`

The output arrays with updated connectivity are given in Figure 1.11 for a particular example:

- `NodesOnElement` = the new couplers will be added to the end of the list of elements
- `RegionOnElement` = expanded to contain unique type identifier for the couplers
- `RegionsOnInterface` = array describing relation between interface couplers, regions, and interface identifiers

The array `RegionsOnInterface` is supplied so that the user is aware of which elements within `NodesOnElement` correspond to which interface type (region pair). Each row in this array corresponds to an interface type or region pair that is present in the model. The six columns in `RegionsOnInterface` correspond to:

- `matID` = material ID or row in `MateT` where the properties for this interface type are assigned
- `regA` = region on side 1 of the interface (lesser region ID)
- `regB` = region on side 2 of the interface (greater region ID)
- `regI` = `regB*(regB-1)/2 + regA` = interface identifier
- `coup1` = element ID of first coupler for this interface identifier
- `coup2` = element ID of last coupler for this interface identifier

Couplers of each interface type (region pair) are inserted concurrently so that their numbering in `NodesOnElement` is sequential.

1.3.B. Lower-level functions

A summary of the inputs required for the node duplication script `DEIProgram` is provided here; an example is shown in Figure 1.3. See example file `DEIPex1_2`.

- `Coordinates(node,1:ndm)` = spatial coordinates of node `"node"`
- `NodesOnElement(elem,1:nen)` = list of nodes connected to element `"elem"`

- `RegionOnElement(elem)` = region ID of element `"elem"`
- `InterTypes(reg2,reg1)`= couplers are to be inserted along the interface between region "`reg1`" and region "`reg2`" if the entry is "1", and no insertion along the interface is performed if the value is "0".
- `numnp` = total number of nodes in mesh = `size(Coordinates,1)`
- `numel` = total number of elements in mesh = `size(NodesOnElement,1)`
- `nummat` = number of different regions = `max(RegionOnElement)`
- `ndm` = number of spatial dimensions (2 or 3) = `size(Coordinates,2)`
- `nen` = maximum number of nodes per element = `size(NodesOnElement,2)`

The updated mesh arrays from the script are listed below:

- `Coordinates` array is updated with the duplicated nodes appended to the end
- `NodesOnElement` array is updated with the revised connectivity of the duplicated nodes
- `ElementsOnFacet(fac,1:4)` = `[elem1 elem2 fac1 fac2]` where "`elem1`" and "`elem2`" are the two elements on either side of facet "`fac`" in the mesh and "`fac1`" and "`fac2`" are the local facet numbers of the respective elements attached to "`fac`"; listing of the local facet numbering for different element types is provided in Section 7.2
- `FacetsOnInterface(1:numfac)` = identifiers for facets according to the interface/intraface identifier `regI`. The values are stored in compressed sparse row format according to the bounds in the indexing array `FacetsOnInterfaceNum`. See the example in Figure 1.10.
- `ElementsOnNode` = elements that are attached to each node, the inverse of the `NodesOnElement` table. This array references the original conforming mesh.
- `ElementsOnNodeNum` = number of elements that are attached to each node
- `ElementsOnNodeDup(locE,node)` = new ID of the node "`node`" that is attached to the element "`elem`" that is listed in `ElementsOnNode(locE,node)` after the duplication process.
- `numfac` = total number of interface/intraface facets in the mesh. Facets on the domain boundary are excluded and are placed in other arrays.

The generation of the connectivity for couplers is accomplished using `FormCZ` and `FormDG`, with the input specified in Figure 1.10:

- `SurfacesIi` = selected rows from `ElementsOnFacet`, designating the particular facets to be assigned with coupler connectivity
- `NodesOnElement`
- `RegionOnElement`
- `Coordinates`
- `numSIi` = number of facets = `size(SurfacesIi,1)`
- `nen_bulk` = the max number of nodes per element in the original contiguous mesh
- `ndm` = number of spatial dimensions (2 or 3) = `size(Coordinates,2)`
- `numel` = current number of elements in mesh = `size(NodesOnElement,1)`
- `nummat` = current number of different regions plus the number of distinct inserted coupler types = `max(RegionOnElement)`

- `maxmat` = maximum number of different couplers combinations to insert; usually a value of 2 is sufficient. The module allows for elements with different number of nodes per element to be inserted during the same call to `FormDG`, and each separate pair (e.g. linear triangles next to linear quadrilaterals) will be assigned a unique "material" identifier.

The output arrays with updated connectivity are given in Figure 1.11 for a particular example:

- `NodesOnElement` = the new couplers will be added to the end of the list of elements
- `RegionOnElement` = expanded to contain unique type identifier for the couplers
- `MateT` = `[reg1 reg2 nel1 nel2]` where region "`reg1`" is on the first side of the coupler and region "`reg2`" is on the second side of the coupler for all facets in the list `SurfacesIi`, and "`nel2`" and "`nel2`" are the actual number of nodes per element of the solid elements on either side of the coupler.

The data structure `facet_data` associated with the higher-level function `DEIPFunction` and `InterFunction` groups together many of the intermediate arrays used by the lower-level scripts for coupler insertion. These topological data arrays may be useful to the user, and hence are listed below.

- Counters: `numSI`, `numfac`, `numinttype`, `numCL`
- Element ID connected to interior or boundary facets: `numEonB`, `numEonF`, `ElementsOnBoundary`, `ElementsOnFacet`
- Element ID connected to nodes: `ElementsOnNode`, `ElementsOnNodeDup`, `ElementsOnNodeNum`, `ElementsOnNodeNum2`
- Facet ID connected to elements: `FacetsOnElement`, `FacetsOnElementInt`
- Facet ID connected to regions: `FacetsOnInterface`, `FacetsOnInterfaceNum`
- Facet ID connected to elements: `FacetsOnNode`, `FacetsOnNodeCut`, `FacetsOnNodeInt`, `FacetsOnNodeNum`
- Node ID connected to elements and regions: `NodeCGDG`, `NodeReg`, `NodesOnElementCG`, `NodesOnElementDG`, `NodesOnInterface`, `NodesOnInterfaceNum`

1.4. Visual demonstration example

Several example models are provided with the program and are listed in Section 8. One file entitled DEIPaPlotExample contains a workflow linking the mesh generation function in Section 2.4.A, insertion of couplers with DEIP in Section 1.3.B, batch mechanical analysis using `FEA_Program` in Section 2, and plotting of the results using `plotElemCont2` in Section 3.5. The user simply executes this script and then presses any key when the script pauses before the plots. The figures show the regions of elements, the axial stress field computed using DG couplers at the interfaces, and the axial stress field computed using CZ couplers at the interfaces.

2. FEA PROGRAM

A companion linear finite element program, `FEA_Program`, is provided with the `DEIProgram` in order to simplify the verification of the generated discontinuous element meshes. The structure of the program is modeled after the software FEAP originally developed by Robert Taylor [8].

Discussion of the input relevant to the analysis is provided in the context of the example file `Lel01_2d_T3`. The user is advised to consult this file alongside the descriptions below.

The program `FEA_Program` may be executed through two modes: (i) interactive or (ii) batch. Details on the batch mode will be provided in a future version. An example batch script `TestAll` is provided for testing the suite of examples with the `DEIProgram`. The interactive mode is the default execution mode, which is obtained either by typing "`FEA_Program`" into the command line of MATLAB or by opening the 'FEA_Program.m' script file and pressing the "Run" button in the Editor window. The program will then open a dialog box, similar to the one shown in Figure 2.1. A valid input file (described below) can then be selected, and the program will determine the solution to the linear finite element problem.

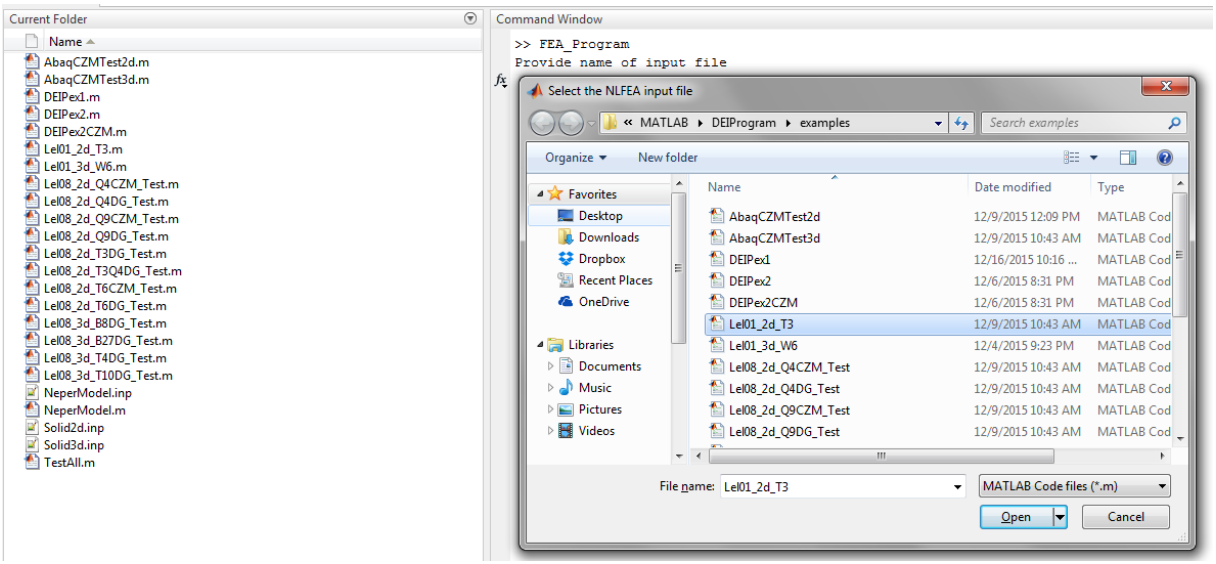


Figure 2.1. Input file selection for `FEA_Program`

2.1. Input file format

Similar to the `DEIProgram` in Section 1.2, `FEA_Program` is a script file which expects particular named arrays in order to execute the finite element analysis (FEA). These arrays define the geometry, loading and boundary conditions on the finite element model. For further discussions on relevant data structures and problem definitions for FEA, the user may consult published textbooks such as [9].

A simple, valid input file for the program is contained in `Lel01_2d_T3` which demonstrates the minimal required and optional arrays necessary for the analysis. The required arrays are highlighted in Figure 2.2; optional arrays are shown in Figure 2.3. The input file corresponds to a

plane strain rectangular domain meshed with two linear triangular elements and an applied pressure load on the top surface. The domain geometry is depicted in Figure 2.4.

```
%% Required Model Inputs

% Mesh Nodal Coordinates
Coordinates = [0 0
               40 0
               0 20
               40 20];

% Mesh Element Connectivities
NodesOnElement = [1 4 3 0
      1 2 4 0];
RegionOnElement = [1 1]';

% Mesh Material Properties
MateT = [1e6 0.25 1];
MatTypeTable = [1; 1; 0];

% Problem Size Parameters
ProbType = [length(Coordinates) size(NodesOnElement,1) size(MateT,1) ...
            2 2 size(NodesOnElement,2)];
```

Figure 2.2. Minimal data arrays required within input file for `FEA_Program`

```
%% Optional Model Inputs

% Mesh Boundary Conditions and Loads
NodeBC = [1 2 0
          2 1 0
          2 2 0];
numBC = 3;
NodeLoad = [3 2 -4
            4 2 -4];
% numNodalF = 2;
numNodalF = 0;
SurfacesL = [ 3 4 1 2    0     -0.2     0];
numSL = 1;
% numSL = 0;
```

Figure 2.3. Optional data arrays in input file for `FEA_Program`

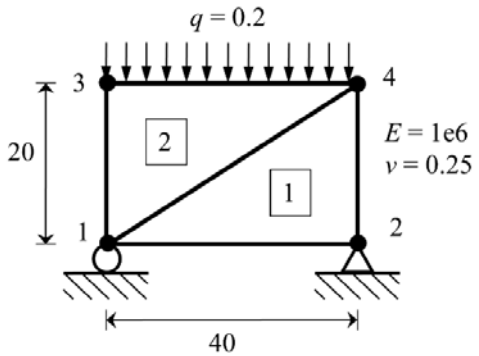


Figure 2.4. Problem description for linear finite element analysis

Several of the arrays and variables have identical meanings to those defined in Section 1.2 for `DEIProgram`. Several of the arrays are briefly described below.

```
ProbType:        = [numnp numel nummat ndm ndf nen]
numnp:           = number of nodes in the mesh (length(Coordinates))
numel:           = number of elements in the mesh (length(NodesOnElements))
nummat:          = number of materials (length(MateT))
ndm:             = number of spatial dimensions
ndf:             = number of degrees of freedom per node
nen:             = maximum number of nodes per element
Coordinates:     = table of mesh nodal coordinates defining the
                   geometry of the mesh; format of the table is as
                   follows:
                       Nodes  |              x-coord  y-coord
                       n1     | Coordinates = [x1     y1
                       n2     |                x2     y2
                       ...    |                ..     ..
                       nnumnp |                xnumnp ynumnp];
NodesOnElement:  = table of mesh connectivity information, specifying
                   how nodes are attached to elements and how materials
                   are assigned to elements; entries in the first nen
                   columns correspond to the rows of Coordinates
                   representing the nodes attached to element e;
                   entries in the last nen+1 column are rows from MateT
                   signifying the material properties assigned to
                   element e; format of the table is as follows:
                       Elements  |         n1    n2    n3    n4   mat
                       e1        | NonE = [e1n1  e1n2  e1n3  e1n4 e1mat
                       e2        |         e2n1  e2n2  e2n3  e2n4 e2mat
                       ...       |          ..    ..    ..    ..   ..
                       enumel    |         values for element numel   ];
MateT:           = table of mesh material properties for each distinct
                   set of material properties; these sets are
                   referenced by element e by setting the value of
                   RegionOnElement(e) to the row number of the desired
                   material set; format of the table is as follows:
                       Materials  |           E   v   t
                       mat1       |  MateT = [E1  v1  t1
                       mat2       |           E2  v2  t2
                       ...        |           ..  ..  ..];
MatTypeTable:    = [mat1 mat2 ...  List of materials, starting from 1
                    iel1 iel2 ...] Element type ID (from Section 2.3)
NodeBC:          = table of prescribed nodal displacement boundary
                   conditions; it contains lists of nodes, the
                   direction of the displacement prescribed (x=1, y=2),
                   and the value of the displacement (set 0 for fixed
                   boundary); the length of the table must match numBC,
                   otherwise an error will result;  format of the
                   table is as follows:
                       BCs  |            nodes direction value
                       bc1  |  NodeBC = [bc1n   bc1dir   bc1u
                       bc2  |            bc2n   bc2dir   bc2u
                       ...  |             ..     ..       .. ];
```

```
  NodeLoad:        = table of prescribed nodal forces; it contains lists
                     of nodes, the direction of the force prescribed
                     (x=1, y=2), and the value of the force; the length
                     of the table must match numNodalF,
                     otherwise an error will result; format of the table
                     is as follows:
                         Loads  |              nodes direction value
                         P1     |  NodeLoad = [ P1n    P1dir    P1P
                         P2     |               P2n    P2dir    P2P
                         ...    |               ..     ..       .. ];
SurfacesL:         = table of element facets on which to applied traction
                     loads; tractions are converted to nodal loads by
                     surface quadrature at the beginning of the analysis;
                     format of the table is as follows:
                      Loads  |                elem fac traction (x,y,z)
                      P1     | SurfacesL = [0 0 P1e P1f P1tx P1ty P1tz
                      P2     |              0 0 P1e P1f P1tx P1ty P1tz
                      ...    |              . . ..  ..  ..   ..   ..];
```

The facet numbering required for the array `SurfacesL` is listed in Section 7.2 for the various elements types provided in `FEA_Program`.

These data arrays may be generated within the designated input file using a variety of manners:

(a) Directly typed into the script file
(b) Inclusion of couplers generated by calling `DEIProgram` within the script
(c) Using the `block` utility scripts described in Section 2.4
(d) Loaded from an ABAQUS input file using the scripts described in Section 5
(e) Loaded from a Gmsh mesh file using the scripts described in Section 6
(f) Other I/O modules developed by the user

Minimal error checking features are provided within `FEA_Program` to ensure the size and shape of the arrays conform to the required format described above.

2.2. Analysis output

The program will then calculate the external force vector and stiffness matrix for the linear finite element analysis problem and solve the resulting linear system to obtain the nodal displacement vector. The user can optionally define flags in the input file to request additional output quantities to be computed:

`DHist = 1` for storing the nodal displacements in the array `Node_U_V`
`FHist = 1` for storing the nodal reactions in the array `ForcList`
`SHist = 1` for computing the projected/averaged stresses at the nodes in the array `StreList`
`SEHist = 1` for computing stresses at the first quadrature point in the array `StreListE`

These output arrays may be visualized in the open source software Paraview by using the provided `PostParaview` script discussed in Section 4. Also, the provided plotting modules can be used as discussed in Section 3. The results of the finite element analysis for the example `Lel01_2d_T3` are shown in Figure 2.5. The nodal solution vector and averaged nodal stresses are shown in Figure 2.6. The columns in `Node_U_V` are nodal values of displacements in the x and y direction for each node (row). The rows of `StreList` are nodal values of stress for each

node (column); stress ordering is xx, yy, xy, von Mises, maximum principal, minimum principal, and hydrostatic.

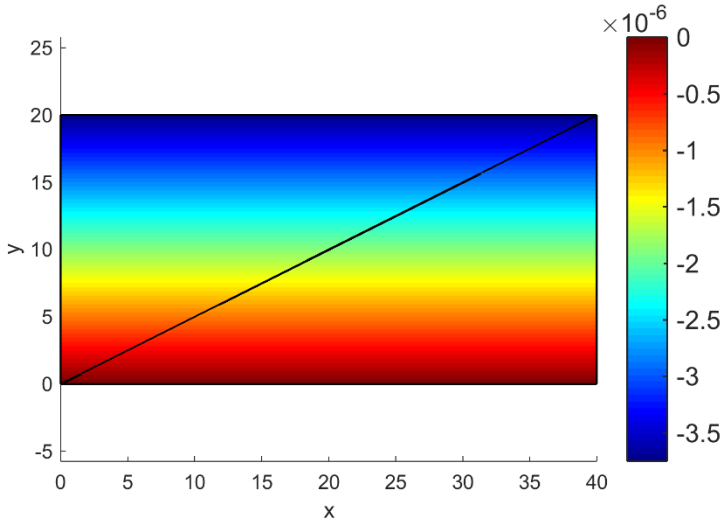


Figure 2.5. Vertical component of displacement field for solution of `Lel01_2d_T3`

```
Command Window

>> Node_U_V
Node_U_V =
   1.0e-05 *
  -0.250000000000000                   0
                   0                   0
  -0.250000000000000  -0.375000000000000
   0.000000000000000  -0.375000000000000
>> StreList
StreList =
                   0   0.000000000000000  -0.000000000000000                   0
  -0.200000000000000  -0.200000000000000  -0.200000000000000  -0.200000000000000
   0.000000000000000   0.000000000000000   0.000000000000000   0.000000000000000
   0.175594229214212   0.175594229214212   0.175594229214212   0.175594229214212
  -0.200000000000000  -0.200000000000000  -0.200000000000000  -0.200000000000000
   0.000000000000000   0.000000000000000  -0.000000000000000   0.000000000000000
  -0.083333333333333  -0.083333333333333  -0.083333333333333  -0.083333333333333
>>
```

Figure 2.6. Nodal displacements and stresses for solution of `Lel01_2d_T3`

2.3. Element library

A restricted set of element types corresponding to linear isotropic elasticity are provided with the program. These element types are sufficient for performing simple tests of the generated discontinuous meshes to ensure that the couplers have been inserted and the nodes properly duplicated. Other linear PDE-type elements could also be added into the program by adding files into the ‘program\ElemRout.m’ file. A sample template element and descriptions of the required operations performed by the element can be found in ‘element\BlankElem.m’.

Currently, supported element shapes include linear and quadratic isoparametric triangular, quadrilateral, tetrahedral, pyramid, wedge, and hexahedral elements, as shown in Section 7.1.

Each element in the mesh must be assigned to a material, and that material must be chosen from the element types in the current library. This definition is accomplished by assigning a material ID through the array `RegionsOnElement(elem) = mat`. Then, the element type associated with the material `mat` is defined through the entry `MatTypeTable(:,mat) = [mat iel]'`. The element type is defined through the value of `iel`, which is the identifier for the element type in the library. Additionally, the associated material parameters must be supplied in the array `MateT`.

The current elements in the library are as follows:

- Element type `iel=1` is a two or three dimensional isotropic elastic element. Material parameters are Young's modulus, Poisson's ratio, and thickness, `MateT=[E v t]`. `PSPS` is an optional flag that can be defined in the input file for designating plane stress '`s`' or plane strain '`n`'. Plane strain is the default for 2D.
- Element type `iel=7` is a two or three dimensional linear spring element, which corresponds to the initial linear stiffness of intrinsic cohesive zone elements. Material parameters are the elastic stiffness K along with the standard output from `FormCZ`, i.e. `MateT=[reg1 reg2 nel1 nel2 K]`.
- Element type `iel=8` is a two or three dimensional Discontinuous Galerkin element for isotropic linear elasticity. The formulation uses the stabilization parameters to define the numerical flux at the interface as a weighted average; see [10]. Material parameters are the standard output from `FormDG`, i.e. `MateT=[reg1 reg2 nel1 nel2]`.

2.4. Meshing utility modules

Several scripts are provided for generating simple meshes and to convert meshes consisting of one type of solid element into another prior to insertion of couplers.

2.4.A. Block2d and Block3d

Scripts are provided for generating tensor product meshes using the standard element types in 2D and 3D. These modules are `block2d` and `block3d`. They are based upon the subroutines provided with the FEAP software [8] and share similar syntax. The user is referred to the provided example files as well as the documentation for FEAP for usage of these scripts.

Example file: Lel08_2d_Q4DG_Test.m

2.4.B. Q4toT3 and Q8toT6

These scripts subdivide a single quadrilateral element into four triangular elements by inserting a node into the center of the quadrilateral element and forming the triangles in a union-jack pattern (edges of the triangles are along the diagonals of the quadrilateral).

Example file: PlateQ4toT3.m

2.4.C. Tri6toQua4

Routine to convert a quadratic 6 node triangular mesh into a 4 node quadrilateral mesh by subdividing triangular elements into 3 quadrilateral elements, adding edge nodes and one central node.

Example: Tri6toQua4_Test.m

2.4.D. Tet10toHex8

Routine to convert a quadratic 10 node tetrahedral mesh into an 8 node hexahedral mesh by subdividing tetrahedral elements into 4 hexahedral elements, adding face nodes and one central node. NOTE: the `DEIProgram3` should be called with an empty `InterTypes` array (no couplers inserted) in order to create the listing of facets between elements. These arrays are needed so that a single mid-face node can be created between each pair of tetrahedral elements.

Example: Tet10toHex8_Test.m

3. PLOTTING MODULE

Several plotting functions are provided for visualizing the mesh and the computed displacements/stresses from an analysis. These functions are general but are not highly optimized. Therefore, they should only be used for small to moderate sized meshes. The `PostParaview` function should be used for visualizing larger meshes in the open source program Paraview.

NOTE: All of the plotting functions are configured for solid elements *only*. The user is responsible for specifying the list of elements `elemlist` to only *include* the solid elements of the mesh and to *exclude* the interface couplers. Usually this is done by passing the number of elements from the continuous mesh, before the insertion of the couplers.

Each of the plotting functions are listed below, along with their argument list and example usage.

3.1. plotMesh2

```
Plot wire-frame of a 2-D mesh
Optional arguments are in {}, defaults are in ().

Default usage (for copy/paste):
plotMesh2(Coordinates,NodesOnElement,1,(1:size(NodesOnElement,1)),[1 0 0 0
0])

Inputs:
 Coordinates = nodal coordinates [x y]
 NodesOnElement = element connectivity [n1 n2 ... nnen]
 {PlotID} = [ModelID {clfyn subp numsubp}]
     ModelID = figure ID for plot window (1)
     clfyn = 1 for clearing figure before plot, retain otherwise (1)
     subp = subplot ID for current contour plot (1)
     numsubp = number of subplots for figure window (1)
 {elemlist} = list of specific elements to plot [e1 e2 ... en]' (1:numel)
 {MNE_ID} = [meshyn nodeyn elemyn {faceyn ecolyn}] (1 0 0 0 0)
     meshyn = 1 for plotting mesh lines, no mesh lines otherwise
     nodeyn = 1 for plotting node ID on figure, no ID on plot otherwise
     elemyn = 1 for plotting elem ID on figure, no ID on plot otherwise
     faceyn = 1 for plotting face with color, no face plot otherwise
     ecolyn = 1 for color-coding edges based on orientation, constant
              otherwise
 {varargin} = {types},{props1},{props2},...,{propsn}
              pairs of commands for figure, axes, etc.
     types = {'fig','node','mesh','cb','xlab','axes',...} is a list of
             all types of property pairs that follow, in the correct
             order
     props1 = {{'PropertyName1',PropertyValue1,'PropertyName2',... is a
             of all properties assigned to a given type
     Examples: varargin = {'title'},{'String','Make a visible title'}
               varargin = {'axes'},{'CLim',[-10 10]} % resets caxis
               varargin = {'surf'},{'FaceColor',[0 1 0]} % resets
                          facecolor
               varargin = {'node'},{'FontSize',32} % sets node ID font
               varargin = {'line'},{'LineWidth',2} % changes mesh lines
               varargin = {'xlab'},{'FontSize',18} % changes xlabel
               varargin = {'xlab','node'},{'FontSize',32},{'FontSize',18}
               varargin = 'xlab','FontSize',32 % one type doesn't need {}
```

Example corresponding to file `DEIPex2`:

```
plotMesh2(Coordinates,NodesOnElement,1,(1:320),[1 0 1 0 0])
```

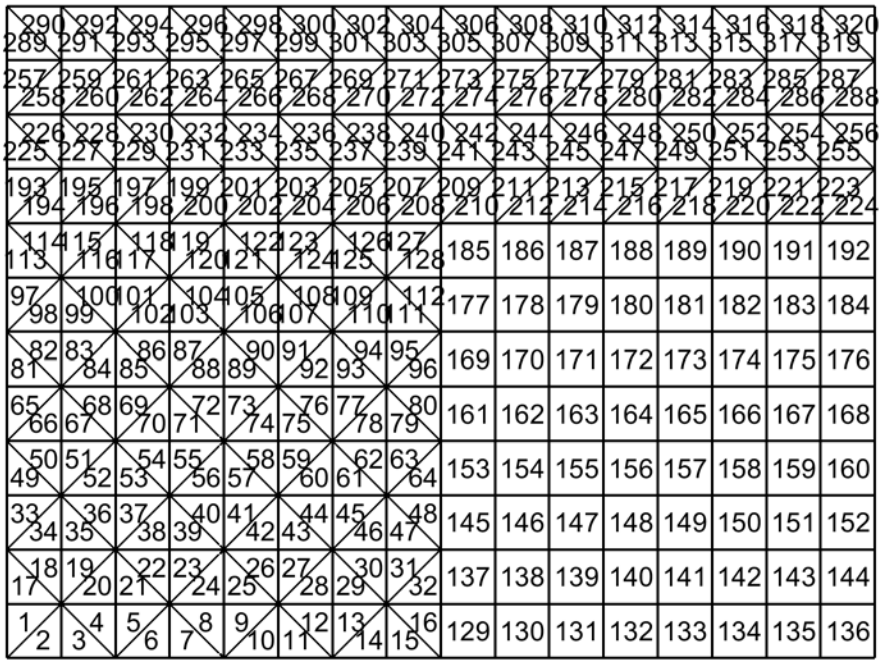


Figure 3.1. Mesh with element numbers

3.2. plotMesh3

```
Plot wire-frame of a 3-D mesh
Optional arguments are in {}, defaults are in ().

Default usage (for copy/paste):
plotMesh3(Coordinates,NodesOnElement,1,(1:size(NodesOnElement,1)),(1:size(Coo
rdinates,1))',[1 0 0 0 0])

Inputs:
 Coordinates = nodal coordinates [x y]
 NodesOnElement = element connectivity [n1 n2 ... nnen]
 {PlotID} = [ModelID {clfyn subp numsubp}]
     ModelID = figure ID for plot window (1)
     clfyn = 1 for clearing figure before plot, retain otherwise (1)
     subp = subplot ID for current contour plot (1)
     numsubp = number of subplots for figure window (1)
 {elemlist} = list of specific elements to plot [e1 e2 ... en]' (1:numel)
 {nodelist} = list of nodes on boundary of mesh (1:size(Coordinates,1))'
 {MNE_ID} = [meshyn nodeyn elemyn {faceyn fcolyn}] (1 0 0 0 0)
     meshyn = 1 for plotting mesh lines, no mesh lines otherwise
     nodeyn = 1 for plotting node ID on figure, no ID on plot otherwise
```

```
    elemyn = 1 for plotting elem ID on figure, no ID on plot otherwise
    faceyn = 1 for plotting face with color, no face plot otherwise
    fcolyn = 1 for color-coding faces based on orientation, constant
             otherwise
{varargin} = {types},{props1},{props2},...,{propsn}
             pairs of commands for figure, axes, etc.
    types = {'fig','node','mesh','cb','xlab','axes',...} is a list of
            all types of property pairs that follow, in the correct
            order
    props1 = {{'PropertyName1',PropertyValue1,'PropertyName2',... is a
            of all properties assigned to a given type
    Examples: varargin = {'title'},{'String','Make a visible title'}
              varargin = {'axes'},{'CLim',[-10 10]} % resets caxis
              varargin = {'surf'},{'FaceColor',[0 1 0]} % resets
                         facecolor
              varargin = {'node'},{'FontSize',32} % sets node ID font
              varargin = {'line'},{'LineWidth',2} % changes mesh lines
              varargin = {'xlab'},{'FontSize',18} % changes xlabel
              varargin = {'xlab','node'},{'FontSize',32},{'FontSize',18}
              varargin = 'xlab','FontSize',32 % one type doesn't need {}
```

Example corresponding to file `AbaqCZMTest3d`:

```
plotMesh3(Coordinates,NodesOnElement,1,(1:2),(1:size(Coordinates,1))',[1 0 1
0 0]); view(3)
```

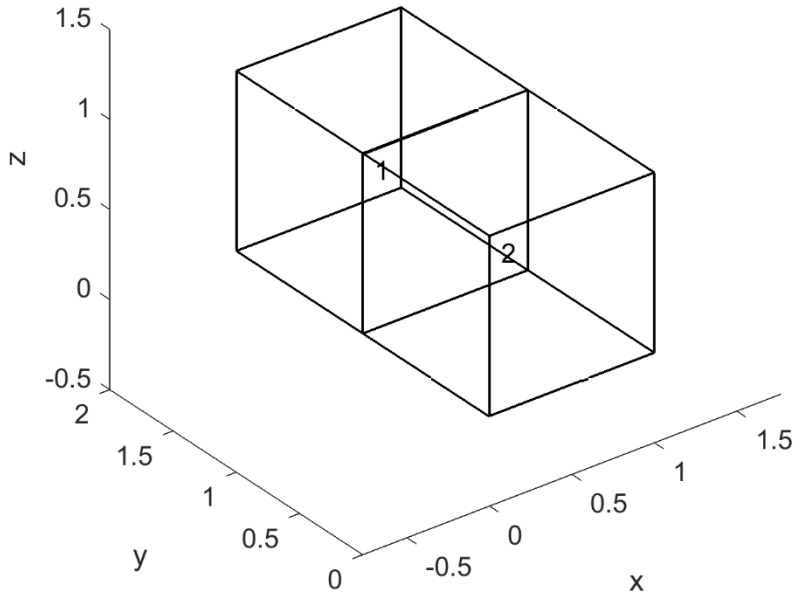


Figure 3.2. Mesh with element numbers

3.3. plotNodeCont2

```
Plot contour of nodal finite element field from a 2-D mesh
Optional arguments are in {}, defaults are in ().

Default usage (for copy/paste):
plotNodeCont2(Coordinates,Contour,NodesOnElement,1,(1:size(NodesOnElement,1))
,[1 0 0],0,[3 4 6 9 0])

Inputs:
 Coordinates = nodal coordinates [x y]
 Contour = nodal solution field [c]
 NodesOnElement = element connectivity [n1 n2 ... nnen]
 {PlotID} = [ModelID {clfyn subp numsubp}]
     ModelID = figure ID for plot window (1)
     clfyn = 1 for clearing figure before plot, retain otherwise (1)
     subp = subplot ID for current contour plot (1)
     numsubp = number of subplots for figure window (1)
 {elemlist} = list of specific elements to plot [e1 e2 ... en]' (1:numel)
 {MNE_ID} = [meshyn nodeyn elemyn] (1 0 0)
     meshyn = 1 for plotting mesh lines, no mesh lines otherwise
     nodeyn = 1 for plotting node ID on figure, no ID on plot otherwise
     elemyn = 1 for plotting elem ID on figure, no ID on plot otherwise
 {zcont3D} = 1 for using Contour as the z-coordinate for the plot,
             creates a flat contour plot otherwise (0)
 {nelPn} = [nelP3 nelP4 nelP6 nelP9 contP]
     nelP3 = number of nodes with pressure dofs for T3 elements (3)
     nelP4 = number of nodes with pressure dofs for Q4 elements (4)
     nelP6 = number of nodes with pressure dofs for T6 elements (6)
     nelP9 = number of nodes with pressure dofs for Q9 elements (9)
     contP = 1 if solution field is pressure (uses lower-order shape
             functions to plot), uses default shape functions otherwise
             (0)
 {varargin} = {types},{props1},{props2},...,{propsn}
              pairs of commands for figure, axes, etc.
     types = {'fig','node','mesh','cb','xlab','axes',...} is a list of
             all types of property pairs that follow, in the correct
             order
     props1 = {{'PropertyName1',PropertyValue1,'PropertyName2',... is a
             of all properties assigned to a given type
     Examples: varargin = {'cb'},{'FontSize',16}
               varargin = {'title'},{'String','Make a visible title'}
               varargin = {'axes'},{'CLim',[-10 10]} % resets caxis
               varargin = {'node'},{'FontSize',32} % sets node ID font
               varargin = {'line'},{'LineWidth',2} % changes mesh lines
               varargin = {'xlab'},{'FontSize',18} % changes xlabel
               varargin = {'xlab','node'},{'FontSize',32},{'FontSize',18}
               varargin = 'xlab','FontSize',32 % one type doesn't need {}
```

Example corresponding to file `DEIPex2`, x-displacement field on deformed configuration:

```
plotNodeCont2(Coordinates+Node_U_V,Node_U_V(:,1),NodesOnElement,1,(1:320),[1
0 0],0,[3 4 6 9 0])
```

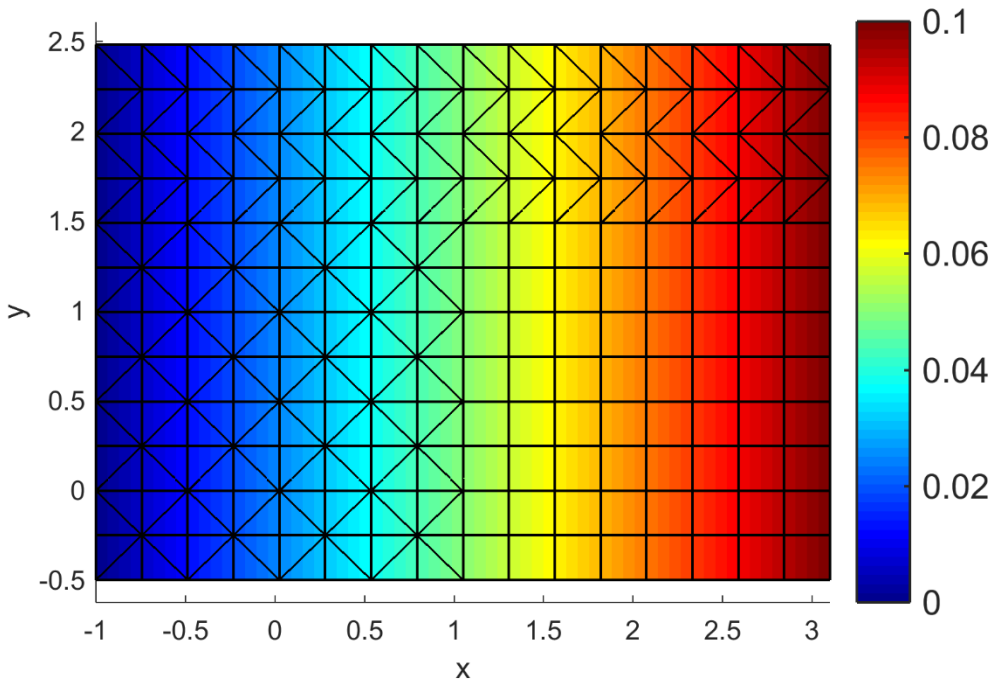


Figure 3.3. Horizontal displacement on deformed configuration

3.4. plotNodeCont3

```
Plot contour of nodal finite element field from a 3-D mesh
Optional arguments are in {}, defaults are in ().

Default usage (for copy/paste):
plotNodeCont3(Coordinates,Contour,NodesOnElement,1,(1:size(NodesOnElement,1))
,(1:size(Coordinates,1))',[1 0 0],[4 8 10 27 0])

Inputs:
 Coordinates = nodal coordinates [x y]
 Contour = nodal solution field [c]
 NodesOnElement = element connectivity [n1 n2 ... nnen]
 {PlotID} = [ModelID {clfyn subp numsubp}]
     ModelID = figure ID for plot window (1)
     clfyn = 1 for clearing figure before plot, retain otherwise (1)
     subp = subplot ID for current contour plot (1)
     numsubp = number of subplots for figure window (1)
 {elemlist} = list of specific elements to plot [e1 e2 ... en]' (1:numel)
 {nodelist} = list of nodes on boundary of mesh (1:size(Coordinates,1))
 {MNE_ID} = [meshyn nodeyn elemyn] (1 0 0)
     meshyn = 1 for plotting mesh lines, no mesh lines otherwise
     nodeyn = 1 for plotting node ID on figure, no ID on plot otherwise
     elemyn = 1 for plotting elem ID on figure, no ID on plot otherwise
 {nelPn} = [nelP3 nelP4 nelP6 nelP9 contP]
     nelP3 = number of nodes with pressure dofs for T4 elements (4)
```

```
    nelP4 = number of nodes with pressure dofs for B8 elements (8)
    nelP6 = number of nodes with pressure dofs for T10 elements (10)
    nelP9 = number of nodes with pressure dofs for T27 elements (27)
    contP = 1 if solution field is pressure (uses lower-order shape
            functions to plot), uses default shape functions otherwise
            (0)
{varargin} = {types},{props1},{props2},...,{propsn}
             pairs of commands for figure, axes, etc.
    types = {'fig','node','mesh','cb','xlab','axes',...} is a list of
            all types of property pairs that follow, in the correct
            order
    props1 = {{'PropertyName1',PropertyValue1,'PropertyName2',... is a
            of all properties assigned to a given type
    Examples: varargin = {'cb'},{'FontSize',16}
              varargin = {'title'},{'String','Make a visible title'}
              varargin = {'axes'},{'CLim',[-10 10]} % resets caxis
              varargin = {'node'},{'FontSize',32} % sets node ID font
              varargin = {'line'},{'LineWidth',2} % changes mesh lines
              varargin = {'xlab'},{'FontSize',18} % changes xlabel
              varargin = {'xlab','node'},{'FontSize',32},{'FontSize',18}
              varargin = 'xlab','FontSize',32 % one type doesn't need {}
```

Example corresponding to file `AbaqCZMTest3d`, y-displacement field on deformed configuration:

```
plotNodeCont3(Coordinates+Node_U_V,Node_U_V(:,2),NodesOnElement,1,(1:2),(1:si
ze(Coordinates,1))',[1 0 0],[4 8 10 27 0])
>> view(3)
```

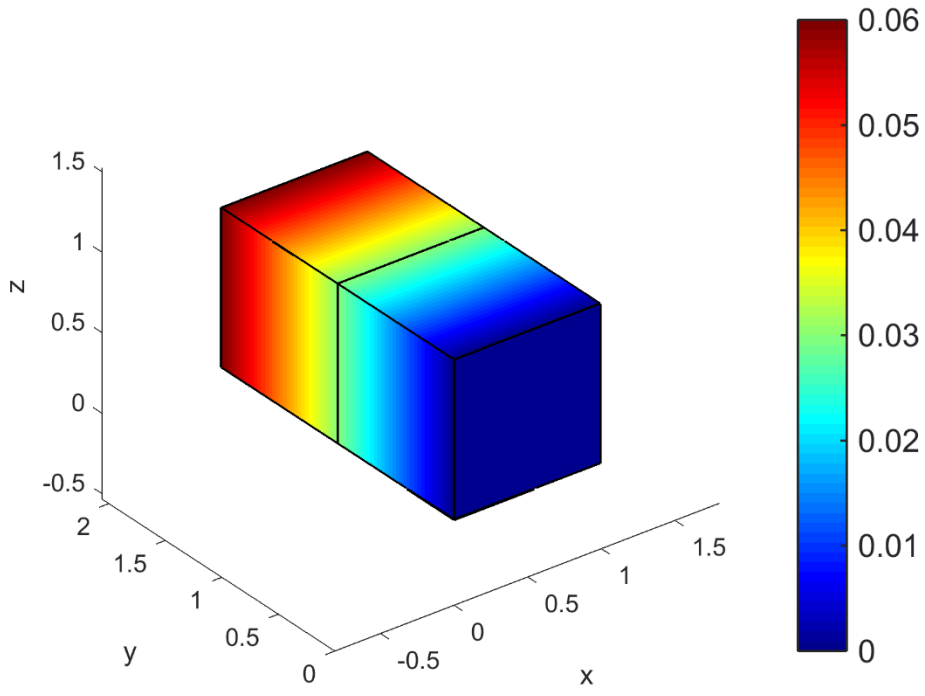


Figure 3.4. Displacement in y direction on deformed configuration

3.5. plotElemCont2

```
Plot contour of elemental finite element field from a 2-D mesh
Optional arguments are in {}, defaults are in ().

Default usage (for copy/paste):
plotElemCont2(Coordinates,Contour,NodesOnElement,1,(1:size(NodesOnElement,1))
,[1 0 0],varargin)

Inputs:
 Coordinates = nodal coordinates [x y]
 Contour = elemental solution field [c]
 NodesOnElement = element connectivity [n1 n2 ... nnen]
 {PlotID} = [ModelID {clfyn subp numsubp}]
     ModelID = figure ID for plot window (1)
     clfyn = 1 for clearing figure before plot, retain otherwise (1)
     subp = subplot ID for current contour plot (1)
     numsubp = number of subplots for figure window (1)
 {elemlist} = list of specific elements to plot [e1 e2 ... en]' (1:numel)
 {MNE_ID} = [meshyn nodeyn elemyn] (1 0 0)
     meshyn = 1 for plotting mesh lines, no mesh lines otherwise
     nodeyn = 1 for plotting node ID on figure, no ID on plot otherwise
     elemyn = 1 for plotting elem ID on figure, no ID on plot otherwise
 {varargin} = {types},{props1},{props2},...,{propsn}
              pairs of commands for figure, axes, etc.
     types = {'fig','node','mesh','cb','xlab','axes',...} is a list of
             all types of property pairs that follow, in the correct
             order
     props1 = {{'PropertyName1',PropertyValue1,'PropertyName2',... is a
             of all properties assigned to a given type
     Examples: varargin = {'cb'},{'FontSize',16}
               varargin = {'title'},{'String','Make a visible title'}
               varargin = {'axes'},{'CLim',[-10 10]} % resets caxis
               varargin = {'node'},{'FontSize',32} % sets node ID font
               varargin = {'line'},{'LineWidth',2} % changes mesh lines
               varargin = {'xlab'},{'FontSize',18} % changes xlabel
               varargin = {'xlab','node'},{'FontSize',32},{'FontSize',18}
               varargin = 'xlab','FontSize',32 % one type doesn't need {}
```

Example corresponding to file DEIPex2:, region ID with element number:

```
plotElemCont2(Coordinates,RegionOnElement,NodesOnElement,1,(1:320),[1 0 1])
```

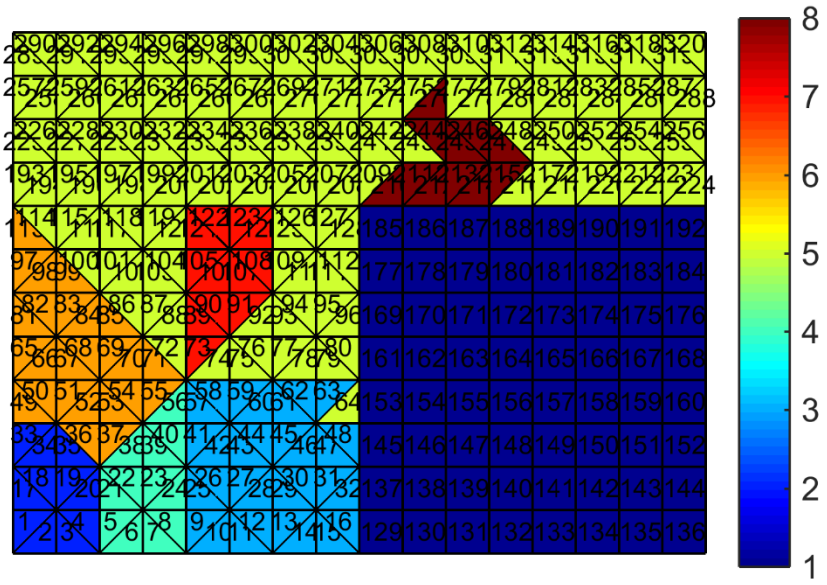


Figure 3.5. Mesh with element numbers

3.6. plotElemCont3

```
Plot contour of elemental finite element field from a 3-D mesh
Optional arguments are in {}, defaults are in ().

Default usage (for copy/paste):
plotElemCont3(Coordinates,Contour,NodesOnElement,1,(1:size(NodesOnElement,1))
,(1:size(Coordinates,1)),[1 0 0],varargin)

Inputs:
 Coordinates = nodal coordinates [x y]
 Contour = nodal solution field [c]
 NodesOnElement = element connectivity [n1 n2 ... nnen]
 {PlotID} = [ModelID {clfyn subp numsubp}]
     ModelID = figure ID for plot window (1)
     clfyn = 1 for clearing figure before plot, retain otherwise (1)
     subp = subplot ID for current contour plot (1)
     numsubp = number of subplots for figure window (1)
 {elemlist} = list of specific elements to plot [e1 e2 ... en]' (1:numel)
 {nodelist} = list of nodes on boundary of mesh (1:size(Coordinates,1))
 {MNE_ID} = [meshyn nodeyn elemyn] (1 0 0)
     meshyn = 1 for plotting mesh lines, no mesh lines otherwise
     nodeyn = 1 for plotting node ID on figure, no ID on plot otherwise
     elemyn = 1 for plotting elem ID on figure, no ID on plot otherwise
 {varargin} = {types},{props1},{props2},...,{propsn}
              pairs of commands for figure, axes, etc.
     types = {'fig','node','mesh','cb','xlab','axes',...} is a list of
```

```
        all types of property pairs that follow, in the correct
        order
props1 = {{'PropertyName1',PropertyValue1,'PropertyName2',... is a
        of all properties assigned to a given type
Examples: varargin = {'cb'},{'FontSize',16}
          varargin = {'title'},{'String','Make a visible title'}
          varargin = {'axes'},{'CLim',[-10 10]} % resets caxis
          varargin = {'node'},{'FontSize',32} % sets node ID font
          varargin = {'line'},{'LineWidth',2} % changes mesh lines
          varargin = {'xlab'},{'FontSize',18} % changes xlabel
          varargin = {'xlab','node'},{'FontSize',32},{'FontSize',18}
          varargin = 'xlab','FontSize',32 % one type doesn't need {}
```

Example corresponding to file `AbaqCZMTest3d`, yy-stress field with color bar legend reset to [2.9 3.1]:

```
plotElemCont3(Coordinates,StreListE(2,:)',NodesOnElement,1,(1:2),(1:size(Coor
dinates,1)),[1 0 0],{'axes'},{'CLim',[2.9 3.1]}); view(3)
```

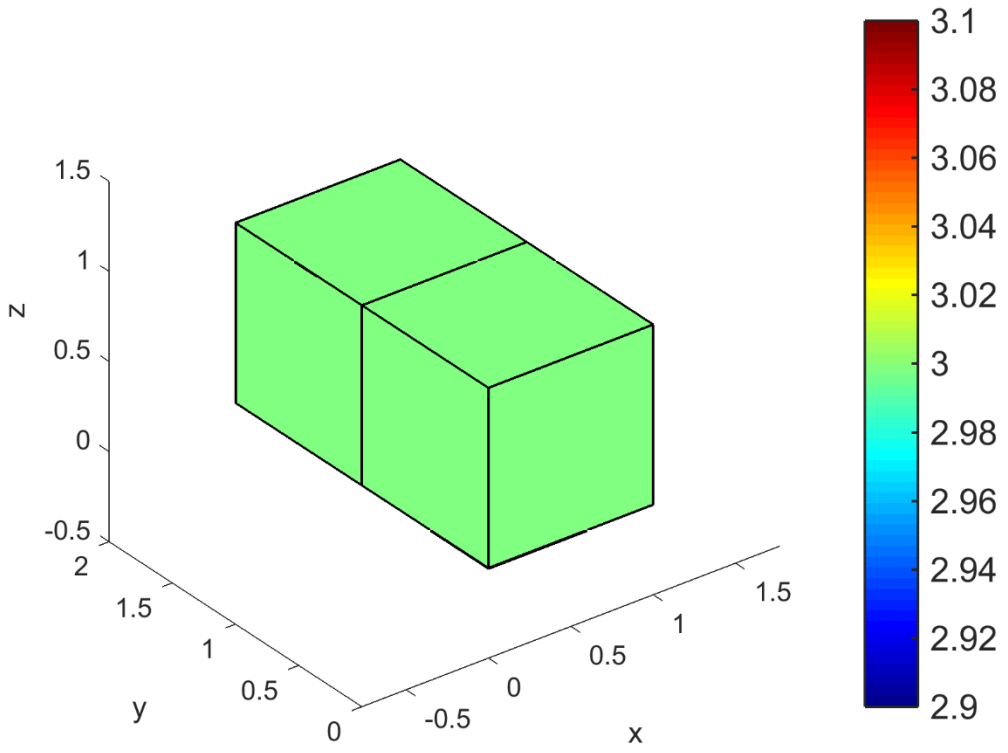


Figure 3.6. Stress yy contour on undeformed configuration

3.7. plotBC

```
Plot boundary conditions of a mesh; blue means 0 value and red means non-zero
Optional arguments are in {}, defaults are in ().

Default usage (for copy/paste):
plotBC(Coordinates,NodeBC,1)
```

```
Inputs:
 Coordinates = nodal coordinates [x y]
 NodeBC = boundary condition codes [node dof value]
 {PlotID} = [ModelID {clfyn subp numsubp}]
     ModelID = figure ID for plot window (1)
     clfyn = 1 for clearing figure before plot, retain otherwise (0)
     subp = subplot ID for current contour plot (1)
     numsubp = number of subplots for figure window (1)
               otherwise
 {varargin} = {types},{props1},{props2},...,{propsn}
              pairs of commands for figure, axes, etc.
     types = {'fig','node','mesh','cb','xlab','axes',...} is a list of
             all types of property pairs that follow, in the correct
             order
     props1 = {{'PropertyName1',PropertyValue1,'PropertyName2',... is a
             of all properties assigned to a given type
     Examples: varargin = {1},{'Color',[0 1 0]} % resets color of markers
                          for dof=1 boundary conditions
               varargin = {2},{'MarkerSize',20} % resets markersize for
                          dof=2 boundary conditions
               varargin = {'title'},{'String','Make a visible title'}
               varargin = {'axes'},{'CLim',[-10 10]} % resets caxis
               varargin = {'xlab'},{'FontSize',18} % changes xlabel
               varargin = {'xlab','node'},{'FontSize',32},{'FontSize',18}
               varargin = 'xlab','FontSize',32 % one type doesn't need {}
```

Example corresponding to file `DEIPex2`:

```
plotMesh2(Coordinates,NodesOnElement,1,(1:320),[1 0 0 0 0]);
plotBC(Coordinates,NodeBC,1)
```

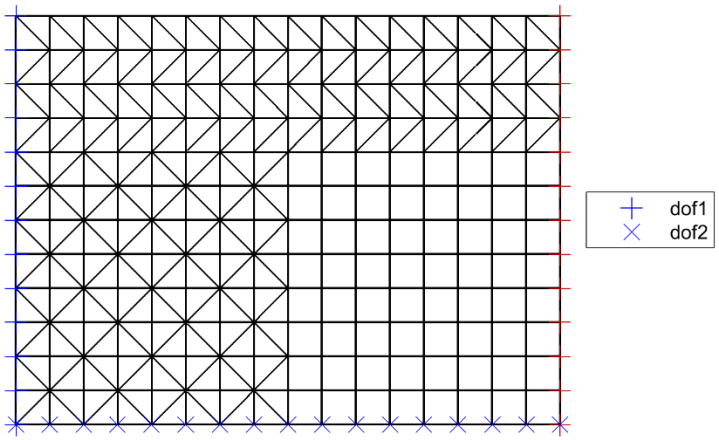


Figure 3.7. Applied boundary conditions

4. PARAVIEW MODULE

Paraview is a popular open source visualization program for finite element analysis. For large geometry files, this program is more suitable compared to the modules provided in Section 3. Therefore, a script `PostParaview` is provided for exporting the results from `DEIProgram` and `FEA_Program` into VTK files compatible with Paraview.

README for the program:

```
0. Specify in the variable "userdefaultdir" the path to the root directory
   that you plan to put all of your output folders
1. Execute file with F5 or play button or from the command window
2. A window opens to specify the output folder for the current VTK files;
   to create a new folder within the "userdefaultdir" directory quickly,
   follow the steps below
3. Click the "Make New Folder" button in the window, which will create a
   new directory in the current folder (currently "userdefaultdir")
4. Press "Enter" key to confirm folder name
5. Press "Enter" key to select the new folder
6. Script then executes and writes the VTK files
```

File name generated will be out0001.vtk for the solid elements in the mesh and cout0001.vtk for the couplers in the mesh. Each of these files may be loaded in Paraview.

There are several lines in the control box for the elements to output, nodes to output, and so forth. These lines usually do not need to be modified. Note that `numSI` is the total number of couplers contained in `NodesOnElement`, and `numel` is assumed to be the sum of the number of solid elements and the number of couplers. Also, `nenPV` is the number of nodes per element of the single type of element in the mesh. Usually this value is assumed to be `nen/2`, since the DG type couplers in the mesh will have twice the number of nodes compared to the solid elements. Further comments are provided within the script to help define the meaning of the parameters. The control box is shown in Figure 4.1.

Several flags are provided to designate the fields that are written to the VTK files.

```
strflag = 0; % flag for stress postprocessing
st2flag = 0; % flag for stress vectors/scalars
serflag = 0; % flag for elemental stress postprocessing
se2flag = 0; % flag for elemental stress vectors/scalars
disflag = 1; % flag for displacement
mflag = 1; % flag for printing region colors
cflag = 1; % flag for couplers
cseflag = 0; % flag for coupler element stresses
```

The script `PostParaview` can be executed from MATLAB either with or without performing a finite element analysis. Thus, the mesh can simply be visualized without computing the displacement field. If only the mesh is desired and the region numbers, then only `mflag` and `cflag` should be set to 1. After a finite element analysis using `FEA_Program`, then the other data will also exist and may be designated for output to the vtk file.

```
FILE | NAVIGATE | EDIT | BREAKPOINTS | RUN
21    %%%%%%%%%%%%%%%%%%%%%%%%%%%%% BEGIN CONTROL BOX %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
22
23    % Steps to output: starting step, increment to step, and last step
24 -  stepstart = 1;
25 -  stepinc = 1;
26 -  stepstop = 1;
27
28 -  startp = stepstart; % first step # for VTK file
29 -  incp = 1; % first step # for VTK file
30 -  stopp = stepstop/1; % first step # for VTK file
31
32    % Nodes to output
33 -  nodestart = 1;
34 -  nodeend = numnp;
35
36    % Elements to output
37 -  elemstart = 1;
38 -  elemend = numel-numSI;
39
40    % Number of nodes per element; assumed constant for all elements elemstart
41    % to elemend
42 -  nenPV = nen/2; % nen to use for Paraview file
43
44    % Couplers to output
45 -  coupstart = numel-numSI+1;
46 -  coupend = numel;
47
48 -  strflag = 0; % flag for stress postprocessing
49 -  st2flag = 0; % flag for stress vectors/scalars
50 -  serflag = 0; % flag for elemental stress postprocessing
51 -  se2flag = 0; % flag for elemental stress vectors/scalars
52 -  disflag = 1; % flag for displacement
53 -  mflag = 1; % flag for printing region colors
54 -  cflag = 1; % flag for couplers
55 -  cseflag = 0; % flag for coupler element stresses
56
57 -  dfoldername = 'ParaOut'; %default name of folder for files to be placed
58 -  userdefaultdir = pwd; %list your default output root directory here
59 -  foldername = uigetdir(userdefaultdir,'Select folder to write VTK files');
60 -  filename = 'out'; %prefNodesOnElement filename (e.g. 'out' -> out01.vtk)
61 -  header = 'Elasticity'; %text at top of VTK files
62
63    %%%%%%%%%%%%%%%%%%%%%%%%%%%%% END CONTROL BOX %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 4.1. Control box for `PostParaview`

5. ABAQUS READ/WRITE MODULE

The finite element analysis software ABAQUS is a commonly used program in industry and academia. Therefore, utility scripts have been written to provide an interface with .inp files generated by this program, so that interface couplers may be directly inserted into meshes and subsequent analyses performed.

`AbaqusInputReader`: Read in a file from ABAQUS
`AbaqusInputWriter`: Write an ABAQUS .inp; modifications to the mesh can be included, such as insertion of interface couplers. Generated file, so long as boundary conditions and material properties and step conditions have been defined, is analysis suitable (meaning they can be imported in ABAQUS CAE or executed from the terminal using ABAQUS Standard).

DISCLAIMER: Documentation of these scripts are not provided at this time. These scripts were developed and tested for simple (single PART) ABAQUS models. The user may need to extend/modify the scripts to treat complex or multi-PART models. Generally, the `AbaqusInputReader` script is reading the .inp file line by line and storing text that cannot be interpreted; this test is reprinted into the written input file by `AbaqusInputWriter` without modification.

Relevant example files are provided for 2D and 3D.

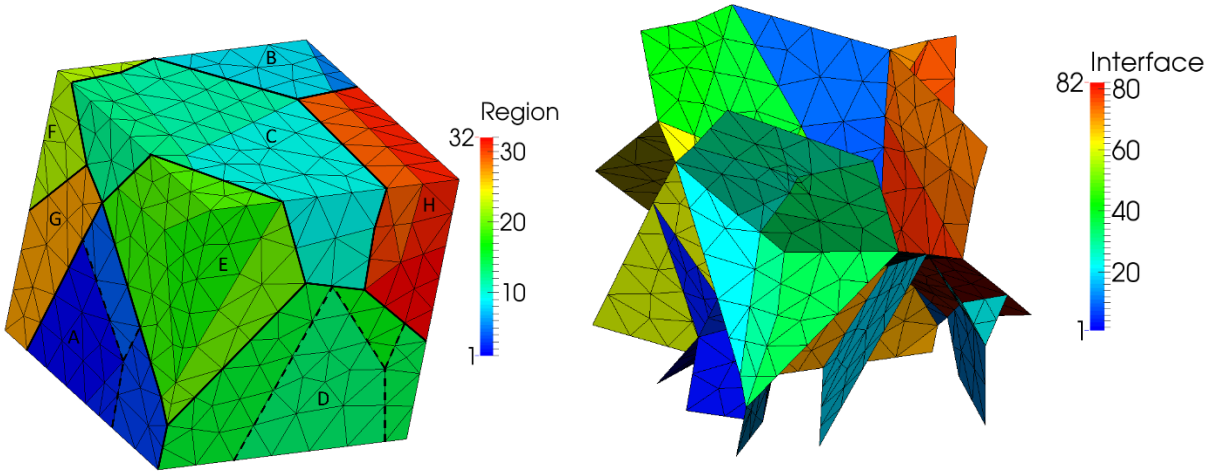
- AbaqCZMTest2d
- AbaqCZMTest3d

Both of these files can be executed, and new versions of the analysis files, which originally do not contain interface couplers, will be generated that incorporate the ABAQUS element types "`COH2D4`" and "`COH3D8`". The interface elements are zero-thickness, and a nonlinear analysis can be performed using these files.

Note that the module currently assumes that the nodal coordinates and element connectivity along with the section assignments are defined in the "PART" portion of the file rather than the "ASSEMBLY" portion. Such a file is usually generated if the section assignments and mesh are generated on a "Dependent Instance". If only the nodes and elements are provided without the material properties and sections, then this issue will not apply.

Another example showing more complex geometry for a polycrystalline domain is contained in `NeperModel`. This input file was generated using the software package Neper [11]; see Figure 5.1. This provides a test case for selectively inserting couplers along preferential interfaces in a complex 3D geometry. A patch test on the domain can subsequently be performed with `FEA_Program`. This file is also an example showing that Section Assignments are not required for the `AbaqusInputReader` to function, only that `*Elset` should be provided elsewhere so that the regions in the mesh can be defined so as to allow coupler insertion.

A second version of the polycrystalline example, `NeperModel_2`, demonstrates how the higher-level functions can be used to insert couplers, and how region-dependent material parameters can be assigned to groups of interface couplers.



(a) (b)

Figure 5.1. Polycrystalline domain containing quadratic tetrahedral elements: (a) solid mesh; (b) inserted couplers between all grains and certain subgrains

Finally, other examples are provided to demonstrate how to modify portions of the connectivity of the solid mesh, update the boundary conditions, and export these changes into the ABAQUS .inp file. The rectangular domain mesh contains 4-noded elements that are first converted to 3-noded elements, and then intraface couplers are inserted in the upper-right quadrant. Uniform displacement boundary conditions are applied, and a nearly constant stress state is produced by all models (exactly constant for the analyses without cohesive couplers).

- PlateTest
- PlateQ4toT3
- PlateCOHnoNotch

An extension of this problem shows how to assign different cohesive properties to different sets of couplers based on the region ID on either side of the interface.

- PlateCOHNotch

Note that a small artificial stiffness is assigned to the cohesive elements in the upper-right-hand corner of the model. This reduces the bulk apparent stiffness of that region, such that when a uniform compressive displacement is applied on the left edge of the domain, the upper half of the domain carries less compressive stress than the lower half (Figure 5.2).

5.1. Mechanical analysis using .inp file

The generated .inp file from `AbaqusInputWriter`, so long as boundary conditions and material properties and step conditions have been defined, is analysis suitable (meaning they can be imported in ABAQUS CAE or executed from the terminal using ABAQUS Standard).

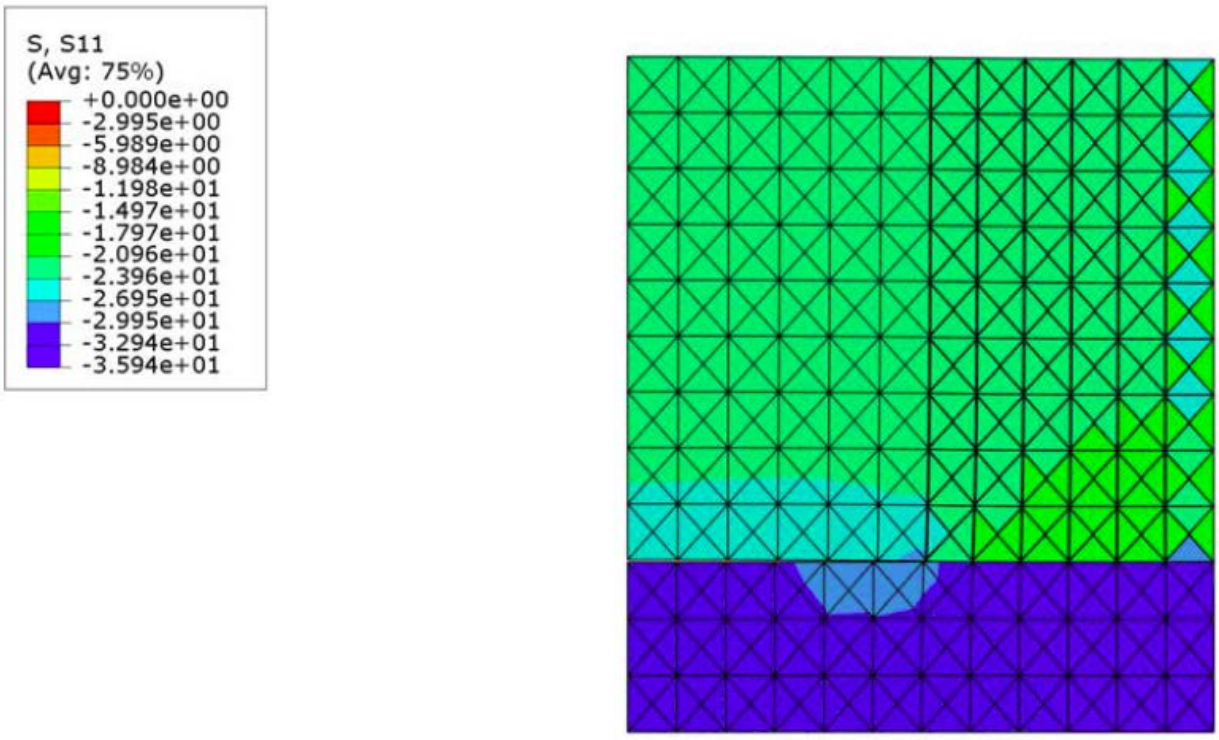


Figure 5.2. Stress σ_{xx} from PlateCOHNotch.inp showing effect of cohesive elements on bulk apparent stiffness of the domain

6. GMSH READ/WRITE MODULE

The mesh generation software Gmsh (http://gmsh.info/) is used commonly in academia and industry to provide geometry for mechanical analysis in programs such as Code_Aster (https://www.code-aster.org). Therefore, utility scripts have been written to provide an interface with .msh files generated by this program, so that interface couplers may be directly inserted into meshes and subsequent analyses performed.

`GmshInputReader`: Read in a file from Gmsh
`GmshInputWriter`: Write an Gmsh .msh; modifications to the mesh can be included, such as insertion of interface couplers and extra sets of nodes. This mesh file can be combined with a command file describing boundary conditions, material properties, solution steps and parameters, etc. to perform a finite element analysis.

DISCLAIMER: Documentation of these scripts are not provided at this time. These scripts were developed and tested for several Gmsh models conforming to MSH ASCII standards 1.0 and 2.0. Current treatment is limited to the single instances of sections '$Nodes', '$Elements', and '$PhysicalNames' in that order. The user may need to extend/modify the scripts to treat complex or multi-part models. Generally, the `GmshInputReader` script is reading the .msh file line by line and storing text that cannot be interpreted; this test is reprinted into the written input file by `GmshInputWriter` without modification.

Relevant example files are provided for 2D and 3D.

- AsterCZMTest2d
- NeperGmshCZ (3d)

Both of these files can be executed, and new versions of the mesh files, which originally do not contain interface couplers, will be generated that incorporate Gmsh element types for the interface couplers (respectively 4-node quadrilaterals and 6-node wedges). The interface elements are zero-thickness.

The dimension of the model is automatically determined by examining the types of elements in the .msh file. The largest spatial dimension is taken as 'solid', and a set is `solidelemID` created that lists the element IDs that have this spatial dimension. Those elements are grouped into `NodesOnElement` in the same order as `solidelemID`. Regions of elements are determined by the 'physical entity' given in the tag of the element and converted in `RegionOnElement` using the listing `RegToGMgroup`. For example, in the file `Solid2d.msh` used by `AsterCZMTest2d`, the 3rd element [3 3 3 6 6 0 10 14 9 4] is a 2d quadrilateral in physical group 6 which is converted through `RegToGMgroup(2)=6` into region 2 and becomes the 3rd element in `NodesOnElement(3,:)=[10 14 9 4]` with `RegionOnElement(3)=2`. Groups of 'points' (element type 15) are also extracted into a cell array `Nsetholder`, so that these lists may be modified by DEIP.

6.1. Mechanical analysis using Gmsh input (Code_Aster)

The example `AsterCZMTest2d` can be analyzed in `FEA_Program` using the linear (spring-like) cohesive element `iel=7` mentioned in Section 2.3.

However, the original and generated .msh files can also be analyzed using Code_Aster. The provided .comm command files are structured to perform a 10 step nonlinear analysis with an incrementally increasing axial displacement on a plane stress 2d bar. When cohesive couplers are inserted, the critical stress is reached at step 4 and damage evolves during the remaining steps until complete separation at step 10. These analyses may be executed using the batch scripts provided with Code_Aster (for example, in Ubuntu Linux):

/path/to/Code_aster_frontend-20170/bin/as_run Solid2d.export

/path/to/Code_aster_frontend-20170/bin/as_run Solid2dPlusCZM.export

Note that the two command files are almost identical and differ only by the inclusion of the material properties and element group associated with the cohesive couplers, set 'GM7'. These sets are automatically numbered by Code_Aster when converting from Gmsh files (see file u7.01.31.pdf in its Documentation). The array `physCZ` is produced by `GmshInputWriter` to help the user identify these sets. For example, for `AsterCZMTest2d`, the 3rd material corresponds to the CZ couplers at the interface and is written to physical entity 7 in the output .msh file and is recorded as `physCZ(:,1)=[3; 7]`. Future extensions are planned for DEIP to assist in the creation of the modified .comm files.

Another example showing more complex geometry for a polycrystalline domain is contained in `NeperGmshDG`. This input file was generated using the software package Neper [11] as an example complex 3D geometry. A patch test on the domain can subsequently be performed with `FEA_Program`.

7. ELEMENT TEMPLATES

Several standard finite elements for two and three dimensional analysis are currently implemented in this program. Other element types may be added at a future date.

Figures are provided to show the local node numbering assigned to the elements in the program. The connectivity of elements provided by external software should be rearranged so as to match these templates.

7.1. Element node numbering

For each figure, the solid element, cohesive zone coupler, and discontinuous Galerkin coupler are shown together. Regarding the couplers, the region with the lower region ID is always assigned as the "top" side of the interface in each of the figures, so the nodes from this solid element always appear numbered first within the connectivity template for the coupler.

7.1.A. Two dimensional elements

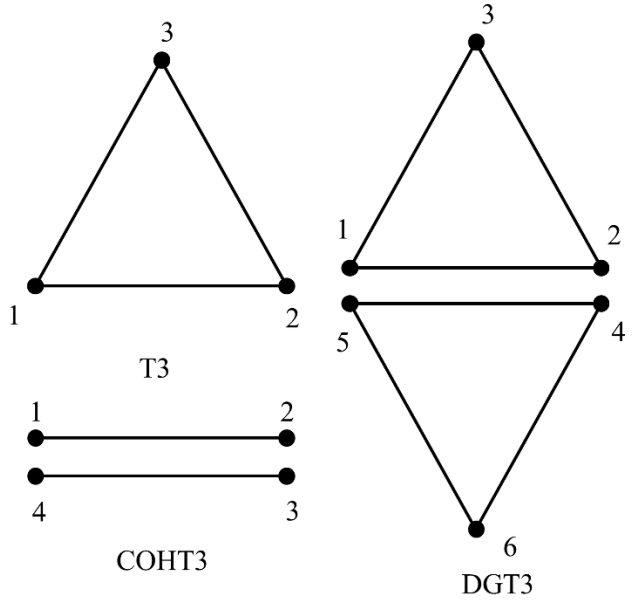


Figure 7.1. Linear triangular element and couplers

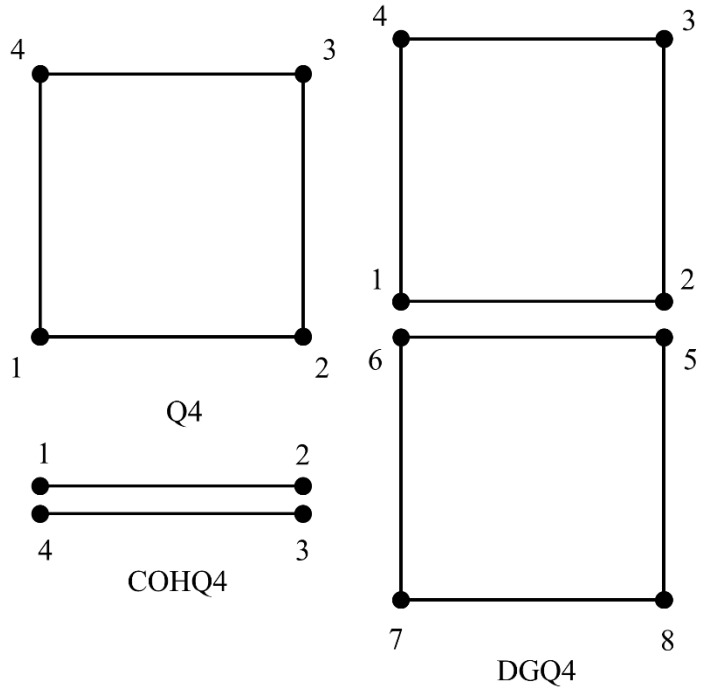


Figure 7.2. Bilinear quadrilateral element and couplers

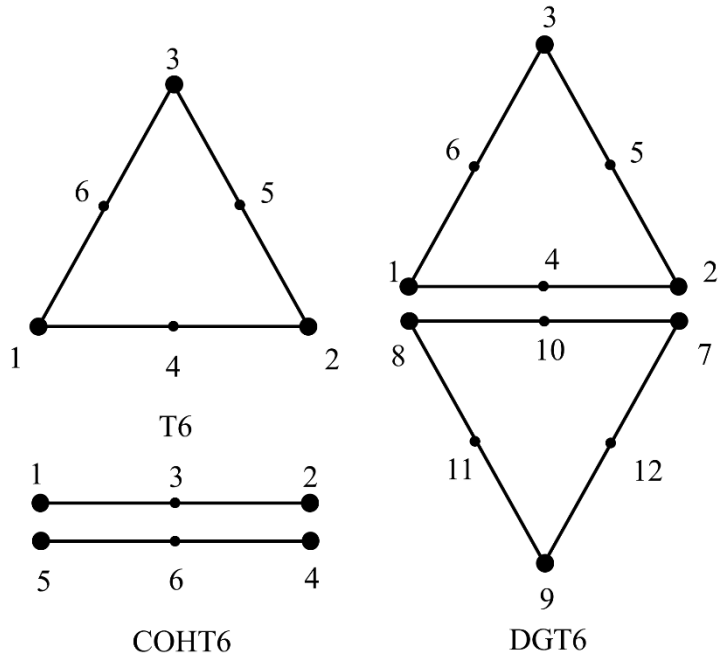


Figure 7.3. Quadratic triangular element and couplers

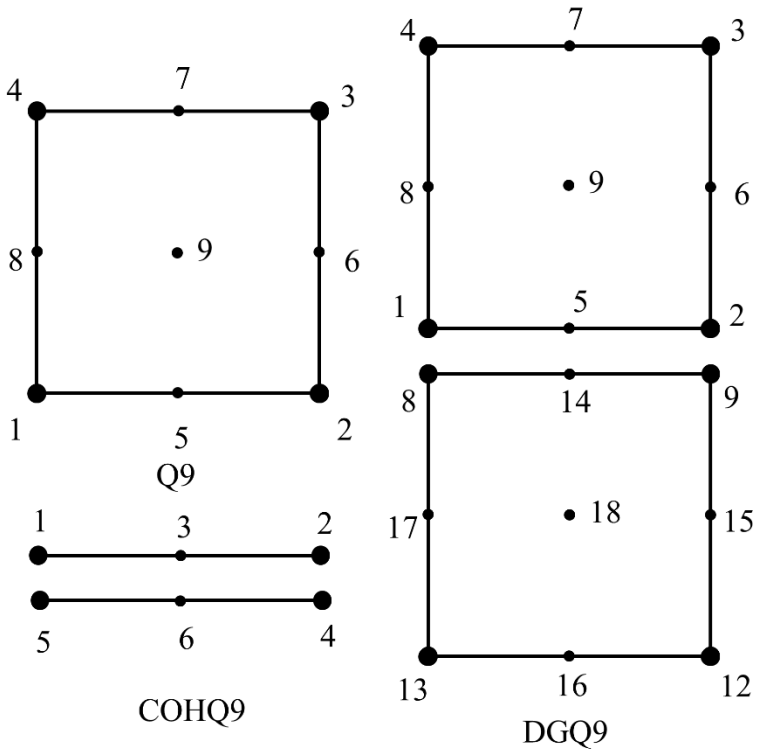


Figure 7.4. Biquadratic element and couplers

7.1.B. Three dimensional elements

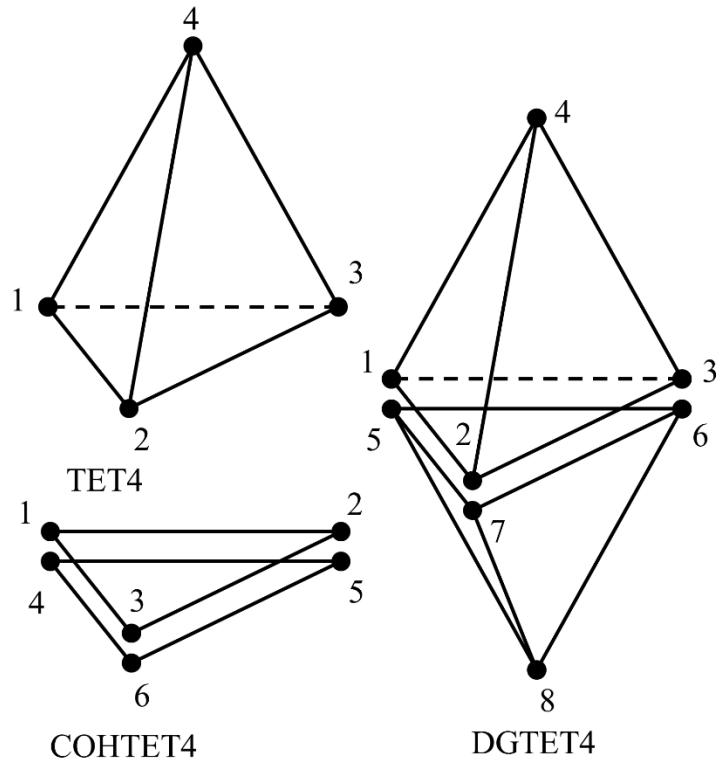


Figure 7.5. Linear tetrahedral element and couplers

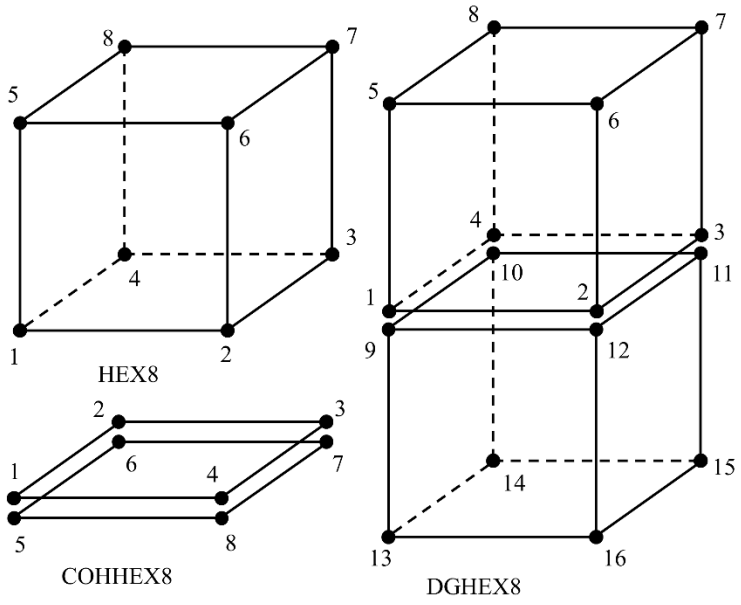


Figure 7.6. Trilinear hexahedral element and couplers

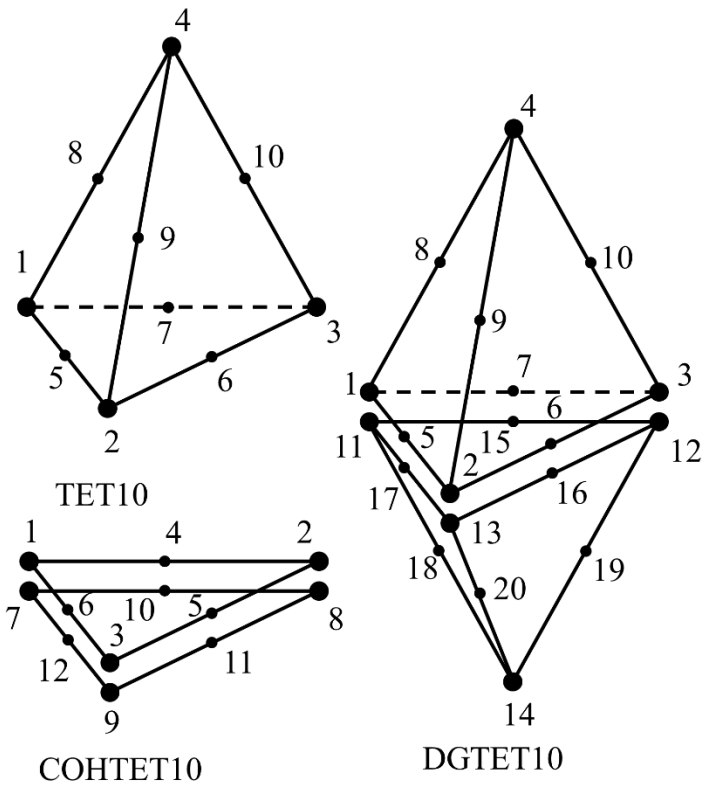


Figure 7.7. Quadratic tetrahedral element and couplers

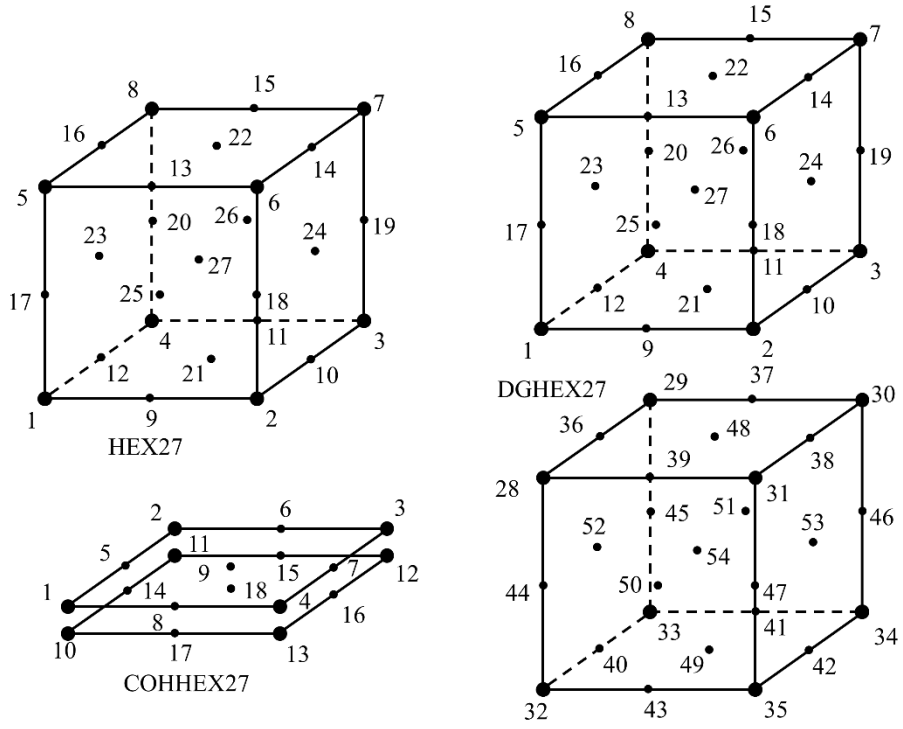


Figure 7.8. Triquadratic hexahedral element and couplers

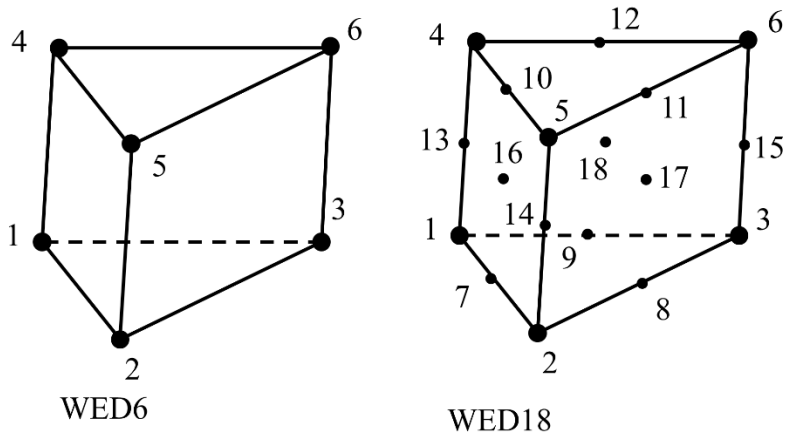


Figure 7.9. Linear and quadratic wedge elements

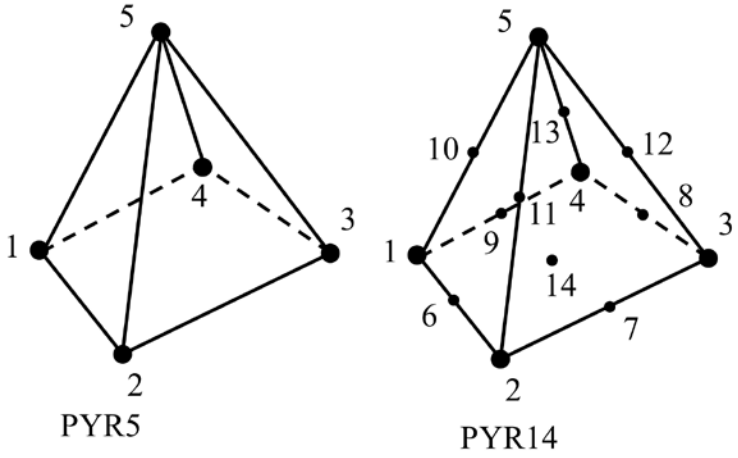


Figure 7.10. Linear and quadratic pyramid elements

Note: for wedge and pyramid elements, DG couplers not implemented. Cohesive couplers correspond to the number of nodes on the facet, to connect to tetrahedral or hexahedral elements

7.2. Element facet numbering

Local facet numbers are assigned to the facets of elements based on the template node numbering. These identifiers are needed for assigning surface tractions and for indicating the facet of an element that is adjacent to a particular coupler. The numbering of facets is shown in Figure 7.11 and Figure 7.12.

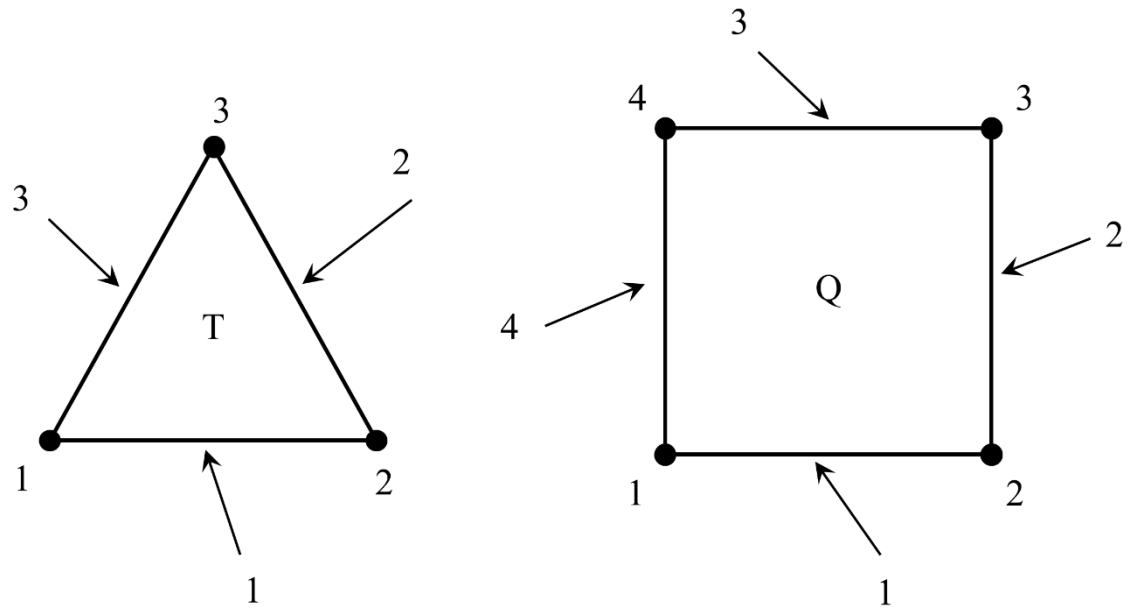


Figure 7.11. Facet numbering for 2D elements

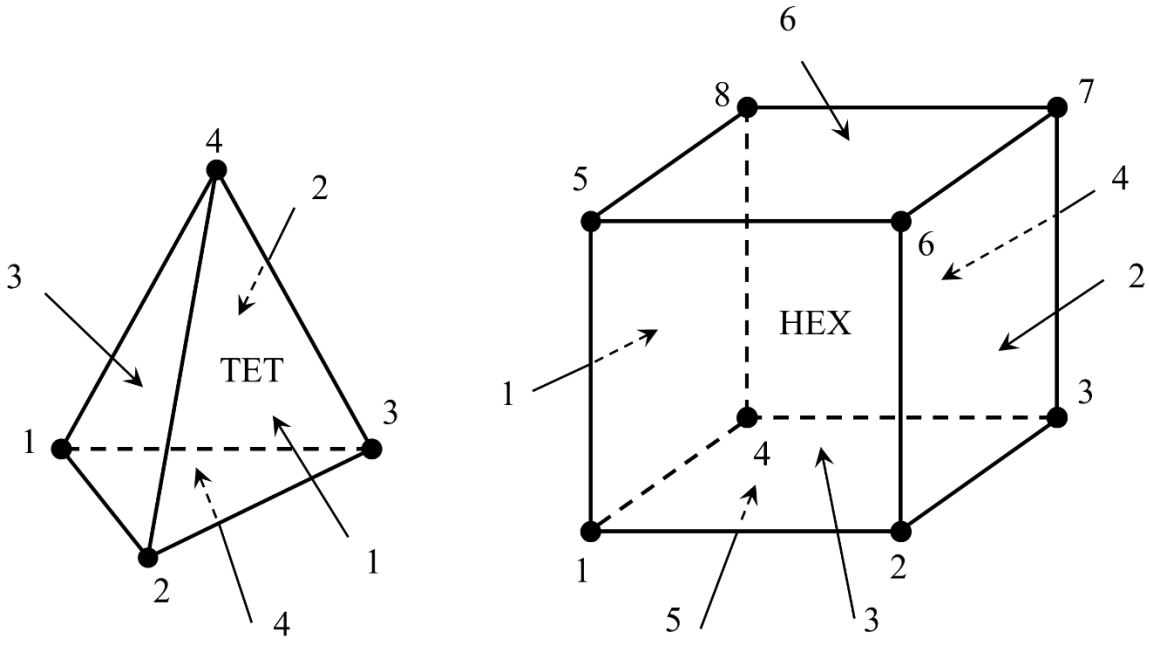


Figure 7.12. Facet numbering for 3D elements

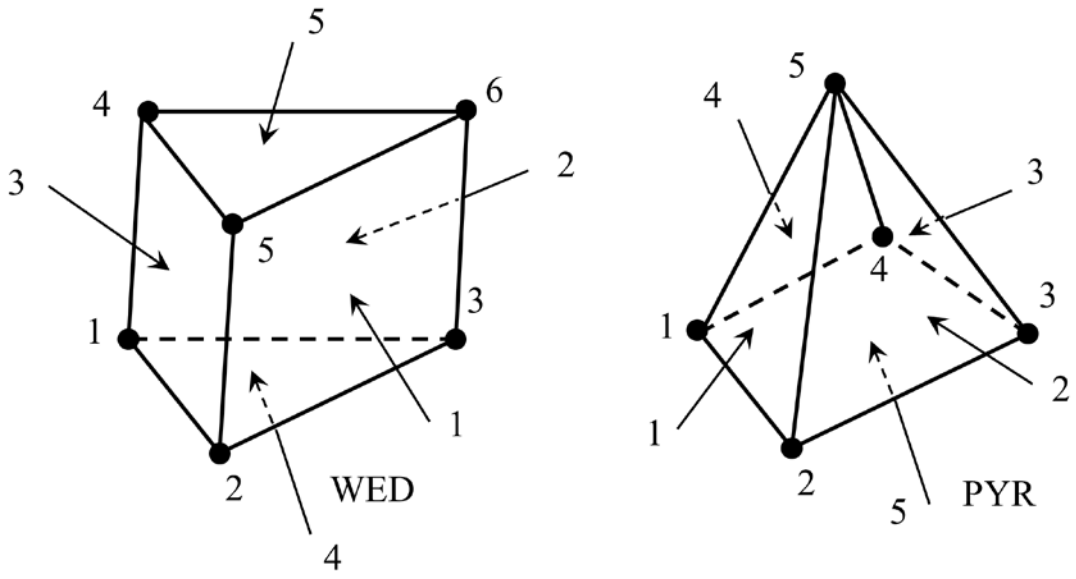


Figure 7.13. Facet numbering for 3D transition elements

8. LIST OF EXAMPLE FILES

A brief description is provided for the examples contained with the source program.

- TestAll: Batch script to execute all of the example files in series.

8.1. Tutorial

- DEIPaPlotExample: generation, analysis, and visualization workflow script. Corresponds to Figure 5 in [1].
- DEIPex1: Simple example of insertion of couplers using DEIProgram2. Corresponds to Figure 1 in [1].
- DEIPex1_2: Simple example of insertion of couplers using DEIProgram2. Corresponds to Figure 1 in [1]. Uses higher-level function calls.
- DEIPex2: Moderate example of insertion of couplers using DEIProgram2. Corresponds to Figure 5 in [1]. Uses DG couplers.
- DEIPex2CZM: Moderate example of insertion of couplers using DEIProgram2. Corresponds to Figure 5 in [1]. Uses CZ couplers.
- Lel01_2d_T3: Simple 2 element test problem. Demonstrates the minimum data required for FEA Program.
- Lel01_3d_T10: Test for quadratic tetrahedral element.
- Lel01_3d_P5: Test for pyramid element.
- Lel01_3d_P14: Test for quadratic pyramid element.
- Lel01_3d_W6: Test for wedge element.
- Lel01_3d_W18: Test for quadratic wedge element.
- Tet10toHex8_Test: Patch test for subdivision of cubic-shaped domain of quadratic tetrahedral elements into linear hexahedral elements.
- Tet10toHex8DG_Test: Patch test for subdivision of cubic-shaped domain of quadratic tetrahedral elements into linear hexahedral elements; includes DG couplers along certain interfaces.
- Tri6toQua4_Test: Patch test for rectangular domain of quadratic triangular elements converted to linear quadrilateral elements with DG couplers along interfaces between regions.

8.2. ABAQUS

- AbaqCZMTest2d: Demonstration of ABAQUS reader/writer for 2d quadrilateral mesh.
- AbaqCZMTest3d: Demonstration of Abaqus reader/writer for 3d hexahedral mesh.
- PlateTest: Rectangular domain for sequence of analyses with intraface couplers; demonstration of ABAQUS reader.
- PlateQ4toT3: Conversion of elements in rectangular domain from quadrilaterals to triangles; demonstration of ABAQUS reader/writer and shows the additional data arrays to be modified for generating the .inp file.
- PlateCOHnoNotch: Insertion of intraface couplers into upper-right hand quadrant of rectangular domain, application of cohesive zone couplers in ABAQUS; demonstration of ABAQUS reader/writer and shows the additional data arrays to be modified for generating the .inp file.

8.3. Cohesive Zone (FEA_CZ)

- Lel08_2d_Q4CZM_Test: Patch test for rectangular domain of Q4 elements with CZ

couplers along interfaces between regions.

- Lel08_2d_Q9CZM_Test: Patch test for rectangular domain of Q9 elements with CZ couplers along interfaces between regions.
- Lel08_2d_T6CZM_Test: Patch test for rectangular domain of T6 elements with CZ couplers along interfaces between regions.

8.4. Discontinuous Galerkin (FEA_DG)

- Lel08_2d_Q4DG_Test: Patch test for rectangular domain of Q4 elements with DG couplers along interfaces between regions.
- Lel08_2d_Q9DG_Test: Patch test for rectangular domain of Q9 elements with DG couplers along interfaces between regions.
- Lel08_2d_T3DG_Test: Patch test for rectangular domain of T3 elements with DG couplers along interfaces between regions.
- Lel08_2d_T3Q4DG_Test: Moderate example of insertion of couplers. Less complex version of interfaces in domain; insertion of DG couplers on region interfaces
- Lel08_2d_T6DG_Test: Patch test for rectangular domain of T6 elements with DG couplers along interfaces between regions.
- Lel08_3d_B8DG_Test: Patch test for prismatic domain of B8 elements with DG couplers along interfaces between regions.
- Lel08_3d_B27DG_Test: Patch test for prismatic domain of B27 elements with DG couplers along interfaces between regions.
- Lel08_3d_T4DG_Test: Patch test for prismatic domain of T4 elements with DG couplers along interfaces between regions.
- Lel08_3d_T10DG_Test: Patch test for prismatic domain of T10 elements with DG couplers along interfaces between regions.
- Lel08_3d_W6DG_Test1: Test for wedge elements, a domain with two wedge elements and one cohesive coupler.
- Lel08_3d_W6DG_Test2: Test for wedge elements, a domain with four wedge elements and cohesive couplers between all adjoining faces.

8.5. GMSH

- AsterCZMTest2d: Demonstration of GMSH reader/writer for 2d quadrilateral mesh; equivalent to AbaqCZMTest2d.
- NeperGmshCZ: Demonstration of GMSH reader/writer for 3d tetrahedral mesh. Generated using Neper program: http://neper.sourceforge.net/. Polycrystalline domain consisting of 10 grains. Cohesive couplers are inserted and written into a new .msh file.
- NeperGmshDG: Demonstration of GMSH reader for 3d tetrahedral mesh. Generated using Neper program: http://neper.sourceforge.net/. Polycrystalline domain consisting of 10 grains. DG couplers are inserted so that a patch test can be performed.

8.6. Models with multiple regions and interface properties (Multi_Material)

- NeperModel: Demonstration of ABAQUS reader for 3d tetrahedral mesh. Generated using Neper program: http://neper.sourceforge.net/. Polycrystalline domain consisting of 32 grains; 8 sets of macro-grains exist with 4 inside of each.
- NeperModel_2: Demonstration of ABAQUS reader for 3d tetrahedral mesh. Generated using Neper program: http://neper.sourceforge.net/. Polycrystalline domain consisting of 32 grains; 8 sets of macro-grains exist with 4 inside of each. Uses higher-level function

calls and assigns different (artificial) properties to each interface.

- PlateCOHNotch: Insertion of intraface couplers into upper-right hand quadrant of rectangular domain, application of cohesive zone couplers in ABAQUS; additional node duplication along a horizontal line in the mesh to create a notch; demonstration of ABAQUS reader/writer and shows the additional data arrays to be modified for generating the .inp file. Capable of assigning different properties to different interface/regions.

9. REFERENCES

1. Truster TJ. Discontinuous element insertion algorithm. *Advances in Engineering Software* 2015, submitted. Pre-print: http://trace.tennessee.edu/utk_civipubs/17/

2. Pandolfi A, Ortiz M. Solid modeling aspects of three-dimensional fragmentation. *Engineering with Computers* 1998; **14**(4):287-308.

3. Nguyen VP. An open source program to generate zero-thickness cohesive interface elements. *Advances in Engineering Software* 2014; **74**:27-39.

4. Park K, Paulino GH. Computational implementation of the PPR potential-based cohesive model in ABAQUS: Educational perspective. *Engineering Fracture Mechanics* 2012; **93**:239-262.

5. Arnold DN, Brezzi F, Cockburn B, Marini LD. Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM journal on numerical analysis* 2002; **39**(5):1749-1779.

6. Liu R, Wheeler MF, Dawson CN. A three-dimensional nodal-based implementation of a family of discontinuous Galerkin methods for elasticity problems. *Computers & Structures* 2009; **87**(3):141-150.

7. Masud A, Truster TJ, Bergman LA. A unified formulation for interface coupling and frictional contact modeling with embedded error estimation. *International Journal for Numerical Methods in Engineering* 2012; **92**(2):141-177.

8. Taylor RL. *FEAP - A Finite Element Analysis Program, User Manual*. University of California: Berkeley, 2013.

9. Hughes TJR. *The finite element method: Linear static and dynamic finite element analysis*, Dover: Mineola, NY, 2000.

10. Truster TJ, Masud A. Primal interface formulation for coupling multiple PDEs: A consistent derivation via the Variational Multiscale method. *Computer Methods in Applied Mechanics and Engineering* 2014; **268**:194-224.

11. Quey R, Dawson PR, Barbe F. Large-scale 3D random polycrystals for the finite element method: Generation, meshing and remeshing. *Computer Methods in Applied Mechanics and Engineering* 2011; **200**(17–20):1729-1745.