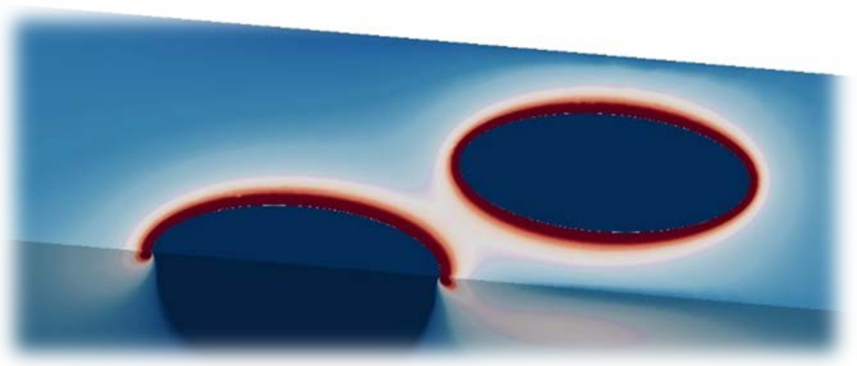
`int_defects` – User Guide



Disclaimer: *This software discussed in this document is provided without any warranty. Its use in structural integrity assessment or for any purpose (technical, scientific or otherwise) is entirely at the user's own risk.*

This document last modified: 15/08/2019 by Harry Coules
Current version of **`int_defects`*:*** v1.2.0
Contact: harry.coules@bristol.ac.uk

Contents

1. Toolbox summary

1.1 Overview

`int_defects` is a toolbox for MATLAB designed to help engineers investigate the interactions between co-planar crack-like flaws in structures. It uses the Abaqus Finite Element Analysis (FEA) package to create and solve models of single flaws and pairs of flaws in plates and pipes of finite thickness. The results can be used to predict the initiation of fracture or plastic collapse. `int_defects` allows large parametric series of models of interacting cracks to be generated and solved automatically. It also contains functions for automatic post-processing, calculation of interaction factors and plotting of the results. `int_defects` is free and open-source software distributed under the terms of the MIT license.

1.2 Intended applications

The `int_defects` toolbox was designed to aid the development of structural integrity assessment procedures such as R6 [1] and BS 7910 [2]. These procedures contain simple criteria for determining when crack-like structural flaws are close enough to have an effect on each other. When they are judged to interact, the procedure takes this interaction into account eg. by mandating the re-characterisation of a pair of flaws as a single, larger enclosing flaw. Ideally, these interaction criteria and recharacterisation rules should be valid for a wide range of flaw geometries, materials and loading conditions. To test them, a large number of FE models need to be formulated, solved and post-processed. `int_defects` provides an easy and quick way to perform the linear-elastic, elastic-plastic and elastic-perfectly-plastic FE analyses required for the development of reliable flaw interaction criteria.

`int_defects` can also be used to quickly formulate and solve FE models of specific co-planar cracks on a case-by-case basis. This is useful when analysing a specific flaw geometry and loading case, eg. as part of fitness-for-service assessment of a specific structure. In such cases, `int_defects` can also be used to perform a series of models to analyse the sensitivity of the crack-driving force to the flaw proximity or other factors.

There are also basic scientific/technical applications of `int_defects`. For example, it can be used to determine general relationships between fracture parameters (eg. SIF and T-stress) and flaw shape/size/proximity.

2. Capabilities

`int_defects` is used for analysing crack-like semi-elliptical surface flaws and crack-like elliptical embedded flaws. The flaws must be in the through-wall plane of a flat plate, or in the axial-radial plane of a pipe. For pipes, the cracks may be on the internal surface, external surface, or embedded (i.e. non-surface-breaking). Single flaws or interacting pairs of flaws can be analysed. The flaws may have arbitrary dimensions and relative positions. The plate/pipe may also have arbitrary dimensions. `int_defects` can perform the following types of analysis:

1. Linear-elastic analysis to determine the crack tip Stress Intensity Factor (SIF), as a function of position on the crack tip line(s).
2. Linear-elastic analysis to determine the level of (elastic) interaction for two flaws which are located close to each other.
3. Elastic-perfectly-plastic analysis to determine the Local Limit Load (LLL) and/or Net Section Limit Load (NSLL) for a flawed structure.
4. Elastic-plastic analysis to determine the J-integral as a function of load and of position on the crack tip line.

When analysing a plate geometry, the loading state can be defined using remotely-applied loads, moments and combinations of the two. For a pipe geometry, an internal pressure load can be used. For both plate and pipe geometries, a load in the plane of the crack can also be defined to produce biaxial crack loading. In pipes, this is used frequently to simulate the axial stress of a closed-ended pipe. For linear elastic models it is also possible to define a crack face pressure load to simulate any arbitrary through-wall distribution of stress: by Bueckner's superposition principle [3–5], in a linear material a crack face pressure distribution will produce an identical SIF at the crack tip line to an equivalent through-thickness stress distribution.

As well as models of defect pairs, `int_defects` can automatically generate finite element models of *single* semi-elliptical surface defects and elliptical embedded defects. Therefore, in addition to its primary use for investigating defect interaction, `int_defects` can also be used for parametric studies of single cracks.

`int_defects` is currently limited to analysing co-planar semi-elliptical surface flaws and elliptical embedded flaws in a flat plate or a pipe. For a pipe geometry, the cracks must be in the axial-radial plane.

In general, if two flaws are co-planar they will have a more adverse effect on each other than if one flaw was out-of-plane and/or tilted relative to the other, although this is not a universal rule in the case of ductile fracture. Therefore, this toolbox can be used to give a conservative estimate of the significance of interaction in most cases. The use of enclosing ellipses and semi-ellipses to characterise arbitrary flaws is recommended in the R6 procedure.

The set of single flaw/flaw pair geometries that can be investigated using `int_defects` is shown in Figure 1. In all cases, the model is symmetric about the plane containing the crack features. The plate (or pipe) wall thickness and width can be defined by the user; to simulate an infinitely-wide plate the user can define arbitrarily wide plate.

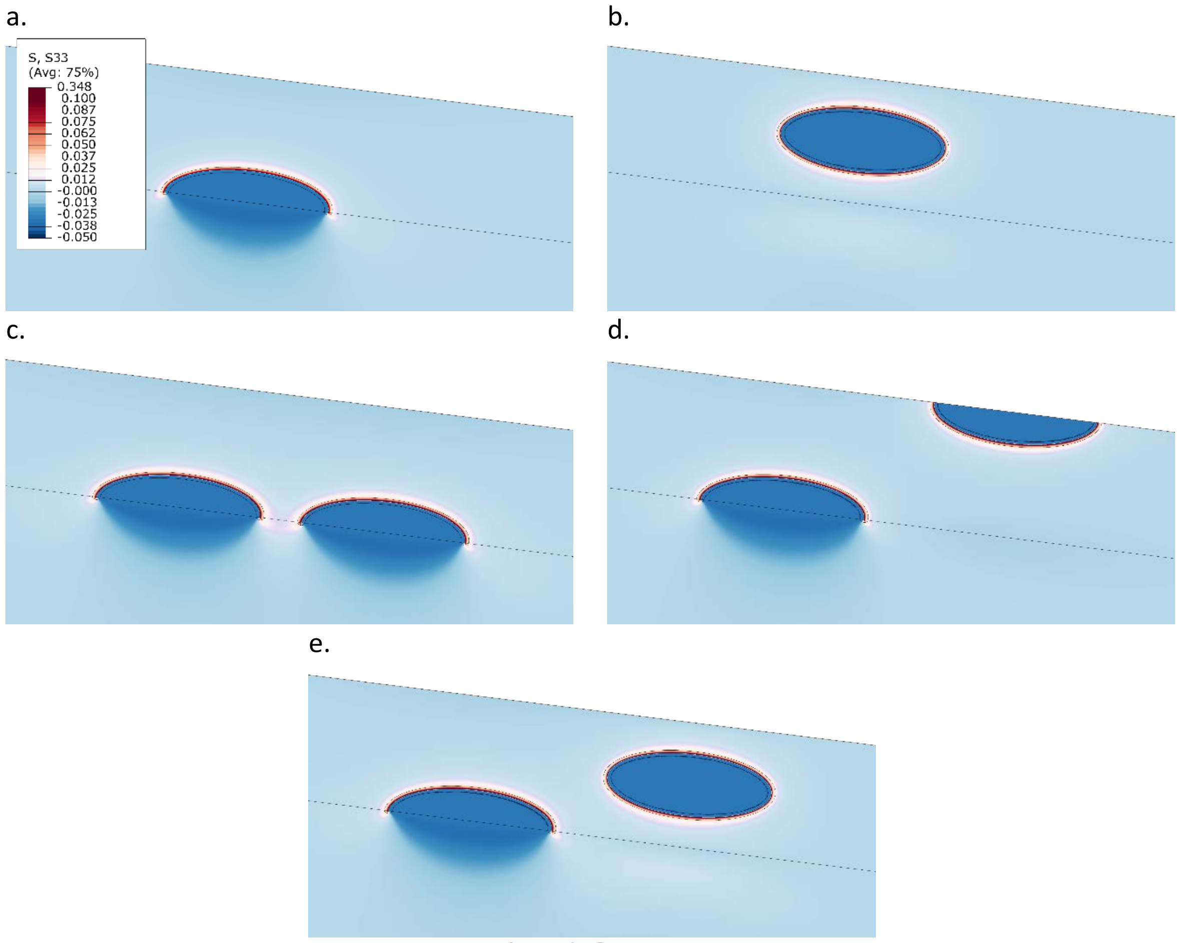


Figure 1: Flaw geometries which can be handled by `int_defects`*. a.) Single surface semi-elliptical crack (showing stress in the vertical direction as a proportion of yield), b.) single embedded elliptical crack, c.) two co-planar surface cracks (same face), d.) two co-planar surface cracks (opposite faces), e.) surface semi-elliptical crack and embedded elliptical crack. In all cases, each crack can be of arbitrary size and aspect ratio.*

3. Installation

3.1 Compatible operating systems and CPU architecture

The `int_defects` toolbox was developed on Windows 7 and CentOS Linux 6.8. It should work on any other recent Windows or Unix-based operating system which has the required software installed. Abaqus (required by `int_defects`) can run on Windows x86-32 and x86-64 architectures, and Linux x86-64.

3.2 Required software

1. An installation of **MATLAB** (The MathWorks, Inc.). `int_defects` was developed using MATLAB R2016a and has been tested with older MATLAB versions as far back as R2013a. The toolbox's behaviour on older versions of MATLAB than R2013a is untested.
2. An installation of the **Abaqus** FEA suite (Dassault Systèmes SE). `int_defects` uses the finite element analysis pre-processor Abaqus/CAE and the FE solver Abaqus/Standard. The toolbox has been tested on Abaqus versions between Abaqus v6.12 and Abaqus 2018.
3. `int_defects` makes use of the **MATLAB Parallel Computing Toolbox (PCT)** if it is available, but can function without it.

The PCT is used to parallelise the execution of Abaqus jobs, which reduces the time taken to run large series of relatively simple FE models. Further information is given in Section 4.3.

3.3 Installation procedure

`int_defects` is installed from a MATLAB toolbox file. Alternatively, the functions that make up the toolbox can be added to the MATLAB path directly by the user, which be more convenient if the user needs to edit the code.

3.3.1 Toolbox installation (recommended)

1. Unpack the installation file int_defects vX.X.X.zip. (where vX.X.X is the toolbox version).
2. Open MATLAB and navigate to the directory containing installation file contents.
3. Double-click on int_defects.mltbx and then click "Install".

To uninstall the toolbox, in the MATLAB "Home" tab go to the "Add-Ons" drop-down. Click "Manage Add-Ons" then "Uninstall" next to int_defects in the toolbox listing.

3.3.2 Alternative installation method: add functions to the MATLAB path

1. Unpack the installation file int_defects.zip.
2. Open MATLAB and navigate the directory containing installation file contents.
3. Right-click on the directory and select "Add to path>Selected folders and subfolders".

When using this alternative method you may have to either re-add the path each time MATLAB is started, or add the path name explicitly to the MATLAB's startup.m script. The former method will not work if you need to call `int_defects` in MATLAB from an external script (see Section A6).

4. Usage guide

This section gives details on how `int_defects` can be used. More detailed worked usage examples can be found in Appendix A.

Detailed usage notes can be found in the code comments of each of the functions included in `int_defects`*. Use the MATLAB command:*

```
doc *my_function_name*
```

or:

```
open *my_function_name*
```

to view them.

4.1 Defining a series of models

In `int_defects`, series of 3D cracked-body models are defined using a MATLAB data structure which always has the name "paramRangeStruct". `int_defects` interprets paramRangeStruct, and uses it to generate individual models which are then run. In a typical workflow, paramRangeStruct is set up by the user (using MATLAB) and then saved in a .mat file called eg. *my_paramRangeStruct*.mat. This filename is then passed as input to the function `int_defects_write_inp_parametric`, which then generates the set of model input files (see Section 4.2).

The parameters that make up paramRangeStruct define the geometry of crack pairs being investigated, their material properties, loading conditions, mesh sizes and model output requests. Section 7 lists the options that can be given in paramRangeStruct.

Some parameters in paramRangeStruct must be defined as a scalar value or a string, in which case all models in the model series will use the same parameter value. Other parameters may be defined using a vector or a cell array of strings, in which case models will all possible combinations of the provided values will be created. Details of which parameters may take multiple values and which must be the same in all models are given in Section 7.

4.2 Generating input files for the models

The function `int_defects_write_inp_parametric` creates a series of model input files by calling Abaqus/CAE. When run, the function it creates a new directory which is named "abaqus_param_files" and then populates it with sub-directories (one for each parameter combination) into which the individual model input files are written. The function will also write the logfile "log1.txt" into the same directory. This logfile gives information about the process of model generation, eg. whether there were any combinations of parameters for which model input files could not be created.

`int_defects` *uses the Abaqus/CAE python scripting interface. To generate a model, a python script listing the actions required to create the model input file is written by MATLAB and then executed by Abaqus/CAE.*

Warning: *Running the function* `int_defects_write_inp_parametric` *will terminate any open instances of Abaqus/CAE that are owned by the current user. Users should save any other models or results files open in Abaqus/CAE before proceeding.*

To create model input files `int_defects_write_inp_parametric` uses a 'master' python script that outlines the set of actions which must be performed by Abaqus/CAE to create a model of the type used by `int_defects`. This master python script is modified individually for each model and then executed using the Abaqus/CAE scripting interface.

`int_defects_write_inp_parametric` also checks on the outcome of each attempt to generate a model input file. If Abaqus/CAE is not able to create or mesh the model as-specified, `int_defects_write_inp_parametric` can relax the meshing constraints and attempt to re-mesh the model.

4.3 Solving the models

The function `int_defects_run_parametric_parallel` is used to execute a series of models using Abaqus/Standard. The series of model input files required for this will (typically) have been created beforehand using `int_defects_write_inp_parametric`. `int_defects_run_parametric_parallel` can execute several models in parallel if this is requested by the user, and if the system and number of available Abaqus licenses will permit. Running several models in parallel can help to minimise the total runtime necessary to execute for a large series of models. In addition, the number of CPUs assigned to each parallel Abaqus job can be defined.

Although Abaqus/Standard itself can use parallelisation when solving individual models, it does not parallelise the model pre-processing and this can be a significant fraction of the total runtime. Therefore, using MATLAB parallelisation to run several Abaqus jobs at the same time can give a significant performance improvement. This improvement is hardware-dependent. The number of parallel Abaqus jobs and the number of CPUs per job must be defined as input to `int_defects_run_parametric_parallel`. The product of these two should not exceed the number of CPUs available on the system.

The function `int_defects_run_parametric_parallel` automatically controls model execution and has several features for ensuring that the model series runs reliably. It can catch and retry failed jobs, perform contour-independence checks on contour integral output, delete unnecessary Abaqus output files, and save information on the outcome of each model execution attempt in a MATLAB-accessible file (workspace_dump.mat).

4.4 Post-processing (crack tip contour integrals)

Contour integral output can be requested using the MATLAB structure which defines the model set (paramRangeStruct), using the options described in Section 7.5. When contour integral output is requested, the user must specify the cracks(s) required, the contour integral type (K_I, J-integral or T-stress) and the number of contours. Contour integral results are given for each node on the crack tip line. The conventions used for automatic numbering of the crack tip nodes, and hence the order in which contour integral output for each node is given, are described in Section 5.3.4.

In `int_defects`, the function `int_defects_run_parametric_parallel` is used to solve all models in a set (see Section 4.3). Once this has been run and the models are complete, a directory labelled `abaqus_param_files` will have been created, containing model output (see Section 5.4). This should be renamed by the user to: `abaqus_param_files_*my_descriptive_name*`. Then, the script `int_defects_read_user` can be run to extract contour integral results from all models in the set. `int_defects_read_user` is a user-friendly "wrapper" script which calls the function `int_defects_read_parametric`. The results are saved automatically in the outputArray structure in the file `workspace_dump.mat`: one array entry for each model. They can then be plotted using the functions `plot_KI_over_crack` or `plot_J_over_crack`.

The Abaqus input files used by `int_defects` request contour integral output for all increments in each model, and contour integral results are extracted by `int_defects_read_user` for every available increment. Therefore, contour integral results should be available for the model's complete time history.

For sets of elastic-plastic models where the J-integral has been requested, it is possible to extract the contour integral result from the model increments where plastic breakthrough to the back face of the plate/pipe occurs. The special-purpose function `int_defects_extract_breakthrough_J` is provided for this purpose; an example of its use is given in Section A5.

For sets of linear elastic models performed to determine the level of elastic interaction between two flaws, the function `int_defects_calc_interaction_factor` may be used calculate interaction factors. This requires results from two or three sets of models: one set of models of interacting cracks and one or two sets of models of single cracks. Interaction factors can be plotted using the function `plot_IF_over_crack`.

4.5 Post-processing (limit load analysis)

For elastic → perfectly-plastic models performed as part of a limit load analysis, local limit loads and global limit loads can be extracted automatically from the results of a set of models by navigating into the directory containing model results (`abaqus_param_files`) and running the function `int_defects_plastic_breakthrough_parametric`. For limit load analysis, the following definitions are used:

- **Local limit load**: The load at which any path of yielded/yielding material forms between the crack and a specified plane (which is normally the opposite surface of the plate/pipe). Any plastic path is allowed, including paths which do not lie on the crack plane.
- **Global limit load**: The load at which global plastic instability occurs. This is determined from the last model increment performed; it is assumed that the solver will stop once the solution becomes unbounded due to plastic instability.

5. Detailed description

5.1 Scope and requirements

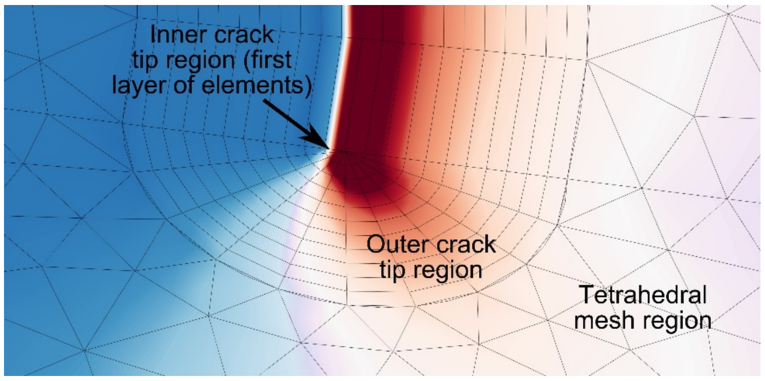
- **LEFM/EPFM scope**: int_defects was originally designed for linear elastic analysis. As a result, much of the output post-processing is designed for manipulating SIF results. Automatic sorting and comparison of SIF data is available, but currently there is no facility available to do automatic comparison of J-integrals.
- **Master .py files**: When generating the input deck for each finite element model, int_defects uses a 'master' .py file. This serves as a template for a Python script which is written (by int_defects) is then executed (by Abaqus/CAE) to generate the model input. Currently, two master .py files are used, one for single cracks and one for pairs of cracks. The current versions are:
    - Single cracks: IntCrackJob1_6.0.4_single_master.py
    - Pairs of cracks (any type): IntCrackJob1_6.0.4_master.py

5.2 FE modelling

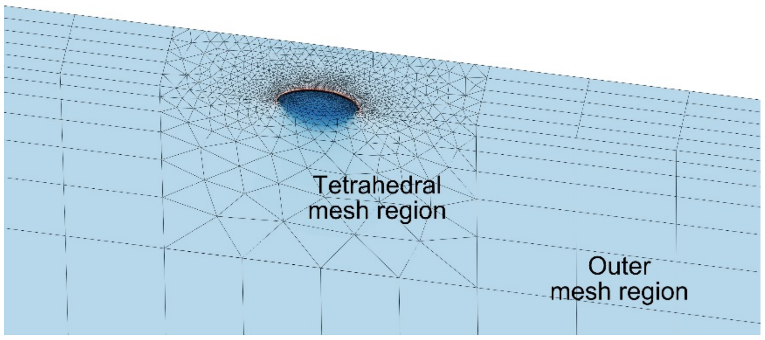
- **Mesh description**: The finite element meshes that are automatically generated using int_defects_write_inp_parametric consist of four main sections, shown in Figure 2. The innermost mesh region (the 'inner' crack tip region) is a ring of 6-noded linear wedge elements which directly surrounds each crack tip. This is generated by setting the element type of a semi-annular region around the crack tip to 'brick' in Abaqus CAE, and defining 'collapsed, single node' conditions at the crack tip line in the crack object. The second region is an 'outer' semi-annular region which contains 8-noded linear brick elements. The size of this outer region can be set manually in paramRangeStruct using rpA2 and rpB2 after varShellSizeFlag has been set as false. The number of elements in the radial direction can be set using noTipElemsRadial2A and noTipElemsRadial2B. This will limit the number of contours that can be requested in contour integral output. The next region is an unevenly-shaped region of quadratic tetrahedral elements which is used to match the complex-shaped crack tip zone regions to the mesh of the rest of the plate. The outermost region is a region of 8-noded linear brick elements which represents the remainder of the plate. Tie constraints are automatically generated to link the region of tetrahedral elements to the regions adjacent to it.
- **Materials:** int_defects can create and run FE models of cracks using any of the following material models:
    - **Linear elasticity**: Models of cracks in plates of linear elastic material (Abaqus keyword: *ELASTIC) are used for determining SIFs and T-stresses. Only *isotropic* linear elasticity is supported by int_defects. The Young's modulus E and Poisson's ratio ν must be given in paramRangeStruct.
    - **Linear elasticity + incremental plasticity:** A material true stress-strain curve for incremental plasticity can be defined using paramRangeStruct.materialParams.PlasticTable. The material model uses a von Mises yield locus and an isotropic hardening law (Abaqus keywords: *ELASTIC and *PLASTIC). This method is used for models of elastic-plastic materials, and for elastic-perfectly-plastic models used to determine limit loads.
    - **Deformation plasticity:** A hyperelastic deformation plasticity model is also available (Abaqus keyword: *DEFORMATION PLASTICITY). This is an approximation of real behaviour for materials which follow a Ramber-Osgood stress-strain relationship and are subjected to monotonic loading.

- **Contour integral output**: It is important to ensure that the number of contours requested is less than the number of nodes in the radial direction in the first two mesh zones. Also, it should be verified that the. `int_defects_contour_independence_check` will check SIF results for contour-dependence. However, the criterion used for determining contour independence was developed in an ad hoc manner, and cannot be guaranteed to work for all cases. Although `int_defects_contour_independence_check` does seem to run for models which give J output, the results will almost certainly be unreliable: the script was not designed for this type of output and I have not tested it.
- **Crack face contact:** The models constructed by `int_defects` do not consider the effect of contact between the opposing surfaces of a crack. This may affect the accuracy of contour integral results when: a.) the loading applied is highly non-uniform across the wall thickness, and/or b.) an inelastic material is used and the loading is strongly non-monotonic with respect to time.

a.



b.



c.

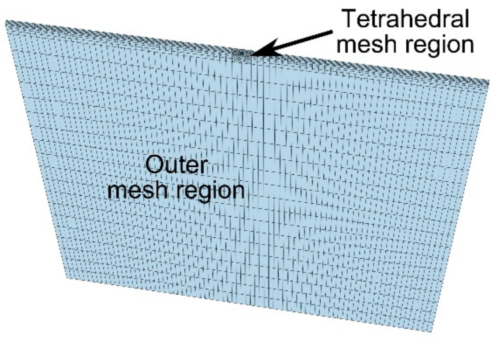


Figure 2: Meshing arrangement used for investigating semi-elliptical and elliptical defects in a plate. a.) Close-up of the intersection of the crack tip line with the plate's surface, b.) Tetrahedral and outer mesh regions, c.) whole model. Only half of the complete plate is modelled; symmetric boundary conditions have been defined on the crack plane.

5.3 Conventions

5.3.1 Units

Like Abaqus, `int_defects` requires that a self-consistent system of units is used throughout, but does not specify what system of units that must be. For example, it would be possible to provide input in SI base units (m, s, N, Pa, J) in which case SIF output in units of Pa √m would be expected. During development of `int_defects`, a Newtons-mm convention (mm, s, N, MPa, N-mm) was normally used as this is more convenient for most engineering applications. In this convention, SIF output in MPa √mm would be expected.

It is important to ensure that a consistent set of units is used when specifying all fields in paramRangeStruct.

5.3.2 Coordinate system

All models use a cartesian coordinate system. For models of plates:

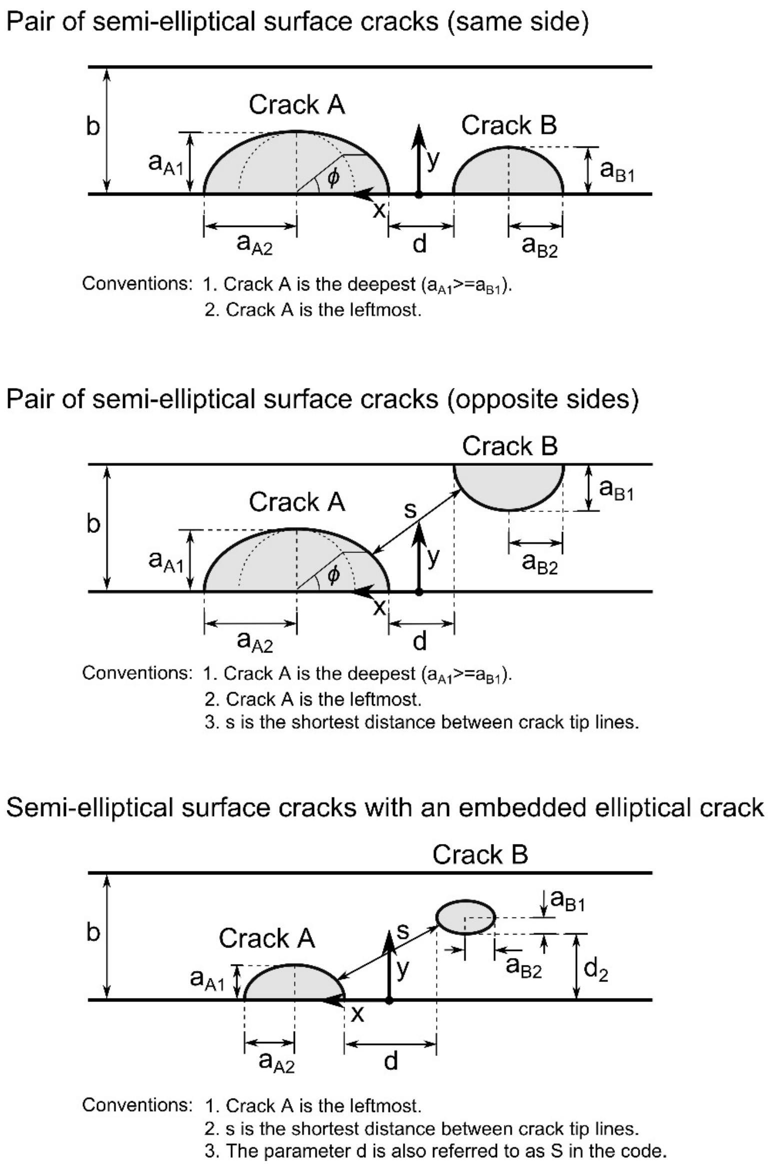
- The coordinate system is defined such that the x-y plane is the model's symmetry plane, and the model exists in +ive z.
- The long edge of the model cross-section in the symmetry plane runs along x and through the origin (making y the through-thickness coordinate).

For models of pipes:

- The coordinate system is defined such that the x-y plane is the model's symmetry plane, and the model exists in +ive z.
- The origin of the coordinate system lies on the axis of the pipe.

5.3.3 Crack geometry and positioning

- For a pair of two surface cracks, Crack A is always the deepest of the two and Crack B is shallower (regardless of the crack widths).
- When there is one surface crack and one embedded crack, Crack A is always at the surface and Crack B is embedded.
- When cracks are offset from each other in the x direction, Crack A is in the +ive x side of the origin and Crack B is on the -ive x side of the origin. If they are not offset, then they will both be aligned with their centres on the x=0 line.


Pair of semi-elliptical surface cracks (same side)
Crack A
Crack B
b
a_{A1}
a_{B1}
ϕ
y
x
a_{A2}
d
a_{B2}
Pair of semi-elliptical surface cracks (opposite sides)
Crack B
Crack A
b
s
a_{A1}
a_{B1}
ϕ
y
x
a_{B2}
a_{A2}
d
Semi-elliptical surface cracks with an embedded elliptical crack
Crack B
a_{B1}
b
Crack A
s
y
a_{A1}
a_{B2}
d_2
x
a_{A2}
d

Conventions: 1. Crack A is the deepest (a_{A1}>=a_{B1}).
2. Crack A is the leftmost.

Conventions: 1. Crack A is the deepest (a_{A1}>=a_{B1}).
2. Crack A is the leftmost.
3. s is the shortest distance between crack tip lines.

Conventions: 1. Crack A is the leftmost.
2. s is the shortest distance between crack tip lines.
3. The parameter d is also referred to as S in the code.

Figure 3: Definition of basic geometric parameters for pairs of cracks.

5.3.4 Ordering of contour integral output along the crack tip line

The numbering of contour integral output locations along the crack tip line is performed automatically by Abaqus/CAE after meshing. The output location order varies depending on the geometry of the crack or crack pair:

- **For single semi-elliptical surface cracks**: output locations proceed *anticlockwise* starting from the point on crack tip line *furthest in negative x* (which is a surface intersection).
- **For single embedded elliptical cracks**: output locations proceed *anticlockwise* starting from the point on the crack tip line *furthest in negative x*.
- **For pairs of semi-elliptical surface cracks emanating from the same side of the wall**: in both Crack A and Crack B, output locations proceed *anticlockwise* starting from the point on the crack tip line *furthest in negative x* (which is a surface intersection).
- **For pairs of semi-elliptical surface cracks emanating from opposite sides of the wall**: In Crack A, output locations proceed *anticlockwise* starting from the point on the crack tip line *furthest in negative x* (i.e. near-side surface intersection). In Crack B, output locations proceed *anticlockwise* starting from the point on the crack tip line *furthest in positive x* (surface intersection).

- **For a semi-elliptical surface crack and an elliptical embedded crack**: In Crack A, output locations proceed *anticlockwise* starting from the point on the crack tip line *furthest in negative x* (i.e. near-side surface intersection). In Crack B, output locations proceed *clockwise* from the point on the crack tip line *furthest in negative x*.

The information on node ordering given above is the expected behaviour at the time of writing. The node order is generated by Abaqus and so this advice by be invalidated by future changes to the Abaqus FEA suite.

5.4 Directory structure used for creating and executing jobs

To create model input files, you first need a master python scripting interface file and a MATLAB .mat file containing the structure paramRangeStruct. Master python files are provided with `int_defects`, while the .mat file must be created by the user. So initially, the working directory should look like this:

```
IntCrackJob1_6.0.4_single_master.py
my_paramRangeStruct.mat
```

When creating model input files, `int_defects_write_inp_parametric` creates a directory structure like this:

```
abaqus_param_files
  → 000001
     → abaqus.rpy
     → IntCrackJob1.inp
     → IntCrackJob1_generated.py
  → 000002
     → abaqus.rpy
     → IntCrackJob1.inp
     → IntCrackJob1_generated.py
  → … etc. (up to the total number of parameter combinations)
  → log1.txt
  → workspace_dump.mat
IntCrackJob1_6.0.4_single_master.py
my_paramRangeStruct.mat
```

When each model is subsequently run by `int_defects_run_parametric_parallel`, the results data file, status file, etc. (but not the output database, .odb) is saved in each subdirectory:

```
abaqus_param_files
  → 000001
     → abaqus.rpy
     → IntCrackJob1.com
     → IntCrackJob1.dat
     → IntCrackJob1.inp
     → IntCrackJob1.msg
     → IntCrackJob1.sim
     → IntCrackJob1.sta
```

```
        → IntCrackJob1_generated.py
    →  000002
        → abaqus.rpy
        → IntCrackJob1.com
        → IntCrackJob1.dat
        → IntCrackJob1.inp
        → IntCrackJob1.msg
        → IntCrackJob1.sim
        → IntCrackJob1.sta
        → IntCrackJob1_generated.py
    →  ... etc. (up to the total number of parameter combinations)
    →  log1.txt
    →  workspace_dump.mat
IntCrackJob1_6.0.4_single_master.py
my_paramRangeStruct.mat
```

Note: If `int_defects_run_parametric_parallel` is run with the input option abaqusCleanupFlag=false, then all Abaqus/Standard output files will be retained in each subdirectory, including the (potential large) .odb file.

When the elastic interaction factor is calculated by `int_defects_calc_interaction_factor` using a set of linear-elastic results for models of crack pairs, and a set (or sets) of corresponding models of single cracks, an additional logfile (log2.txt) is written into the abaqus_param_files directory for the main model set. log2.txt contains details of which cases it was possible to calculate elastic interaction factors for, given the available results.

5.4 paramLogStruct

The main functions in `int_defects`, such as `int_defects_write_inp_parametric` and `int_defects_run_parametric_parallel`, log important information about each model in a model set. The master log of model parameters and execution conditions is the structure “paramLogStruct” in the MATLAB file abaqus_param_files\workspace_dump.mat (see Section 5.4). This keeps a record of:

- Model file names:
    - paramLogStruct(*myModelNo*).filenames
- Model parameters (geometry in absolute distance):
    - paramLogStruct(*myModelNo*).modelParams
- Model parameters (geometry in normalised parameters, as in paramRangeStruct):
    - paramLogStruct(*myModelNo*).naturalParams
- Status of model input file generation:
    - paramLogStruct(*myModelNo*).inpStatus
    - paramLogStruct(*myModelNo*).modelParamsValidityComment
    - paramLogStruct(*myModelNo*).inpWriteExitMessage
    - paramLogStruct(*myModelNo*).inpWriteWarnings
- Status of model execution:
    - paramLogStruct(*myModelNo*).jobExitStatus
    - paramLogStruct(*myModelNo*).contDepWarningFlagLevel1
    - paramLogStruct(*myModelNo*).contDepWarningFlagLevel2

6. List of functions

In the following list, key high-level functions are given in ***bold italic*** type. All functions are fully commented and users are encouraged to check code comments for extra usage information.

6.1. Interpreting paramRangeStruct into model parameters

- `int_defects_calc_model_edge_params.m`
  - Determines model parameters from the parameter set structure paramRangeStruct. Specifically, int_defects_calc_model_edge_params determines model spatial geometry such as the locations of mesh transitions, node seeds etc.
- `int_defects_calc_model_params.m`
  - Determines model parameters from the parameter set structure paramRangeStruct. This function calls int_defects_calc_model_edge_params.

6.2. Elliptic Geometry

- `calc_ellipse_angle.m`
  - Approximates the ellipse parametric angle corresponding to a particular length of arc around an ellipse's circumference.
- `calc_ellipses_approach.m`
  - Approximates the distance of closest approach between two ellipses with given centres and radii.
- `calc_ellipses_approach_multiple_d.m`
  - Runs calc_ellipses_approach for pairs of differently-spaced ellipses with different centre locations.
- `int_defects_d_from_sOverb.m`
  - Approximates the distance between edges in the x-direction for a pair of opposed semi-ellipses which are offset in y. The ellipses have given radii and a given distance of closest approach.
- `test_ellipses_interior.m`
  - Determines whether or not a given point lies within a specified ellipse.

6.3. Processing and plotting results

- `int_defects_contour_independence_check.m`
  - Checks the contour-dependence of Mode I SIFs, when they are calculated as a contour integral from FEA results.
- ***`int_defects_calc_interaction_factor.m`***
  - Calculates elastic fracture interaction factors for interacting flaws from SIF results. It requires SIF results from two sets of models – one set of models of interacting flaws, and one set of models of single flaws in isolation.
- `int_defects_plot_J_vs_stress.m`
  - Determines the J-integral for a given position on the crack tip line for a cracked-body FE model, and plots it against the applied remote stress.
- `int_defects_read_J_at_inc.m`
  - Extracts J-integral values for a specific location and point in time from a set of crack driving force models.
- `int_defects_read_J_at_inc_parametric.m`
  - Loops through a set of elastic-plastic models, running `int_defects_read_J_at_inc` for each.
- ***`int_defects_read_parametric.m`***

- Reads contour integral data from a set of FE models of single or paired interacting cracks.

- ***`int_defects_read_user.m`***
  - User-friendly "wrapper" script for `int_defects_read_parametric` which automates file read/write operations when working in a folder containing multiple results directories.
- `int_defects_upgrade_odbs.m`
  - Loops through a set of numerically-named directories which each contain the results from a single Abaqus model, and upgrades any .odb files therein to be compatible with the latest installed version of Abaqus/CAE.
- `plot_IF_over_crack.m`
  - Generates a plot of elastic interaction factor vs. ellipse parametric angle from two sets of FEA results, one for single cracks and one for pairs of cracks.
- `plot_KI_over_crack.m`
  - Generates a plot of SIF vs ellipse parametric angle from a set of FEA results of elliptical or semi-elliptical crack-like defects in a structure.
- `plot_J_over_crack.m`
  - Generates a plot of J-integral vs ellipse parametric angle from a set of FEA results of elliptical or semi-elliptical crack-like defects in a structure.
- `int_defects_extract_breakthrough_J.m`
  - Extracts the J-integral at various points on the crack tip line from models of embedded elliptical and surface semi-elliptical cracks in a plate of elastic-plastic material. The J-integral results are taken at the increment at which plastic breakthrough from the buried crack to the surface occurs.
- `int_defects_plastic_breakthrough.m`
  - Determines the model increment at which "plastic breakthrough" occurs in a FE model. Plastic breakthrough is defined as the appearance of an unbroken path of elements, each with a plastic strain value greater than a specified tolerance, which connects a specified location (or any one of a set of locations) with a specified plane. This function requires that equivalent plastic strain data is available in the Abaqus data file. To ensure this, the following custom output request can be added in paramRangeStruct when setting up the models:
    paramRangeStruct.outputRequests.custom = '** OUTPUT REQUESTS
     *El Print, Elset = PlatePartitioned-1.Set-1, Position = Centroidal, Frequency=1
     PEEQ'.
- ***`int_defects_plastic_breakthrough_parametric.m`***
  - Loops through a set of models running `int_defects_plastic_breakthrough` for each.

6.4. Creating and running jobs

- `int_defects_flip_normal_in_inp.m`
  - In an Abaqus FE model input file, inverts the direction of the normal vector used for contour integral evaluation.
- ***`int_defects_run_parametric_parallel.m`***
  - Runs a series of FE models representing structures containing crack-like defects. This function is generally run after `int_defects_write_inp_parametric` has produced a set of model input files.

- `int_defects_write_inp.m`
  - Uses Abaqus/CAE to generate a FE input file using a 'master' python script and a set of parameters defining a model of a cracked structure.
- ***`int_defects_write_inp_parametric.m`***
  - Takes a set of parameters defining parametric models and runs `int_defects_write_input` to define a model input file for each.
- `int_defects_write_logfile_1.m`
  - Writes a logfile containing information on the generation of an Abaqus input file. It is called at the end of `int_defects_write_inp_parametric`, and writes the logfile into the working folder.
- `int_defects_write_logfile_2.m`
  - Writes a logfile containing information about the status of the models in a parametric series, and whether the results from these models were successfully used to calculate the interaction factors for pairs of cracks.
- `int_defects_plate2cyl_auto.m`
  - Takes an Abaqus input file for a model of a plate and applies a geometric transformation to the node locations, creating a model of a half-pipe.
- `int_defects_py_line_basic.m`
  - Constructs a single line for a python file which is used for generating a FE model using Abaqus/CAE. nB. `int_defects_py_line_basic` is used for making straightforward edits to the line of the master python file. `int_defects_py_line_edges` is used for making more complex edits.
- `int_defects_py_line_builder.m`
  - Constructs a string containing coordinates, deformation plasticity parameters or plastic flow data in a certain format used by the Abaqus scripting interface. This function is called by `int_defects_py_line_edges`.
- `int_defects_py_line_edges.m`
  - Constructs a single line for a python file which is used for generating a FE model using Abaqus/CAE. nB. `int_defects_py_line_basic` is used for making straightforward edits to the line of the master python file. `int_defects_py_line_edges` is used for making more complex edits.
- `int_defects_write_py.m`
  - Writes a python script for generating a FE model using Abaqus/CAE. It uses an existing 'master' (i.e. template) script, and parameters defined in modelParamStruct. It writes the new file into the current directory. Certain strings eg. #aA1# in the master file indicate where the parameters should be written to.
- `read_py.m`
  - Reads a python file line-by-line.

7. paramRangeStruct options

paramRangeStruct defines the set of models to be run. It includes geometric parameters (e.g. crack sizes and aspect ratios), modelling parameters (e.g. section sizes, node densities), material parameters (e.g. elastic constants), loading parameters and output requests. Parameters of these five different types are given in sub-structures of paramRangeStruct. The sub-structures are named:

- geometryParams
- loadingParams
- materialParams
- modelParams
- outputRequests

So for example, the parameter which defines the Poisson's ratio of the model material is "paramRangeStruct.materialParams.nu".

Parameters named in ***bold italic*** type in the sections below may be given as a vector (for numeric) or a cell array of strings (for strings). This will cause `int_defects` to produce a parametric set of models, with each model taking one of the values given in the vector/cell array. When multiple parameters are given as vectors or cell arrays, `int_defects_write_inp_parametric` will create models for all possible combinations of parameter values.

7.1 geometryParams (paramRangeStruct.geometryParams.*these sub-fields*)

- geometryType – String defining the type of geometry that will be analysed. Must be either 'plate' or 'pipe'. The 'pipe' option is used for models of studying flaws in the axial-radial plane of a pipe wall.
- pipeCrackA – String defining the reference surface for pipe models. If Crack A is on the internal surface of the pipe (or is an embedded crack with the depth measured from the ID) then pipeCrackA= 'internal'. If it is on the outside then pipeCrackA='external'. In models with two cracks, Crack B's position is then given relative to Crack A (see Figure 3). This parameter is not used when geometryType='plate'.
- singleCrackFlag – Logical defining whether the model is of a single crack (false for a pair of cracks).
- oppositeFlag - Logical defining whether Crack B emanates from the opposite surface as Crack A, false for the same side. nB. If Crack B is a sub-surface elliptical crack, it is considered as opposite.
- subsurfaceAFlag & subsurfaceBFlag – Logicals defining whether Crack A and B are subsurface cracks. False for surface semi-elliptical cracks.
- ***aA1Overb*** – Depth of Crack A expressed as a fraction of the wall thickness b.
- ***aA1OveraA2*** – Aspect ratio of Crack A (depth over half-width).
- ***aB1OveraB2*** - Aspect ratio of Crack B (depth over half-width). If this is given as NaN in a two-crack model, the same aspect ratio will be used for Crack B as for Crack A.
- Parameter defining the depth of Crack B. Note for an elliptical embedded crack, the total depth in the through-thickness dimension is 2*aB1. In models which contain two cracks, exactly one of the following variables must be defined:
    - ***aB1OveraA1*** - Depth of Crack B expressed as a fraction of the depth of Crack A.
    - ***aB1Overb*** - Depth of Crack B expressed as a fraction of wall thickness b.
- Parameter defining the separation distance in the along-wall (x) direction. Can be used alone (for pairs of surface cracks), or with an additional parameter defining the depth of Crack B (when there is an embedded crack). Exactly one of the following variables must be defined:

    - ***dOveraA1*** – Distance in x between crack edges (d), normalised to the depth of Crack A (aA1).
    - ***dOverb*** – Distance in x between crack edges (d), normalised to plate thickness (b).
    - ***dOverMax_aA1_2aB1*** – Distance in x between crack edges (d), defined as a ratio of max([aA1,2*aB1]).
    - ***normalisedOffset*** – Distance in x between crack centres, normalised by the mean crack width. normalisedOffset = 1-(d/(aA2+aB2)).
    - ***overlapRatio*** – Ratio of crack width overlapping in x. At overlapRatio = 0, near-side crack tips are at the same location in x. At overlapRatio = 1, the cracks are aligned.
    - ***twoaA2OverdCents*** – Special-purpose parameter, used for comparison with Sethuraman et al. 2003.
    - ***sOverb*** – Parameter defining the distance of closest approach between the crack tip lines. This is used in lieu of parameters defining the separation distances in x and y. sOverb is the closest approach between the crack tip lines (s), normalised to plate thickness (b). Note that for semi-elliptical cracks emanating from the same surface, this is the same as dOverb.
- Parameter defining a crack depth or separation distance in the through-thickness (y) direction. If there is a single crack (Crack A) and it is an elliptical embedded crack, its depth (distance of closest approach to xz plane) can be defined using d2Overa, d2Overb or normalisedOffsetY. Otherwise, when there are two cracks and Crack B is a sub-surface crack, exactly one out of any of the following variables must be defined:
    - ***d2Overa*** – Distance from the xz plane to the closest point on the crack tip line of the subsurface defect (d2), normalised by the depth of Crack A (aA1).
    - ***d2Overb*** - Distance from the xz plane to the closest point on the crack tip line of the subsurface defect (d2), normalised by the wall thickness (b).
    - ***normalisedOffsetY*** – If there are two cracks, this is the distance in x between crack centres, normalised by the mean crack depth: i.e. normalisedOffsetY = 1+(S/(aA1+aB1)). If there is only one crack, this is the distance in y of the crack centre from the plate mid-thickness, normalised by the plate thickness: i.e. yA = d2 + aA1 = b*(0.5-normalisedOffsetY).
    - ***SOvera*** - Distance in y between cracks (S), normalised to Crack A depth (aA1).
    - ***SOverb*** – Distance in y between cracks (S), normalised to wall thickness (b).
    - ***SOverc*** – Distance in y between cracks (S), normalised to Crack A width (aA2).
    - ***SOverSqrtac*** – Distance in y between cracks (S), normalised to Crack A dimensions sqrt(aA1*aA2).
    - ***sOverb*** – Parameter defining the distance of closest approach between the crack tip lines. This is used in lieu of parameters defining the separation distances in x and y. sOverb is the closest approach between the crack tip lines (s), normalised to plate thickness (b).
- plDepth – Depth of the plate. This must be defined for plate models.
- ***b*** – Thickness of the wall. This must be defined for plate models. For pipe models, either the b or riOverro parameter is required.
- ri – Scalar defining the internal radius of the pipe. Must be defined when analysing models with pipe geometry.
- ***riOverro*** – Pipe inner/outer radius ratio. For pipe models, either the b or riOverro parameter is required.
- Parameter defining the half-width of the plate (or half-length of pipe).

  - plHalfWidth – Half-width of the plate (or half-length of pipe). Default is 50*b.
  - plHalfWidthOveraA2 - Half-width of the plate (or half-length of pipe) normalised by the half-width of Crack A (aA2).
  - plHalfWidthOveraA2plusb - Half-width of the plate (or half-length of pipe) normalised by the half-width of Crack A plus the plate thickness (aA2 + b).

7.2 loadingParams (paramRangeStruct.loadingParams.*these sub-fields*)

- loadingType – String defining the type of loading to be applied to the model. Can be 'load', 'moment', 'combined', 'crackFacePressure', 'pipePressure' or 'pipePressureInclCrack'. The effects of each option are as follows:
  - 'load': Load is applied to the far edge of the plate through a kinematic coupling.
  - 'moment': Moment is applied to the far edge of the plate through a kinematic coupling.
  - 'combined': A load and a moment are applied to the far edge of the plate through a kinematic coupling.
  - 'crackFacePressure': Pressure loading is applied directly to all crack faces. In LEFM, this is used to simulate any arbitrary through-wall stress distribution. The pressure can be an arbitrary function of (see CFPArray).
  - 'pipePressure': Used when geometryType = 'pipe', this applies a uniform pressure to the pipe's internal surface. The crack faces are not loaded.
  - 'pipePressureInclCrack': Used when geometryType = 'pipe', this applies a uniform pressure to the pipe's internal surface, and to the faces of any cracks which are open to the internal surface.
- ***CFPArray*** – n-by-1 cell array of strings defining the loading distributions when loadingType='crackFacePressure'. Typical string would be '1-(0*(Y/1000))' (tension) and '1-(2*(Y/1000))' (through-wall bending). Note that there is no automatic normalisation of the pressure w.r.t. the plate thickness.
- **loadMagnitude** – numeric array defining the magnitude of the applied load when loadingType='load' (or the applied moment when loadingType='moment'). loadMagnitude should be a column vector of loads/moments if loadingType='load' or loadingType='moment'. The length n of the column vector is equal to the number of different loading states to be considered. When loadingType='combined', loadMagnitude is an array of size n-by-2 (the first column specifies load, the second specifies moment). If the parameter loadMagnitudeUnitWidthFlag is included and set to true, loadMagnitude specifies the load/moment *per unit of plate width*. Otherwise loadMagnitude specifies the total load/moment *applied to the whole plate*. Notes:
  - The default width of the plate being analysed is 100x the thickness b, although it can be set explicitly (using plHalfWidth). For example, when using a N-mm-MPa convention: to analyse a plate of 50 mm thickness with a uniform tension of 500 MPa applied will require a loadMagnitude of 50mm x (100x50mm) x 500MPa = 1.25e8 Newtons.
  - In bending, a *negative* bending moment about the x-axis will cause a tensile stress the y=0 side of the plate thickness. This is generally what is required, since this will cause crack-opening of cracks emanating from the y=0 face.
  - For calculating the required moment for linear elastic cases (i.e. when calculating SIFs) the equation for ligament stress may be useful:

$$\sigma = \frac{My}{I}$$

where σ is the ligament stress, M is the moment, y is the distance from the neutral axis, and I is the second moment of area. The second moment of area of a beam of rectangular cross-section is:

$$I = \frac{bd^3}{12}$$

where b is the beam width and d is its depth.

- For calculating the required moment for limit load cases, the equation for the fully-plastic bending moment for a beam of rectangular cross-section may be useful:

$$M = \left(\frac{bd^2}{4}\right)\sigma_y$$

- When using loadingType='combined' none of the values in the array loadMagnitude should be zero. To analyse pure tension as part of a set of 'combined' loading states, use an arbitrarily small bending moment with it eg. […; 1.25e8, 1 ; …]. The same goes for analysing pure bending as part of a set of 'combined' loads: use an arbitrarily small tension load as well.

- loadMagnitudeUnitWidthFlag – Logical which may be used in conjunction with loadMagnitude. When set to true, this indicates that the values of load and/or moment specified in loadMagnitude are *per unit of plate width*. When set to false (which is the default if loadMagnitudeUnitWidthFlag is omitted), the values of load and/or moment specified in loadMagnitude are *for the whole plate*.
- pressureMagnitude – Scalar which defines the pipe internal pressure when loadingType='pipePressure' or loadingType='pipePressureInclCrack'.
- biaxialFlag - Logical which defines whether the loading on the crack is biaxial. For pressurised pipes, biaxialFlag=false is used for simulating open-ended conditions and biaxialFlag=true is used for simulating closed-ended conditions.
- biaxialStress – Scalar which defines the magnitude of the biaxial stress. To simulate the axial stress which results from pressure in closed-ended pipes, set biaxialStress=NaN and `int_defects` will calculate the appropriate axial stress based on the pressure and the pipe geometry.

Loading conditions and model incrementation should be considered carefully. For example, in a limit load analysis you should first estimate the load/moment/pressure which will cause the structure to reach global collapse, then ensure that the magnitude of loads applied to the model is greater than this. When using pipe geometry under pressure, consider whether the pipe is open ended (biaxialFlag=false) or closed-ended (biaxialFlag=true, biaxialStress=NaN).

7.3 materialParams (paramRangeStruct.materialParams.*these sub-fields*)

- type – Defines the type of material model to be used. Can be 'linear elasticity', 'deformation plasticity' or 'incremental plasticity'. These correspond directly to the standard material model types available in Abaqus.
- Elastic parameters (required for all material models).
  - E – Young's modulus
  - ***nu*** – Poisson's ratio. Default is 0.3.
- Parameters required by the 'deformation plasticity' model type.

    - s0 – yield strength.
    - n – hardening modulus (in the order of 10 for typical stainless steels)
    - alpha – yield offset. Typically 0.002. Note that the yield offset parameter definition used by Abaqus differs from the conventional definition. See Abaqus documentation on *DEFORMATION PLASTICITY for details.
- Parameters required by the 'incremental plasticity' model type.
    - plasticTable – n-by-2 table of true stress vs logarithmic plastic strain data.

7.4 modelParams (paramRangeStruct.modelParams.*these sub-fields*)

- nlgeomFlag – Logical indicating whether or not the model should be considered geometrically non-linear, or whether small-displacement formulation should be used. It is recommended that geometrically non-linear analysis should be used for determination of limit loads using elastic → perfectly-plastic models.
- noTipElemsA & noTipElemsB – Number of elements along one semi-ellipse of the Crack A and Crack B tip lines. 80 is a sensible value of these parameters for most applications. Meshing problems may occur if too few elements are used to define the crack tip line.
- noTipElemsRadial2A & noTipElemsRadial2B – Number of elements in the radial direction in the 'outer' crack tip zone. Default is 5. Note that the number of elements radially in the crack tip zones will limit the number of contours that can be used for K and J output.
- varShellSizeFlag – Logical defining whether the size of the crack tip element zones is defined dynamically. In almost all cases this should be set to true, allowing int_defects to define the crack tip element zone sizes.
- ***rpA2*** & rpB2 – These parameters are used to statically define the size of the 'outer' annular zone of elements which surround the crack tip. They are only used when varShellSizeFlag=false. In most models, these parameters are excluded from paramRangeStruct and will be defaulted to NaN, in which case int_defects automatically sets the annular region size.
- stepStr – A string which can be used to set step properties explicitly. It takes the form of a piece of Abaqus/CAE scripting interface code which will be used to generate the step options, eg. 'mdb.models['Model-1'].steps['ApplyLoad'].setValues(initialInc=0.05, maxInc=0.05, maxNumInc=1000, minInc=0.001, timePeriod=1)'. For elastic-plastic and limit load analyses, the user should ensure that they use appropriate increment size settings, since these may affect the precision of the results.
- modifyNaturalParamStructStr – A string which can be used to make initial modifications to naturalParamStruct at the beginning of `int_defects_calc_model_params`. This can be use to (for example) to create a set of models which represent assessment procedure re-characterisation of the defects run in an earlier set of models. Currently this must either be 'DNVGL-RP-F108', or must be and empty string or excluded altogether (in which case no initial modifications are made).

7.5 outputRequests (paramRangeStruct.outputRequests.*these sub-fields*)

- contourFreqA & contourFreqB – Frequency of contour integral output. Can be: an integer (for every n increments, eg. 1 for every increment) or 'LAST_INCREMENT'. (nB. Instead of frequency=n or frequency=LAST_INCREMENT, Abaqus would also accept numIntervals=*integer* or timeInterval=*positive scalar*).
- contourNameA & contourNameB – Strings defining names for the output requests.
- contourTypeA & contourTypeB – Type of contour integral output. Can be: 'none', 'K_FACTORS', 'J_INTEGRAL' or 'T_STRESS'. (nB. Abaqus would allow 'C_INTEGRAL' also).

- custom – A string defining a custom output request, which will be added to the *STEP block of the Abaqus .inp file. This is most useful for requesting field variable output. For example,
  ```
  '** OUTPUT REQUESTS
   *El Print, Elset = PlatePartitioned-1.Set-1, Position = Centroidal, Frequency=1
   PEEQ'
  ```
  Would be used to request equivalent plastic strain output for the complete model.
- noContoursA & noContoursB – Number of contours. Note that this is limited by the number of elements in the radial direction of the crack tip mesh region(s), which is defined by noTipElemsRadial2A & noTipElemsRadial2B. So for example, noContoursA should be less than or equal to noTipElemsRadial2A, or the Abaqus model will fail due to the invalid contour output request.

8. Validation

Validation of `int_defects` output has been performed via comparison with previous results from other authors. `int_defects` allows the user to define many aspects of the analysis. Many of these, such as the FE mesh density and the time incrementation in an elastic-plastic model, can affect the accuracy of the result. The validation examples listed in this section have only compared `int_defects` results acquired using "sensible" input options.

In all of the cases listed below, the agreement between results from `int_defects` and existing solutions was considered good. Detailed quantitative results are not given here; the purpose of this listing is to allow users to replicate the validation if necessary.

It is strongly emphasised that users should perform their own validation tests specific to the analysis type and geometry that they are studying. A range of analyses are possible using `int_defects`, and any general validation effort cannot guarantee the accuracy of results in all possible applications.

Please help to build these lists of validation cases by reporting results to: harry.coules@bristol.ac.uk

8.1 Linear-elastic analysis

Comparison between linear elastic analysis results from `int_defects` and results from existing literature have been performed. Comparison cases include:

- Single semi-elliptical surface cracks in an elastic plate under tension and bending. FEA solution of Newman & Raju [6]. Comparison of SIFs across crack front.
- Twin semi-elliptical surface cracks in an elastic plate under tension. FEA solutions of Yoshimura et al. [7] and Sethuraman et al. [8]. Comparison of SIFs across crack front. This comparison is discussed in: "*Stress intensity interaction between dissimilar semi-elliptical surface cracks*", Coules 2016 [9].

8.2 Inelastic analysis

For elastic-plastic and limit-load analysis of 3D cracks, the existing literature is more limited than for linear-elastic solutions. Comparison cases for which `int_defects` has been checked include:

- Single semi-elliptical surface crack in an elastic-plastic plate under tension. The material stress-strain curve is described by a "Linear Plus Power Law" (LPPL) relationship. FEA solutions of Allen & Wells [10]. Comparison of SERR across crack front.
- Single semi-elliptical surface crack in an elastic-plastic plate under tension. Material stress-strain curve (aluminium alloy 2219-T8) defined directly from a tensile test. FEA solutions from the ASTM modelling round-robin coordinated by Wells & Allen [11], which were validated by comparison to experimental fracture test results. Comparison of SERR across crack front.
- Single semi-elliptical surface flaws in an elastic-perfectly-plastic plate under tension. Classic limit load solution by Willoughby & Davey – an experimentally-validated analytical (plastic hinge) result [12]. Comparison of Local Limit Load. This comparison is presented in: "Analysis of defect interaction in inelastic materials", Coules & Bezensek 2019 [13].
- Single semi-elliptical internal axial-radial surface flaw in a thick-walled pipe under combined centripetal acceleration and thermal shock. Experimentally-validated round-robin FEA results from the NESC-1 study [14]. Comparison of SERR at deepest and near-surface points. This comparison used models based on the `int_defects` package, but which extended it (eg.

to thermal shock). This comparison is presented in: “Parametric design of scaled-down pressurised thermal shock test specimens using inelastic analysis”, Coules et al. 2017 [15].

9. References

9.1 Main reference list

[1] EDF, *R6: Assessment of the Integrity of Structures Containing Defects, Revision 4, Amendment 11*. EDF Energy, Gloucester, 2015.

[2] BSi, *BS 7910:2013+A1 (incorporating corrigenda 2) - Guide to methods for assessing the acceptability of flaws in metallic structures*. BSi, 2013.

[3] H. F. Bueckner, "Field singularities and related integral representations," in *Methods of Analysis and Solutions of Crack Problems*, vol. 1, G. C. Sih, Ed. Noordhoff International, 1973, pp. 239–314.

[4] H. F. Bueckner, "Novel principle for the computation of stress intensity factors," *ZAMM Zeitschrift fur Angewandte Mathematik und Mechanik*, vol. 50, no. 9, pp. 529–546, 1970.

[5] J. R. Rice, "Some remarks on elastic crack-tip stress fields," *International Journal of Solids and Structures*, vol. 8, no. 6, pp. 751–758, 1972.

[6] J. C. Newman and I. S. Raju, "Analyses of surface cracks in finite plates under tension or bending loads," NASA, 1578, 1979.

[7] S. Yoshimura, J.-S. Lee, and G. Yagawa, "Automated system for analyzing stress intensity factors of three-dimensional cracks: Its application to analyses of two dissimilar semi-elliptical surface cracks in plate," *Journal of Pressure Vessel Technology, Transactions of the ASME*, vol. 119, no. 1, pp. 18–26, 1997.

[8] R. Sethuraman, G. S. S. Reddy, and I. T. Ilango, "Finite element based evaluation of stress intensity factors for interactive semi-elliptic surface cracks," *International Journal of Pressure Vessels and Piping*, vol. 80, no. 12, pp. 843–859, 2003.

[9] H. E. Coules, "Stress intensity interaction between dissimilar semi-elliptical surface cracks," *International Journal of Pressure Vessels and Piping*, vol. 146, pp. 55–64, 2016.

[10] P. A. Allen and D. N. Wells, "Interpolation methodology for elastic-plastic J-integral solutions for surface cracked plates in tension," *Engineering Fracture Mechanics*, vol. 119, pp. 173–201, 2014.

[11] D. N. Wells and P. A. Allen, "Analytical Round Robin for Elastic-Plastic Analysis of Surface Cracked Plates: Phase I Results," NASA, NASA/TM-2012-217456, 2012.

[12] A. A. Willoughby and T. G. Davey, "Plastic collapse at part wall flaws in plates," in *Fracture mechanics: perspectives and directions. Proceeding of the 20th National Symposium. ASTM STP1020*, 1989, pp. 390–409.

[13] H. E. Coules and B. Bezensek, "Analysis of defect interaction in inelastic materials," in *Proceedings of the ASME 2019 Pressure Vessels & Piping Conference*, 2019, no. PVP2019–93219.

[14] R. Bass, J. Wintle, R. C. Hurst, and N. Taylor, "NESC-I Project Overview," European Commission, 2001.

[15] H. E. Coules, P. J. Orrock, and C. E. Truman, "Parametric design of scaled-down pressurised thermal shock test specimens using inelastic analysis," *Engineering Fracture Mechanics*, vol. 176, pp. 308–325, 2017.

9.2 List of publications that describe studies using `int_defects`

- B. Bezensek, H. E. Coules, J. Sharples and Y. Tkach, "Proposed updates to the buried-to-surface flaw recharacterization rules in the Annex E of BS 7910", *Proceedings of the ASME 2019 International Conference on Ocean, Offshore and Arctic Engineering*, OMAE2019-96327, 2019.
- H. E. Coules and B. Bezensek, "Analysis of defect interaction in inelastic materials", *Proceedings of the ASME 2019 Pressure Vessels & Piping Conference*, PVP2019-93219, 2019.
- H. E. Coules, "On predicting the interaction of crack-like defects in ductile fracture", *International Journal of Pressure Vessels and Piping*, vol. 162, pp. 98-101, 2018.

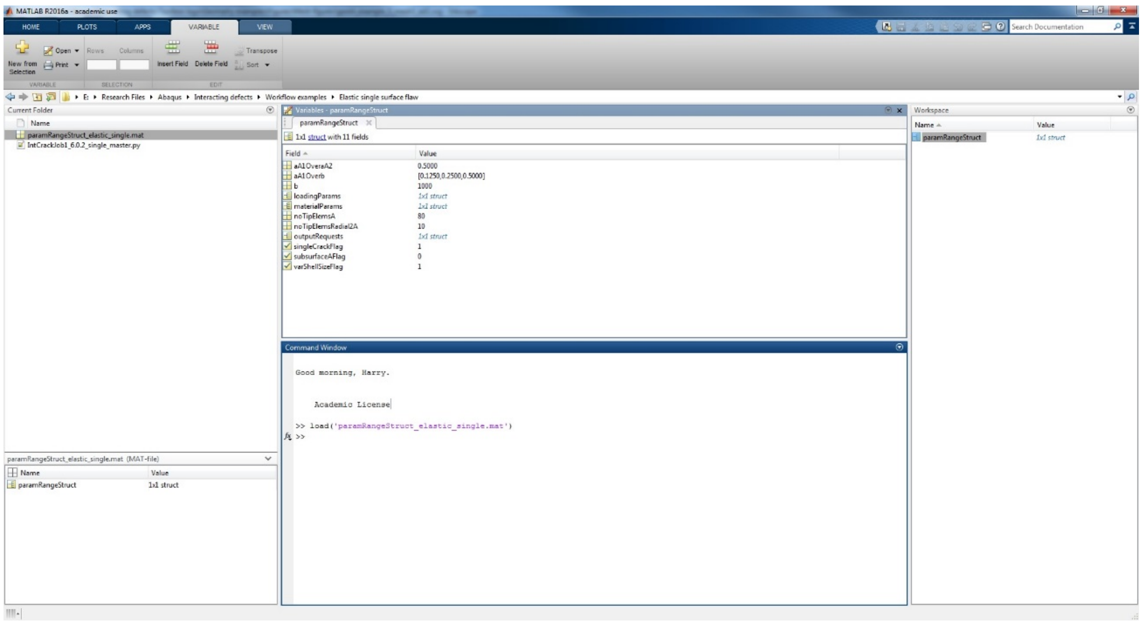
- H. E. Coules, “Interaction of surface cracks subjected to non-uniform distributions of stress”, *International Journal of Pressure Vessels and Piping*, vol. 157, pp. 20-29, 2018.
- H. E. Coules, “Flaw interaction under bending, residual stress and thermal shock loading”, *Procedia Structural Integrity*, vol. 13, pp. 361-366, 2018.
- H. E. Coules, P. J. Orrock and C. E. Truman, “Parametric design of scaled-down pressurised thermal shock test specimens using inelastic analysis”, *Engineering Fracture Mechanics*, vol. 176, pp. 308-325, 2017.

Appendix A: Workflow examples

A.1 Simple elastic analysis of a single surface flaw (with varying flaw size)

In this example, we create three models of a single semi-elliptical surface flaw in an elastic plate under remote tension. The flaw has an aspect ratio of a/c = 0.5 in all cases, while the normalised flaw depth is a/b = 0.125, 0.25, 0.5.

1. Copy the master python file "IntCrackJob1_6.0.4_single_master.py" into the working directory.
2. Using MATLAB, construct the data structure paramRangeStruct defining the model set. Save it in a .mat file.

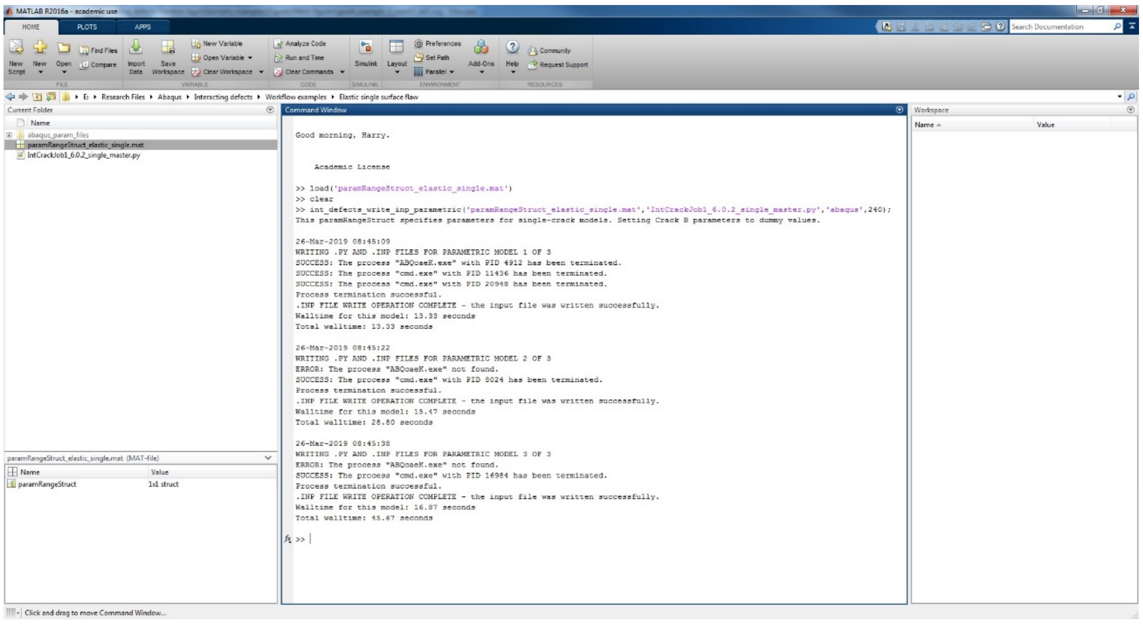


1x1 struct with 11 fields

Field	Value
aA1OveraA2	0.5000
aA1Overb	[0.1250,0.2500,0.5000]
b	1000
loadingParams	1x1 struct
materialParams	1x1 struct
noTipElemsA	80
noTipElemsRadial2A	10
outputRequests	1x1 struct
singleCrackFlag	1
subsurfaceAFlag	0
varShellSizeFlag	1

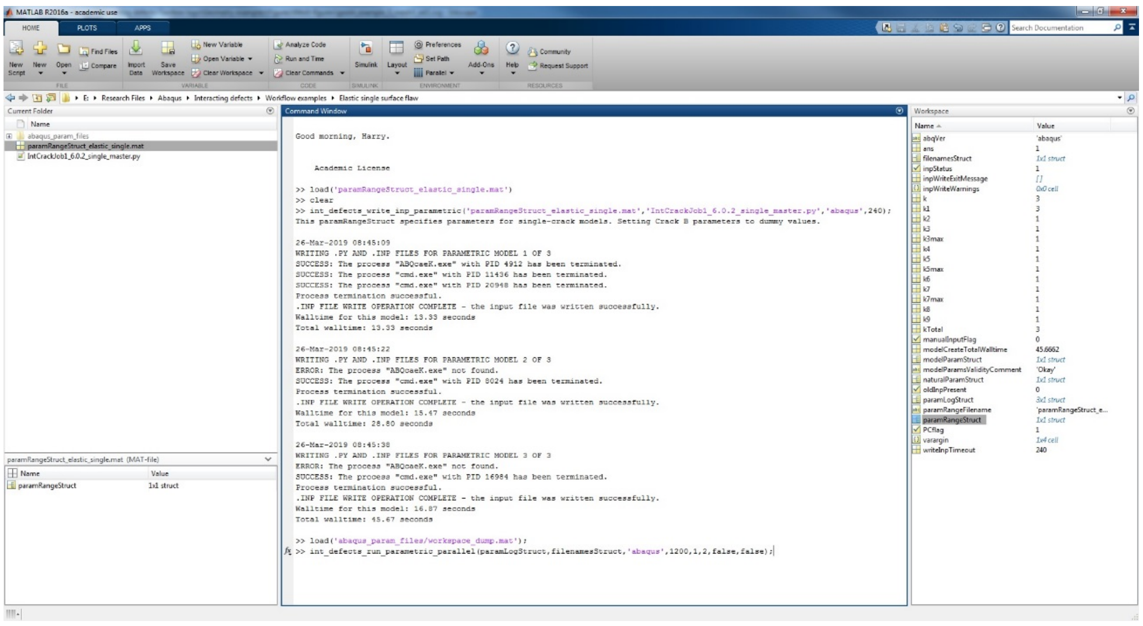
```
Good morning, Harry.


   Academic License

>> load('paramRangeStruct_elastic_single.mat')
>>
```

3. Run `int_defects_write_inp_parametric`.



```
Good morning, Harry.


   Academic License

>> load('paramRangeStruct_elastic_single.mat')
>> clear
>> int_defects_write_inp_parametric('paramRangeStruct_elastic_single.mat','IntCrackJob1_6.0.2_single_master.py','abaqus',240);
This paramRangeStruct specifies parameters for single-crack models. Setting Crack B parameters to dummy values.

26-Mar-2019 08:45:09
WRITING .PY AND .INP FILES FOR PARAMETRIC MODEL 1 OF 3
SUCCESS: The process "ABQcaeK.exe" with PID 4912 has been terminated.
SUCCESS: The process "cmd.exe" with PID 11436 has been terminated.
SUCCESS: The process "cmd.exe" with PID 20948 has been terminated.
Process termination successful.
.INP FILE WRITE OPERATION COMPLETE - the input file was written successfully.
Walltime for this model: 13.33 seconds
Total walltime: 13.33 seconds

26-Mar-2019 08:45:22
WRITING .PY AND .INP FILES FOR PARAMETRIC MODEL 2 OF 3
ERROR: The process "ABQcaeK.exe" not found.
SUCCESS: The process "cmd.exe" with PID 8024 has been terminated.
Process termination successful.
.INP FILE WRITE OPERATION COMPLETE - the input file was written successfully.
Walltime for this model: 15.47 seconds
Total walltime: 28.80 seconds

26-Mar-2019 08:45:38
WRITING .PY AND .INP FILES FOR PARAMETRIC MODEL 3 OF 3
ERROR: The process "ABQcaeK.exe" not found.
SUCCESS: The process "cmd.exe" with PID 16984 has been terminated.
Process termination successful.
.INP FILE WRITE OPERATION COMPLETE - the input file was written successfully.
Walltime for this model: 16.87 seconds
Total walltime: 45.67 seconds

>>
```

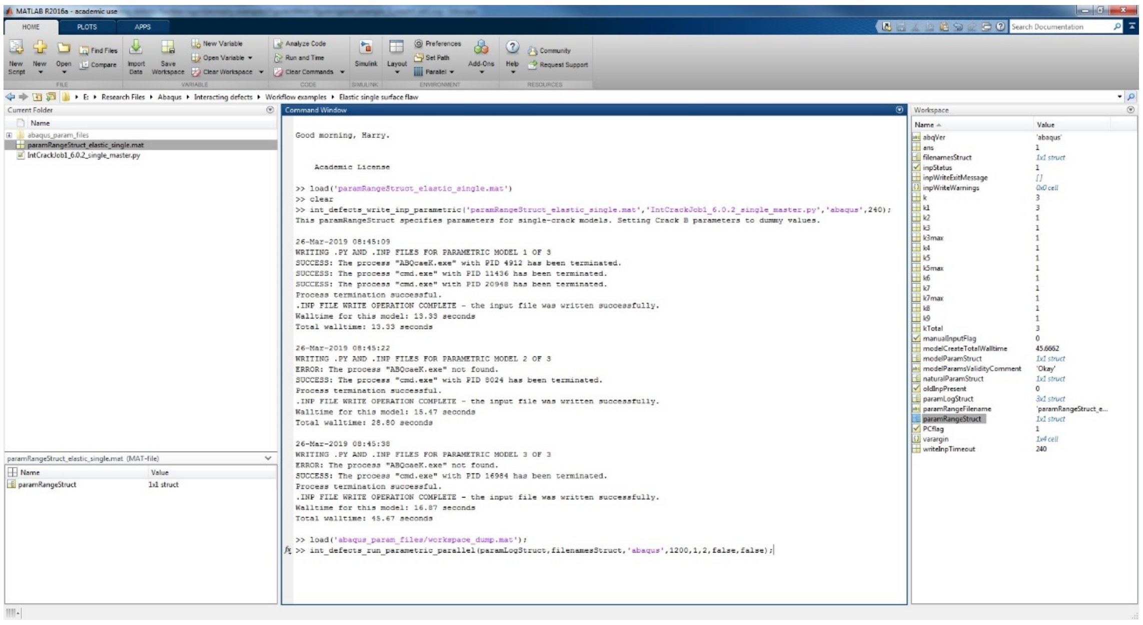
4. Run `int_defects_run_parametric_parallel`. This will run the Abaqus/Standard analysis.



```
Good morning, Harry.


   Academic License

>> load('paramRangeStruct_elastic_single.mat')
>> clear
>> int_defects_write_inp_parametric('paramRangeStruct_elastic_single.mat','IntCrackJob1_6.0.2_single_master.py','abaqus',240);
This paramRangeStruct specifies parameters for single-crack models. Setting Crack B parameters to dummy values.

26-Mar-2019 08:45:09
WRITING .PY AND .INP FILES FOR PARAMETRIC MODEL 1 OF 3
SUCCESS: The process "ABQcaeK.exe" with PID 4912 has been terminated.
SUCCESS: The process "cmd.exe" with PID 11436 has been terminated.
SUCCESS: The process "cmd.exe" with PID 20948 has been terminated.
Process termination successful.
.INP FILE WRITE OPERATION COMPLETE - the input file was written successfully.
Walltime for this model: 13.33 seconds
Total walltime: 13.33 seconds

26-Mar-2019 08:45:22
WRITING .PY AND .INP FILES FOR PARAMETRIC MODEL 2 OF 3
ERROR: The process "ABQcaeK.exe" not found.
SUCCESS: The process "cmd.exe" with PID 8024 has been terminated.
Process termination successful.
.INP FILE WRITE OPERATION COMPLETE - the input file was written successfully.
Walltime for this model: 15.47 seconds
Total walltime: 28.80 seconds

26-Mar-2019 08:45:38
WRITING .PY AND .INP FILES FOR PARAMETRIC MODEL 3 OF 3
ERROR: The process "ABQcaeK.exe" not found.
SUCCESS: The process "cmd.exe" with PID 16984 has been terminated.
Process termination successful.
.INP FILE WRITE OPERATION COMPLETE - the input file was written successfully.
Walltime for this model: 16.87 seconds
Total walltime: 45.67 seconds

>> load('abaqus_param_files/workspace_dump.mat');
>> int_defects_run_parametric_parallel(paramLogStruct,filenamesStruct,'abaqus',1200,1,2,false,false);
```

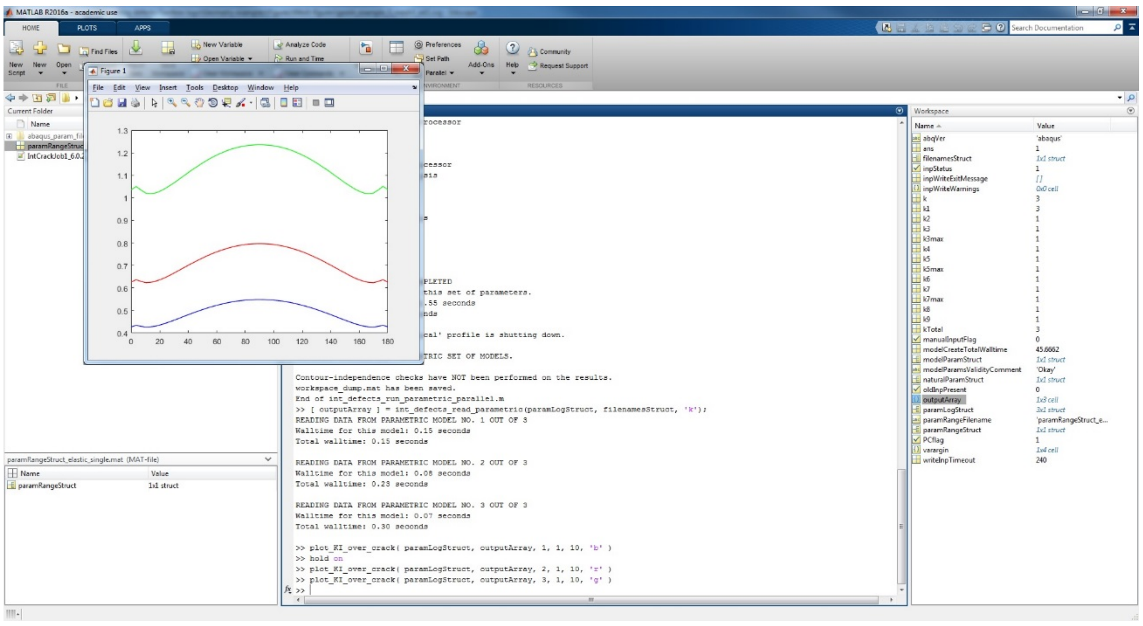
Name	Value
abqVer	'abaqus'
ans	1
filenamesStruct	1x1 struct
inpStatus	1
inpWriteExitMessage	[]
inpWriteWarnings	0x0 cell
k	3
k1	3
k2	1
k3	1
k3max	1
k4	1
k5	1
k5max	1
k6	1
k7	1
k7max	1
k8	1
k9	1
kTotal	3
manualInputFlag	0
modelCreateTotalWalltime	45.6662
modelParamStruct	1x1 struct
modelParamsValidityComment	'Okay'
naturalParamStruct	1x1 struct
oldInpPresent	0
paramLogStruct	3x1 struct
paramRangeFilename	'paramRangeStruct_e...
paramRangeStruct	1x1 struct
PCflag	1
varargin	1x4 cell
writeInpTimeout	240

5. Run `int_defects_read_parametric`. This will read the results from the .dat files into the array outputArray.



```
Good morning, Harry.


   Academic License

>> load('paramRangeStruct_elastic_single.mat')
>> clear
>> int_defects_write_inp_parametric('paramRangeStruct_elastic_single.mat','IntCrackJob1_6.0.2_single_master.py','abaqus',240);
This paramRangeStruct specifies parameters for single-crack models. Setting Crack B parameters to dummy values.

26-Mar-2019 08:45:09
WRITING .PY AND .INP FILES FOR PARAMETRIC MODEL 1 OF 3
SUCCESS: The process "ABQcaeK.exe" with PID 4912 has been terminated.
SUCCESS: The process "cmd.exe" with PID 11436 has been terminated.
SUCCESS: The process "cmd.exe" with PID 20948 has been terminated.
Process termination successful.
.INP FILE WRITE OPERATION COMPLETE - the input file was written successfully.
Walltime for this model: 13.33 seconds
Total walltime: 13.33 seconds

26-Mar-2019 08:45:22
WRITING .PY AND .INP FILES FOR PARAMETRIC MODEL 2 OF 3
ERROR: The process "ABQcaeK.exe" not found.
SUCCESS: The process "cmd.exe" with PID 8024 has been terminated.
Process termination successful.
.INP FILE WRITE OPERATION COMPLETE - the input file was written successfully.
Walltime for this model: 15.47 seconds
Total walltime: 28.80 seconds

26-Mar-2019 08:45:38
WRITING .PY AND .INP FILES FOR PARAMETRIC MODEL 3 OF 3
ERROR: The process "ABQcaeK.exe" not found.
SUCCESS: The process "cmd.exe" with PID 16984 has been terminated.
Process termination successful.
.INP FILE WRITE OPERATION COMPLETE - the input file was written successfully.
Walltime for this model: 16.87 seconds
Total walltime: 45.67 seconds

>> load('abaqus_param_files/workspace_dump.mat');
>> int_defects_run_parametric_parallel(paramLogStruct,filenamesStruct,'abaqus',1200,1,2,false,false);
```

6. You can plot the SIF as a function of position on the crack tip (given by the ellipse parametric angle) using `plot_KI_over_crack`. The SIF data can be found in: mainStruct.outputArray.


MATLAB R2016a - academic use
HOME
PLOTS
APPS
Figure 1
Current Folder
Workspace
Search Documentation
paramRangeStruct_elastic_single.mat (MAT-file)

```
rocessor

cessor
sis

s

PLETED
this set of parameters.
.55 seconds
nds

cal' profile is shutting down.

TRIC SET OF MODELS.

Contour-independence checks have NOT been performed on the results.
workspace_dump.mat has been saved.
End of int_defects_run_parametric_parallel.m
>> [ outputArray ] = int_defects_read_parametric(paramLogStruct, filenamesStruct, 'k');
READING DATA FROM PARAMETRIC MODEL NO. 1 OUT OF 3
Walltime for this model: 0.15 seconds
Total walltime: 0.15 seconds

READING DATA FROM PARAMETRIC MODEL NO. 2 OUT OF 3
Walltime for this model: 0.08 seconds
Total walltime: 0.23 seconds

READING DATA FROM PARAMETRIC MODEL NO. 3 OUT OF 3
Walltime for this model: 0.07 seconds
Total walltime: 0.30 seconds

>> plot_KI_over_crack( paramLogStruct, outputArray, 1, 1, 10, 'b' )
>> hold on
>> plot_KI_over_crack( paramLogStruct, outputArray, 2, 1, 10, 'r' )
>> plot_KI_over_crack( paramLogStruct, outputArray, 3, 1, 10, 'g' )
>>
```

Name	Value
paramRangeStruct	1x1 struct

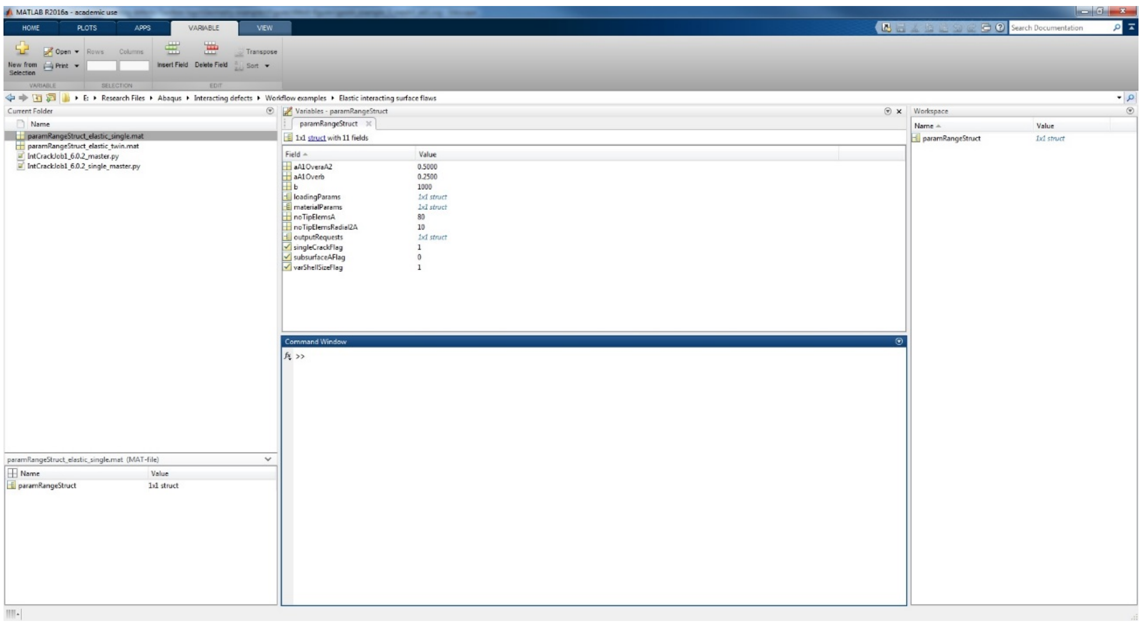
Name	Value
abqVer	'abaqus'
ans	1
filenamesStruct	1x1 struct
inpStatus	1
inpWriteExitMessage	[]
inpWriteWarnings	0x0 cell
k	3
k1	3
k2	1
k3	1
k3max	1
k4	1
k5	1
k5max	1
k6	1
k7	1
k7max	1
k8	1
k9	1
kTotal	3
manualInputFlag	0
modelCreateTotalWalltime	45.6662
modelParamStruct	1x1 struct
modelParamsValidityComment	'Okay'
naturalParamStruct	1x1 struct
oldInpPresent	0
outputArray	1x3 cell
paramLogStruct	3x1 struct
paramRangeFilename	'paramRangeStruct_e...
paramRangeStruct	1x1 struct
PCflag	1
varargin	1x4 cell
writeInpTimeout	240

A.2 Determining elastic interaction factor for a flaw pair (with varying separation distance)

To determine the elastic interaction factors for the flaw pair, we will create two sets of models: one set which contains all the pairs of interacting surface flaws what we want to analyse, and one set which contains single surface flaws. The set of single surface flaws must contain flaws with all the sizes and aspect ratios that occur in the twin flaw models. In the case where there is interaction between an embedded and a surface defect, a third set of models is required (all sizes/aspect ratios of the embedded defects alone).

In this example, the surface flaws always have the same depth (a/b = 0.25) and aspect ratio (a/c = 0.5). Three different flaw spacings are investigated (d/b = 0.125, 0.25, 0.5).

1. Copy the master python files “IntCrackJob1_6.0.4_single_master.py” and “IntCrackJob1_6.0.4_master.py” into the working directory.
2. Using MATLAB, construct the data structure paramRangeStruct defining the model set of *single* flaws. Save it in a .mat file (paramRange_struct_elastic_single.mat here).



MATLAB R2016a - academic use

E: ▸ Research Files ▸ Abaqus ▸ Interacting defects ▸ Workflow examples ▸ Elastic interacting surface flaws

Current Folder

- paramRangeStruct_elastic_single.mat
- paramRangeStruct_elastic_twin.mat
- IntCrackJob1_6.0.2_master.py
- IntCrackJob1_6.0.2_single_master.py

paramRangeStruct_elastic_single.mat (MAT-file)

Name	Value
paramRangeStruct	1x1 struct

Variables - paramRangeStruct

1x1 struct with 11 fields

Field	Value
aA1OveraA2	0.5000
aA1Overb	0.2500
b	1000
loadingParams	1x1 struct
materialParams	1x1 struct
noTipElemsA	80
noTipElemsRadial2A	10
outputRequests	1x1 struct
singleCrackFlag	1
subsurfaceAFlag	0
varShellSizeFlag	1

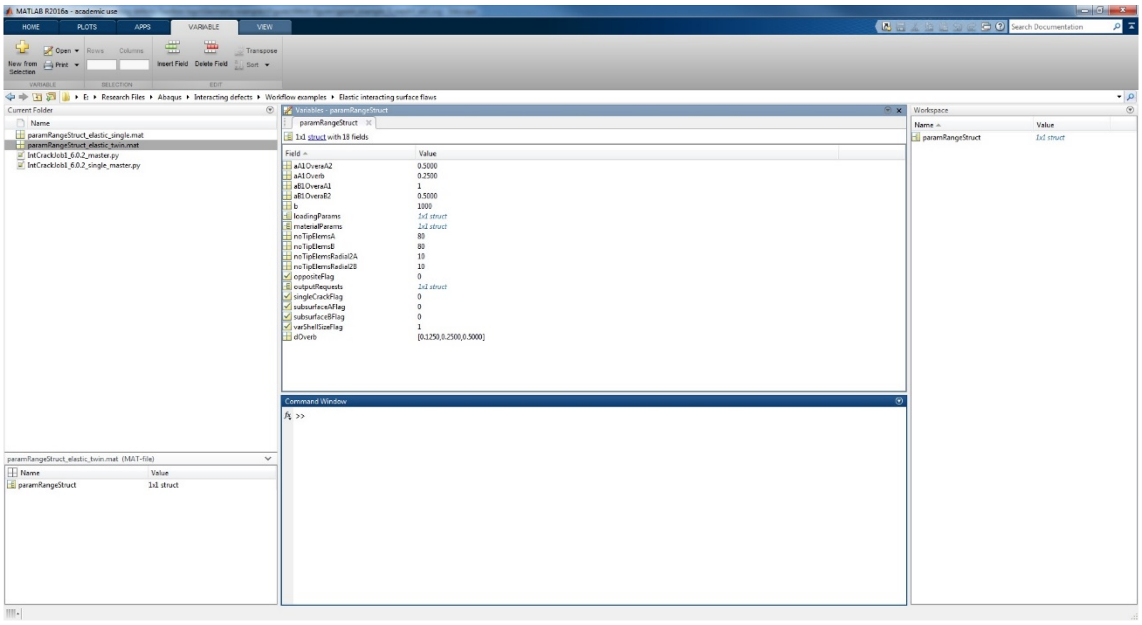
Command Window

```
>>
```

Workspace

Name	Value
paramRangeStruct	1x1 struct

3. Using MATLAB, construct the data structure paramRangeStruct defining the model set of *interacting* flaws. Save it in a .mat file (paramRange_struct_elastic_twin.mat here).



MATLAB R2016a - academic use

E: ▸ Research Files ▸ Abaqus ▸ Interacting defects ▸ Workflow examples ▸ Elastic interacting surface flaws

Current Folder

- paramRangeStruct_elastic_single.mat
- paramRangeStruct_elastic_twin.mat
- IntCrackJob1_6.0.2_master.py
- IntCrackJob1_6.0.2_single_master.py

paramRangeStruct_elastic_twin.mat (MAT-file)

Name	Value
paramRangeStruct	1x1 struct

Variables - paramRangeStruct

1x1 struct with 18 fields

Field	Value
aA1OveraA2	0.5000
aA1Overb	0.2500
aB1OveraA1	1
aB1OveraB2	0.5000
b	1000
loadingParams	1x1 struct
materialParams	1x1 struct
noTipElemsA	80
noTipElemsB	80
noTipElemsRadial2A	10
noTipElemsRadial2B	10
oppositeFlag	0
outputRequests	1x1 struct
singleCrackFlag	0
subsurfaceAFlag	0
subsurfaceBFlag	0
varShellSizeFlag	1
dOverb	[0.1250,0.2500,0.5000]

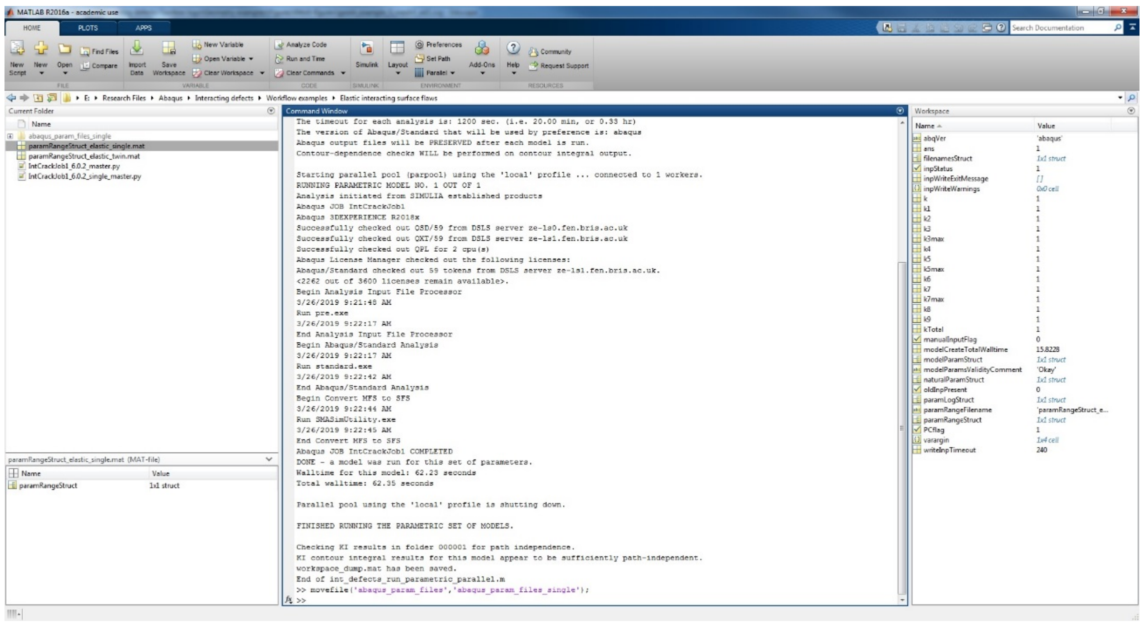
Command Window

```
>>
```

Workspace

Name	Value
paramRangeStruct	1x1 struct

4. Run `int_defects_write_inp_parametric` for the single flaws, then run `int_defects_run_parametric_parallel` to perform the Abaqus/Standard analysis for single flaws. Once this is finished, rename the created directory “abaqus_param_files” to “abaqus_param_files_single”.



E: ▸ Research Files ▸ Abaqus ▸ Interacting defects ▸ Workflow examples ▸ Elastic interacting surface flaws

Current Folder:
- abaqus_param_files_single
- paramRangeStruct_elastic_single.mat
- paramRangeStruct_elastic_twin.mat
- IntCrackJob1_6.0.2_master.py
- IntCrackJob1_6.0.2_single_master.py

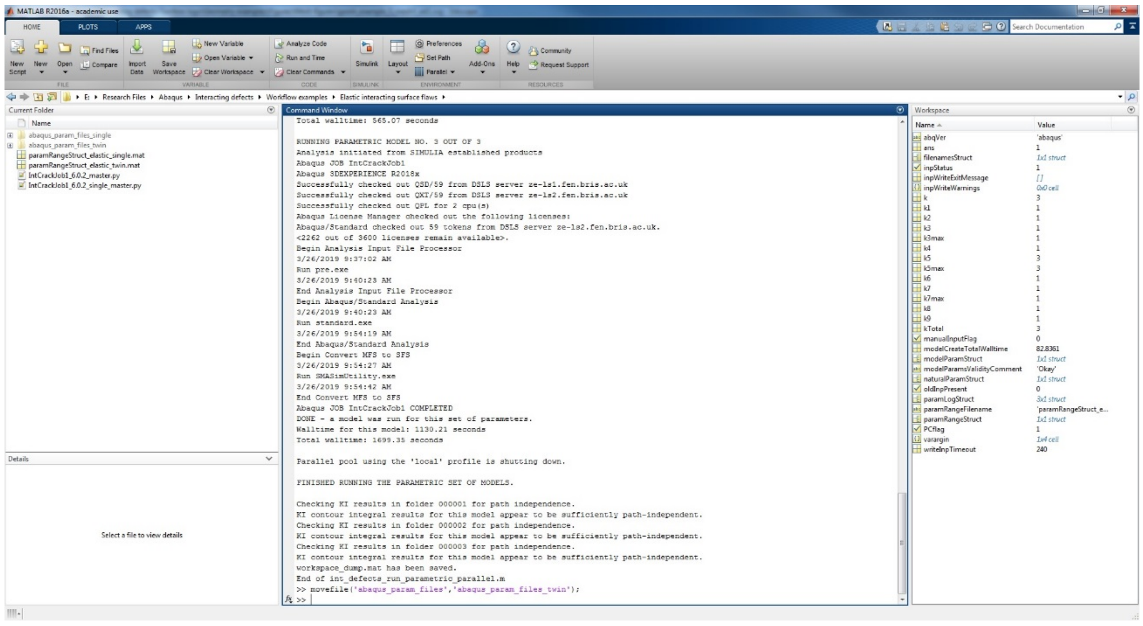
paramRangeStruct_elastic_single.mat (MAT-file)

Name	Value
paramRangeStruct	1x1 struct

```
The timeout for each analysis is: 1200 sec. (i.e. 20.00 min, or 0.33 hr)
The version of Abaqus/Standard that will be used by preference is: abaqus
Abaqus output files will be PRESERVED after each model is run.
Contour-dependence checks WILL be performed on contour integral output.

Starting parallel pool (parpool) using the 'local' profile ... connected to 1 workers.
RUNNING PARAMETRIC MODEL NO. 1 OUT OF 1
Analysis initiated from SIMULIA established products
Abaqus JOB IntCrackJob1
Abaqus 3DEXPERIENCE R2018x
Successfully checked out QSD/59 from DSLS server ze-ls0.fen.bris.ac.uk
Successfully checked out QXT/59 from DSLS server ze-ls1.fen.bris.ac.uk
Successfully checked out QPL for 2 cpu(s)
Abaqus License Manager checked out the following licenses:
Abaqus/Standard checked out 59 tokens from DSLS server ze-ls1.fen.bris.ac.uk.
<2262 out of 3600 licenses remain available>.
Begin Analysis Input File Processor
3/26/2019 9:21:48 AM
Run pre.exe
3/26/2019 9:22:17 AM
End Analysis Input File Processor
Begin Abaqus/Standard Analysis
3/26/2019 9:22:17 AM
Run standard.exe
3/26/2019 9:22:42 AM
End Abaqus/Standard Analysis
Begin Convert MFS to SFS
3/26/2019 9:22:44 AM
Run SMASimUtility.exe
3/26/2019 9:22:45 AM
End Convert MFS to SFS
Abaqus JOB IntCrackJob1 COMPLETED
DONE - a model was run for this set of parameters.
Walltime for this model: 62.23 seconds
Total walltime: 62.35 seconds

Parallel pool using the 'local' profile is shutting down.

FINISHED RUNNING THE PARAMETRIC SET OF MODELS.

Checking KI results in folder 000001 for path independence.
KI contour integral results for this model appear to be sufficiently path-independent.
workspace_dump.mat has been saved.
End of int_defects_run_parametric_parallel.m
>> movefile('abaqus_param_files','abaqus_param_files_single');
>>
```

Name	Value
abqVer	'abaqus'
ans	1
filenamesStruct	1x1 struct
inpStatus	1
inpWriteExitMessage	[]
inpWriteWarnings	0x0 cell
k	1
k1	1
k2	1
k3	1
k3max	1
k4	1
k5	1
k5max	1
k6	1
k7	1
k7max	1
k8	1
k9	1
kTotal	1
manualInputFlag	0
modelCreateTotalWalltime	15.8228
modelParamStruct	1x1 struct
modelParamsValidityComment	'Okay'
naturalParamStruct	1x1 struct
oldInpPresent	0
paramLogStruct	1x1 struct
paramRangeFilename	'paramRangeStruct_e...
paramRangeStruct	1x1 struct
PCflag	1
varargin	1x4 cell
writeInpTimeout	240

5. Follow Step 3 for the twin flaw model set, this time renaming “abaqus_param_files” to “abaqus_param_files_twin”.



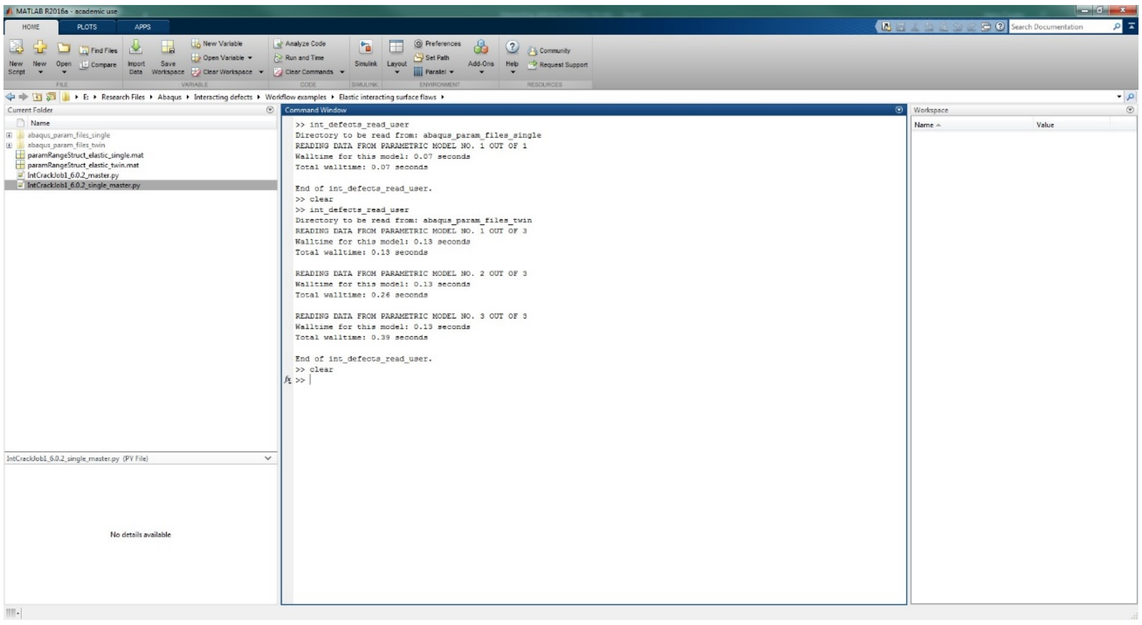
E: ▸ Research Files ▸ Abaqus ▸ Interacting defects ▸ Workflow examples ▸ Elastic interacting surface flaws ▸

Current Folder:
- abaqus_param_files_single
- abaqus_param_files_twin
- paramRangeStruct_elastic_single.mat
- paramRangeStruct_elastic_twin.mat
- IntCrackJob1_6.0.2_master.py
- IntCrackJob1_6.0.2_single_master.py

```
Total walltime: 565.07 seconds

RUNNING PARAMETRIC MODEL NO. 3 OUT OF 3
Analysis initiated from SIMULIA established products
Abaqus JOB IntCrackJob1
Abaqus 3DEXPERIENCE R2018x
Successfully checked out QSD/59 from DSLS server ze-ls1.fen.bris.ac.uk
Successfully checked out QXT/59 from DSLS server ze-ls2.fen.bris.ac.uk
Successfully checked out QPL for 2 cpu(s)
Abaqus License Manager checked out the following licenses:
Abaqus/Standard checked out 59 tokens from DSLS server ze-ls2.fen.bris.ac.uk.
<2262 out of 3600 licenses remain available>.
Begin Analysis Input File Processor
3/26/2019 9:37:02 AM
Run pre.exe
3/26/2019 9:40:23 AM
End Analysis Input File Processor
Begin Abaqus/Standard Analysis
3/26/2019 9:40:23 AM
Run standard.exe
3/26/2019 9:54:19 AM
End Abaqus/Standard Analysis
Begin Convert MFS to SFS
3/26/2019 9:54:27 AM
Run SMASimUtility.exe
3/26/2019 9:54:42 AM
End Convert MFS to SFS
Abaqus JOB IntCrackJob1 COMPLETED
DONE - a model was run for this set of parameters.
Walltime for this model: 1130.21 seconds
Total walltime: 1699.35 seconds

Parallel pool using the 'local' profile is shutting down.

FINISHED RUNNING THE PARAMETRIC SET OF MODELS.

Checking KI results in folder 000001 for path independence.
KI contour integral results for this model appear to be sufficiently path-independent.
Checking KI results in folder 000002 for path independence.
KI contour integral results for this model appear to be sufficiently path-independent.
Checking KI results in folder 000003 for path independence.
KI contour integral results for this model appear to be sufficiently path-independent.
workspace_dump.mat has been saved.
End of int_defects_run_parametric_parallel.m
>> movefile('abaqus_param_files','abaqus_param_files_twin');
>>
```

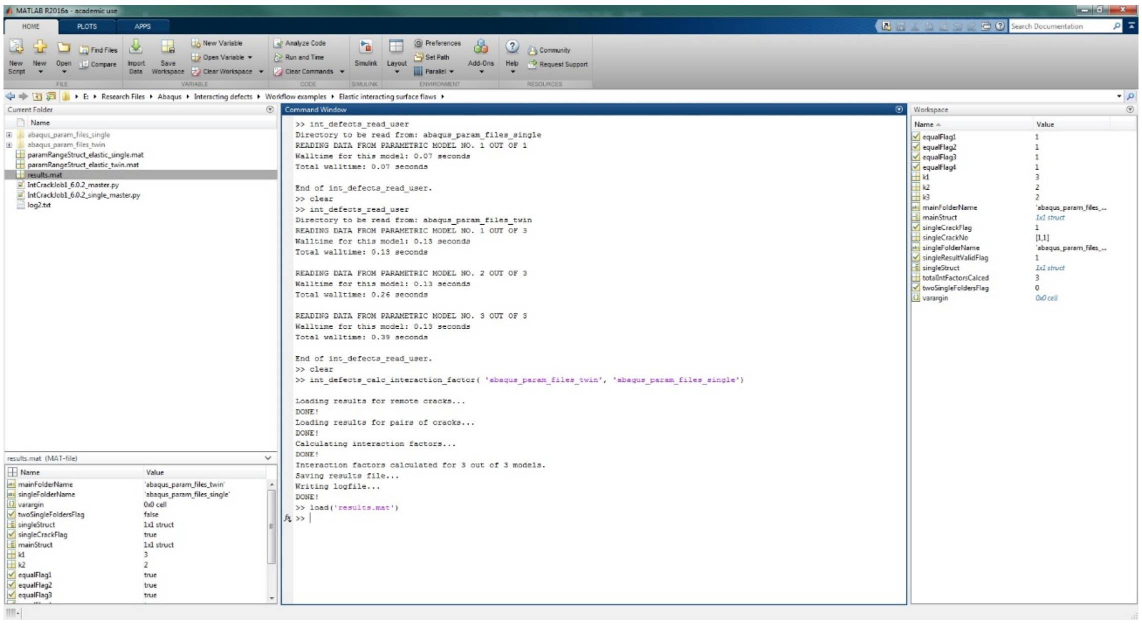
Name	Value
abqVer	'abaqus'
ans	1
filenamesStruct	1x1 struct
inpStatus	1
inpWriteExitMessage	[]
inpWriteWarnings	0x0 cell
k	3
k1	1
k2	1
k3	1
k3max	1
k4	1
k5	3
k5max	3
k6	1
k7	1
k7max	1
k8	1
k9	1
kTotal	3
manualInputFlag	0
modelCreateTotalWalltime	82.8361
modelParamStruct	1x1 struct
modelParamsValidityComment	'Okay'
naturalParamStruct	1x1 struct
oldInpPresent	0
paramLogStruct	3x1 struct
paramRangeFilename	'paramRangeStruct_e...
paramRangeStruct	1x1 struct
PCflag	1
varargin	1x4 cell
writeInpTimeout	240

6. Run `int_defects_read_user` once for each of abaqus_param_files_single and abaqus_param_files_twin. This will extract the results from each model set, and save them in the workspace_dump.mat file in each directoy.



```
>> int_defects_read_user
Directory to be read from: abaqus_param_files_single
READING DATA FROM PARAMETRIC MODEL NO. 1 OUT OF 1
Walltime for this model: 0.07 seconds
Total walltime: 0.07 seconds

End of int_defects_read_user.
>> clear
>> int_defects_read_user
Directory to be read from: abaqus_param_files_twin
READING DATA FROM PARAMETRIC MODEL NO. 1 OUT OF 3
Walltime for this model: 0.13 seconds
Total walltime: 0.13 seconds

READING DATA FROM PARAMETRIC MODEL NO. 2 OUT OF 3
Walltime for this model: 0.13 seconds
Total walltime: 0.26 seconds

READING DATA FROM PARAMETRIC MODEL NO. 3 OUT OF 3
Walltime for this model: 0.13 seconds
Total walltime: 0.39 seconds

End of int_defects_read_user.
>> clear
>>
```

7. Run `int_defects_calc_interaction_factor`. This will calculate interaction factors. It will write a results file (“results.mat”) into the current directory, along with a logfile “log2.txt”.



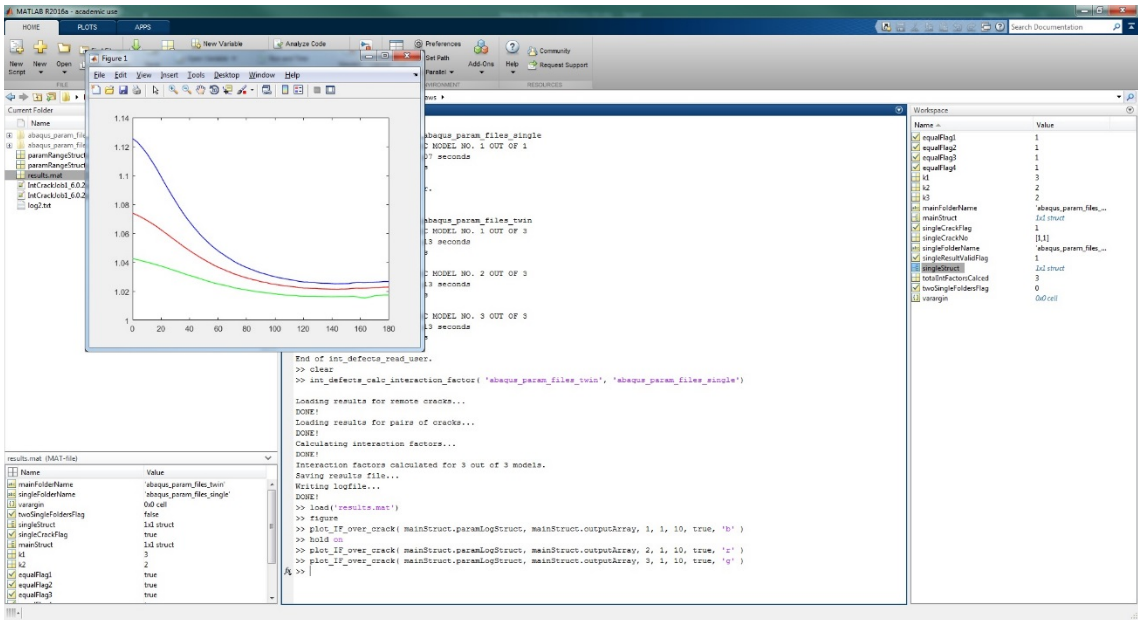
```
>> int_defects_read_user
Directory to be read from: abaqus_param_files_single
READING DATA FROM PARAMETRIC MODEL NO. 1 OUT OF 1
Walltime for this model: 0.07 seconds
Total walltime: 0.07 seconds

End of int_defects_read_user.
>> clear
>> int_defects_read_user
Directory to be read from: abaqus_param_files_twin
READING DATA FROM PARAMETRIC MODEL NO. 1 OUT OF 3
Walltime for this model: 0.13 seconds
Total walltime: 0.13 seconds

READING DATA FROM PARAMETRIC MODEL NO. 2 OUT OF 3
Walltime for this model: 0.13 seconds
Total walltime: 0.26 seconds

READING DATA FROM PARAMETRIC MODEL NO. 3 OUT OF 3
Walltime for this model: 0.13 seconds
Total walltime: 0.39 seconds

End of int_defects_read_user.
>> clear
>> int_defects_calc_interaction_factor( 'abaqus_param_files_twin', 'abaqus_param_files_single')

Loading results for remote cracks...
DONE!
Loading results for pairs of cracks...
DONE!
Calculating interaction factors...
DONE!
Interaction factors calculated for 3 out of 3 models.
Saving results file...
Writing logfile...
DONE!
>> load('results.mat')
>>
```

Name	Value
equalFlag1	1
equalFlag2	1
equalFlag3	1
equalFlag4	1
k1	3
k2	2
k3	2
mainFolderName	'abaqus_param_files_...
mainStruct	1x1 struct
singleCrackFlag	1
singleCrackNo	[1,1]
singleFolderName	'abaqus_param_files_...
singleResultValidFlag	1
singleStruct	1x1 struct
totalIntFactorsCalced	3
twoSingleFoldersFlag	0
varargin	0x0 cell

results.mat (MAT-file)

Name	Value
mainFolderName	'abaqus_param_files_twin'
singleFolderName	'abaqus_param_files_single'
varargin	0x0 cell
twoSingleFoldersFlag	false
singleStruct	1x1 struct
singleCrackFlag	true
mainStruct	1x1 struct
k1	3
k2	2
equalFlag1	true
equalFlag2	true
equalFlag3	true

8. You can now load the results in results.mat and run `plot_IF_over_crack` to generate plots of the interaction factor vs position on the crack tip line. The interaction factor data can be found in mainStruct.outputArray.



```
abaqus_param_files_single
C MODEL NO. 1 OUT OF 1
07 seconds

abaqus_param_files_twin
C MODEL NO. 1 OUT OF 3
13 seconds

C MODEL NO. 2 OUT OF 3
13 seconds

C MODEL NO. 3 OUT OF 3
13 seconds

End of int_defects_read_user.
>> clear
>> int_defects_calc_interaction_factor( 'abaqus_param_files_twin', 'abaqus_param_files_single')

Loading results for remote cracks...
DONE!
Loading results for pairs of cracks...
DONE!
Calculating interaction factors...
DONE!
Interaction factors calculated for 3 out of 3 models.
Saving results file...
Writing logfile...
DONE!
>> load('results.mat')
>> figure
>> plot_IF_over_crack( mainStruct.paramLogStruct, mainStruct.outputArray, 1, 1, 10, true, 'b' )
>> hold on
>> plot_IF_over_crack( mainStruct.paramLogStruct, mainStruct.outputArray, 2, 1, 10, true, 'r' )
>> plot_IF_over_crack( mainStruct.paramLogStruct, mainStruct.outputArray, 3, 1, 10, true, 'g' )
>>
```

results.mat (MAT-file)

Name	Value
mainFolderName	'abaqus_param_files_twin'
singleFolderName	'abaqus_param_files_single'
varargin	0x0 cell
twoSingleFoldersFlag	false
singleStruct	1x1 struct
singleCrackFlag	true
mainStruct	1x1 struct
k1	3
k2	2
equalFlag1	true
equalFlag2	true
equalFlag3	true

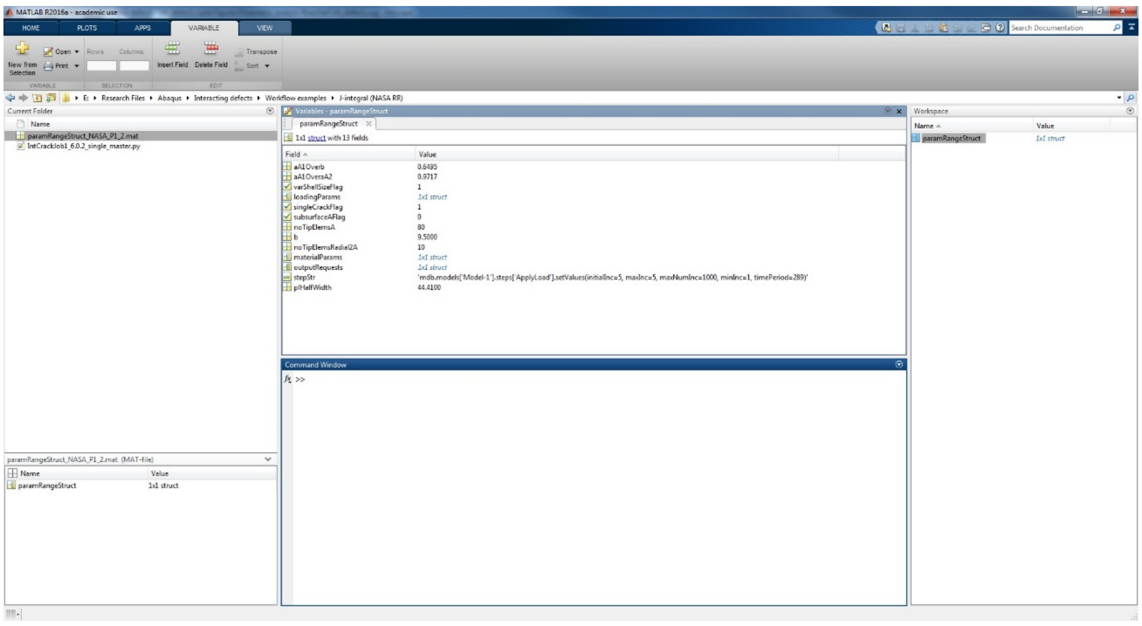
Workspace

Name	Value
equalFlag1	1
equalFlag2	1
equalFlag3	1
equalFlag4	1
k1	3
k2	2
k3	2
mainFolderName	'abaqus_param_files_...
mainStruct	1x1 struct
singleCrackFlag	1
singleCrackNo	[1,1]
singleFolderName	'abaqus_param_files_...
singleResultValidFlag	1
singleStruct	1x1 struct
totalIntFactorsCalced	3
twoSingleFoldersFlag	0
varargin	0x0 cell

A.3 Determining the J-integral at a given load for a single flaw or flaw pair

This example generates and solves a model of the NASA Analytical Round-Robin for Analysis of Surface Cracked Plates (Phase 1) [11]. It performs an elastic-plastic analysis of a single model; there is no variation in loading conditions/material/geometry, which are all known (they are specified in the round-robin protocol).

1. Copy the master python file “IntCrackJob1_6.0.4_single_master.py” into the working directory.
2. Using MATLAB, construct the data structure paramRangeStruct defining the model of the Phase 1 round-robin case. Elastic-plastic material behaviour should be defined using a table of true stress vs true strain values (paramRangeStruct.materialParams.plasticTable). Note that this defines the material’s plastic behaviour using incremental plasticity theory, with a von Mises yield locus and an isotropic hardening characteristic. J-integral output should be requested (paramRangeStruct.outputRequests.contourNameA = ’CrackAOutJ’ and paramRangeStruct.outputRequests.contoutTypeA = ‘J_INTEGRAL’). To ensure many increments of J-integral output during a ramp-up of loading, a maximum increment size should be set: paramRangeStruct.stepStr = ‘mdb.models['Model-1'].steps['ApplyLoad'].setValues(initialInc=5, maxInc=5, maxNumInc=1000, minInc=1, timePeriod=289)’.



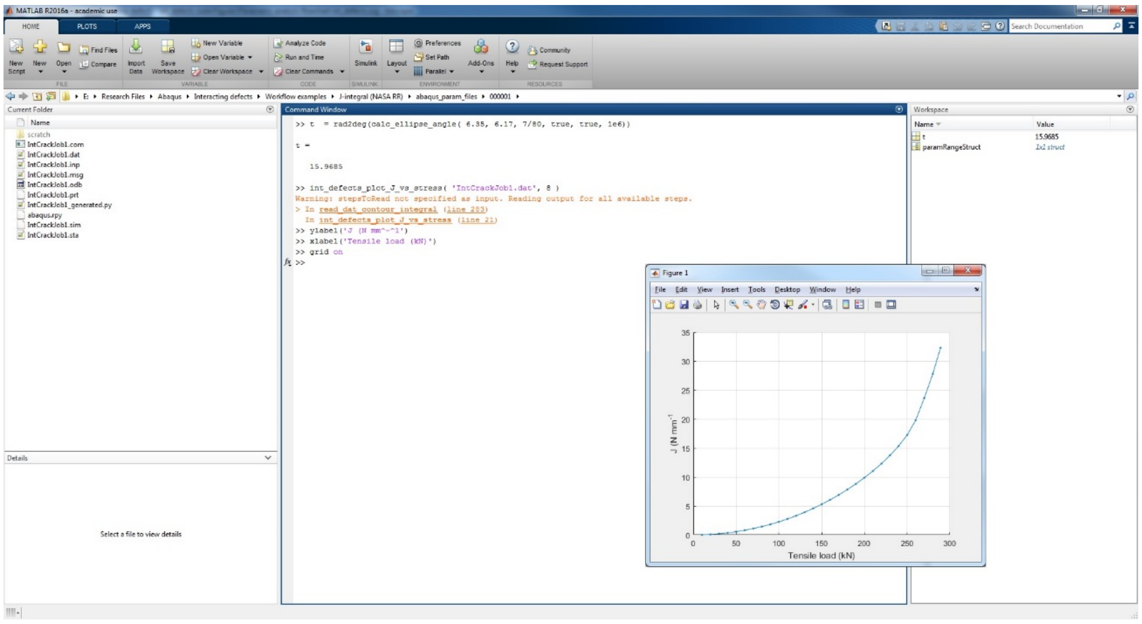
1x1 struct with 13 fields

Field	Value
aA1Overb	0.6495
aA1OveraA2	0.9717
varShellSizeFlag	1
loadingParams	1x1 struct
singleCrackFlag	1
subsurfaceAFlag	0
noTipElemsA	80
b	9.5000
noTipElemsRadial2A	10
materialParams	1x1 struct
outputRequests	1x1 struct
stepStr	'mdb.models['Model-1'].steps['ApplyLoad'].setValues(initialInc=5, maxInc=5, maxNumInc=1000, minInc=1, timePeriod=289)'
plHalfWidth	44.4100

3. Create and run the models. For this example, model creation and execution was performed on a remote Linux server using the following commands:

```
int_defects_write_inp_parametric('paramRangeStruct_NASA_P1_2.mat','IntCrackJob1_6.0.4_single_master.py','abaqus',600);

load('abaqus_param_files/workspace_dump.mat');

int_defects_run_parametric_parallel(paramLogStruct,filenamesStruct,'abaqus',44000,1,4,false,true);
```

4. Navigate to the model directory for Model #1 and run `int_defects_plot_J_vs_stress` to plot J vs step time (which is equivalent to the applied load in kN in this case). In this example, it has been determined that the 8th node along the crack tip line is at an ellipse parametric angle of 15.97°. This is close to 17° which was used for comparison in the NASA round robin, and so the J-integral for Node 8 is plotted:



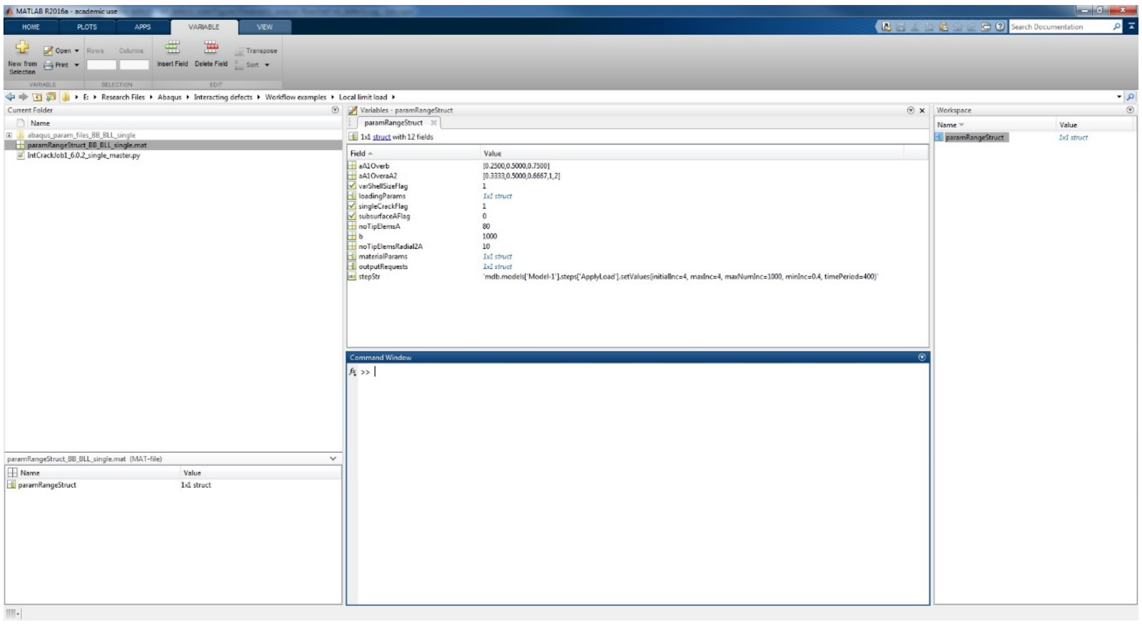
```
>> t  = rad2deg(calc_ellipse_angle( 6.35, 6.17, 7/80, true, true, 1e6))

t =

   15.9685

>> int_defects_plot_J_vs_stress( 'IntCrackJob1.dat', 8 )
Warning: stepsToRead not specified as input. Reading output for all available steps.
> In read_dat_contour_integral (line 283)
  In int_defects_plot_J_vs_stress (line 21)
>> ylabel('J (N mm^-^1')
>> xlabel('Tensile load (kN)')
>> grid on
>>
```

The J-integral result shown in the figure above agrees well with round-robin FEA results from other labs (see [11], p20, Figure 10a).

A.4 Determining the local limit load for a plate containing a single semi-elliptical surface flaw (varying flaw depth and aspect ratio)

You can use `int_defects` for determining the Local Limit Load (LLL) for flawed plates, i.e. the load at which an unbroken plastic ligament extends from the flaw to the back face of the plate. LLLs can be used in structural integrity analysis to conservatively estimate the plastic collapse parameter L_r [1]. This example calculates the LLL (expressed as a limit stress) for 15 different single semi-elliptical surface flaws in a wide plate under tension: all combinations of a/b = 0.25, 0.5, 0.75 and a/c = 0.3333, 0.5, 0.6667, 1, 2. Material yielding is predicted using a von Mises yield locus.

1. Copy the master python file “IntCrackJob1_6.0.4_single_master.py” into the working directory.
2. Using MATLAB, construct the data structure paramRangeStruct defining the models. Parameters defining a/b and a/c are given as vectors. The material is defined as being elastic-perfectly-plastic (i.e. no strain-hardening) with a yield stress of 360 MPa by setting: paramRangeStruct.materialParams.plasticTable = [360,0]. Von Mises equivalent plastic strain field output is requested for the whole model using:
   paramRangeStruct.outputRequests.custom = ‘** OUTPUT REQUESTS
    *El Print, Elset = PlatePartitioned-1.Set-1, Position = Centroidal, Frequency=1
    PEEQ’. No contour integral output was requested:
   paramRangeStruct.outputRequests.contourTypeA = ‘none’.



Variables - paramRangeStruct

1x1 struct with 12 fields

Field	Value
aA1Overb	[0.2500,0.5000,0.7500]
aA1OveraA2	[0.3333,0.5000,0.6667,1,2]
varShellSizeFlag	1
loadingParams	1x1 struct
singleCrackFlag	1
subsurfaceAFlag	0
noTipElemsA	80
b	1000
noTipElemsRadial2A	10
materialParams	1x1 struct
outputRequests	1x1 struct
stepStr	'mdb.models['Model-1'].steps['ApplyLoad'].setValues(initialInc=4, maxInc=4, maxNumInc=1000, minInc=0.4, timePeriod=400)'

Current Folder: abaqus_param_files_BB_BLL_single, paramRangeStruct_BB_BLL_single.mat, IntCrackJob1_6.0.2_single_master.py

paramRangeStruct_BB_BLL_single.mat (MAT-file)

Name	Value
paramRangeStruct	1x1 struct

3. Create and run the models. For this example, model creation and execution was performed on a remote Linux server using the following commands:

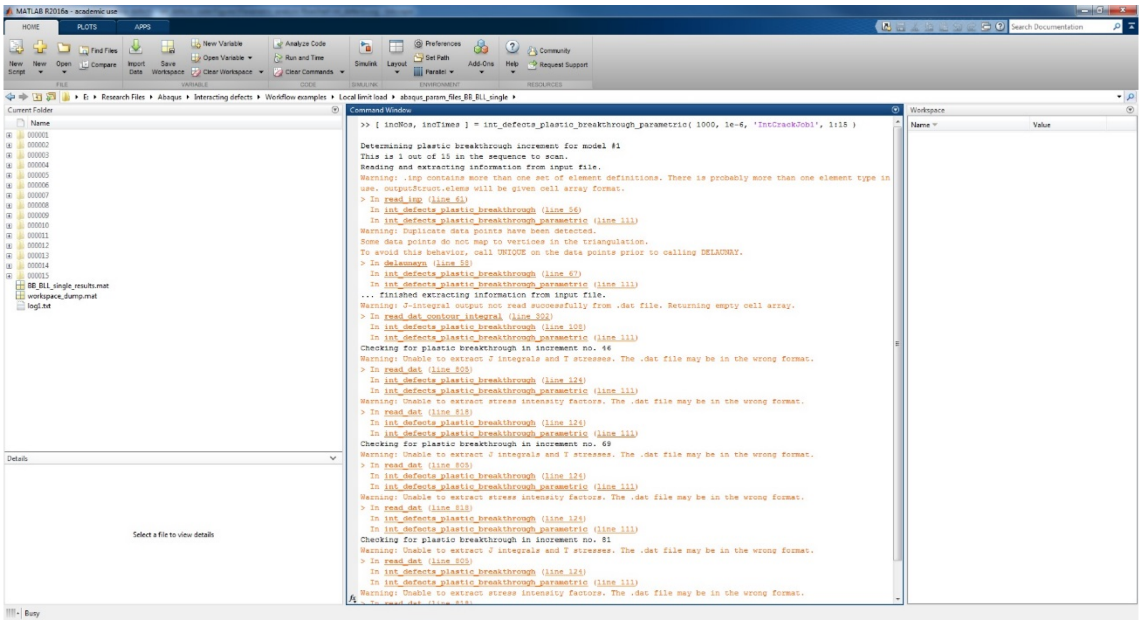
```
int_defects_write_inp_parametric('paramRangeStruct_BB_BLL_single.mat
','IntCrackJob1_6.0.4_single_master.py','abaqus',600);
```

```
load('abaqus_param_files/workspace_dump.mat');
```

```
int_defects_run_parametric_parallel(paramLogStruct,filenamesStruct,'
abaqus',44000,2,4,false,false);
```

Note that in the call of `int_defects_run_parametric_parallel`, contDepCheckFlag = false. There should be no contour integral data in the .dat file, and so this does not need to be checked for contour-dependence.

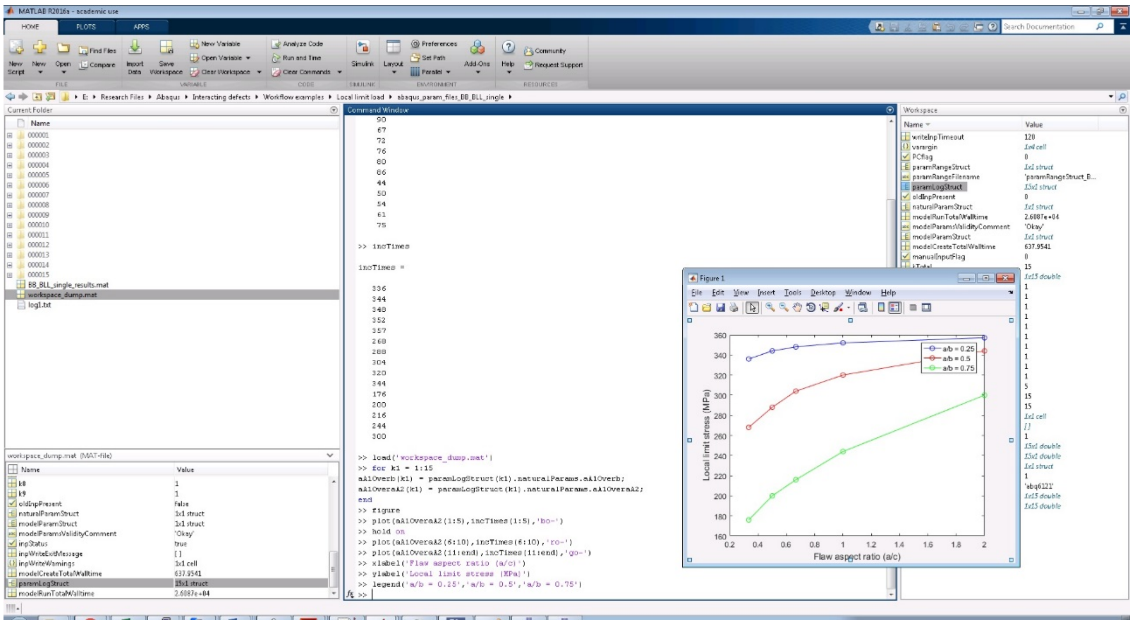
4. Navigate into the abaqus_param_files directory and use `int_defects_plastic_breakthrough_parametric` to determine plastic limit loads. This may take several minutes.



```
>> [ incNos, incTimes ] = int_defects_plastic_breakthrough_parametric( 1000, 1e-6, 'IntCrackJob1', 1:15 )

Determining plastic breakthrough increment for model #1
This is 1 out of 15 in the sequence to scan.
Reading and extracting information from input file.
Warning: .inp contains more than one set of element definitions. There is probably more than one element type in
use. outputStruct.elems will be given cell array format.
> In read_inp (line 61)
  In int_defects_plastic_breakthrough (line 56)
  In int_defects_plastic_breakthrough_parametric (line 111)
Warning: Duplicate data points have been detected.
Some data points do not map to vertices in the triangulation.
To avoid this behavior, call UNIQUE on the data points prior to calling DELAUNAY.
> In delaunayn (line 58)
  In int_defects_plastic_breakthrough (line 67)
  In int_defects_plastic_breakthrough_parametric (line 111)
... finished extracting information from input file.
Warning: J-integral output not read successfully from .dat file. Returning empty cell array.
> In read_dat_contour_integral (line 302)
  In int_defects_plastic_breakthrough (line 108)
  In int_defects_plastic_breakthrough_parametric (line 111)
Checking for plastic breakthrough in increment no. 46
Warning: Unable to extract J integrals and T stresses. The .dat file may be in the wrong format.
> In read_dat (line 805)
  In int_defects_plastic_breakthrough (line 124)
  In int_defects_plastic_breakthrough_parametric (line 111)
Warning: Unable to extract stress intensity factors. The .dat file may be in the wrong format.
> In read_dat (line 818)
  In int_defects_plastic_breakthrough (line 124)
  In int_defects_plastic_breakthrough_parametric (line 111)
Checking for plastic breakthrough in increment no. 69
Warning: Unable to extract J integrals and T stresses. The .dat file may be in the wrong format.
> In read_dat (line 805)
  In int_defects_plastic_breakthrough (line 124)
  In int_defects_plastic_breakthrough_parametric (line 111)
Warning: Unable to extract stress intensity factors. The .dat file may be in the wrong format.
> In read_dat (line 818)
  In int_defects_plastic_breakthrough (line 124)
  In int_defects_plastic_breakthrough_parametric (line 111)
Checking for plastic breakthrough in increment no. 81
Warning: Unable to extract J integrals and T stresses. The .dat file may be in the wrong format.
> In read_dat (line 805)
  In int_defects_plastic_breakthrough (line 124)
  In int_defects_plastic_breakthrough_parametric (line 111)
Warning: Unable to extract stress intensity factors. The .dat file may be in the wrong format.
```

Note: In the above image, an arbitrarily-small value of equivalent plastic strain (1e-6) has been used as the strain tolerance for `int_defects_plastic_breakthrough_parametric`.

5. Plot LLLs against aspect ratio for different crack depths:



```
     90
     67
     72
     76
     80
     86
     44
     50
     54
     61
     75

>> incTimes

incTimes =

   336
   344
   348
   352
   357
   268
   288
   304
   320
   344
   176
   200
   216
   244
   300

>> load('workspace_dump.mat')
>> for k1 = 1:15
aA1Overb(k1) = paramLogStruct(k1).naturalParams.aA1Overb;
aA1OveraA2(k1) = paramLogStruct(k1).naturalParams.aA1OveraA2;
end
>> figure
>> plot(aA1OveraA2(1:5),incTimes(1:5),'bo-')
>> hold on
>> plot(aA1OveraA2(6:10),incTimes(6:10),'ro-')
>> plot(aA1OveraA2(11:end),incTimes(11:end),'go-')
>> xlabel('Flaw aspect ratio (a/c)')
>> ylabel('Local limit stress (MPa)')
>> legend('a/b = 0.25','a/b = 0.5','a/b = 0.75')
>>
```

Name	Value
k8	1
k9	1
oldInpPresent	false
naturalParamStruct	1x1 struct
modelParamStruct	1x1 struct
modelParamsValidityComment	'Okay'
inpStatus	true
inpWriteExitMessage	[ ]
inpWriteWarnings	1x1 cell
modelCreateTotalWalltime	637.9541
paramLogStruct	15x1 struct
modelRunTotalWalltime	2.6087e+04

Name	Value
writeInpTimeout	120
varargin	1x4 cell
PCflag	0
paramRangeStruct	1x1 struct
paramRangeFilename	'paramRangeStruct_B...
paramLogStruct	15x1 struct
oldInpPresent	0
naturalParamStruct	1x1 struct
modelRunTotalWalltime	2.6087e+04
modelParamsValidityComment	'Okay'
modelParamStruct	1x1 struct
modelCreateTotalWalltime	637.9541
manualInputFlag	0
kTotal	15
	1x15 double
	1
	1
	1
	1
	1
	1
	1
	1
	1
	1
	5
	15
	15
	1x1 cell
	[]
	1
	15x1 double
	15x1 double
	1x1 struct
	1
	'abq6121'
	1x15 double
	1x15 double

A.5 Determining the J-integral at plastic breakthrough

Occasionally, it is useful to know the J-integral which occurs at a flaw at the point when a plastic ligament develops between it and the back face of the wall. This can be used to compare J-integral results for different materials, or as part of a strain-based analysis. For a strain-hardening material, the presence of plasticity at any point in the model must be judged using some finite value of strain (eg. 5% von Mises equivalent strain).

In this example, we will determine the J-integral for an embedded elliptical flaw in a plate of elastic-plastic material subjected to a combined tension and remote bending load. The tension and bending loads are increased proportionally to one-another, and the increment of loading at which a ligament containing ≥5% plastic strain connects the flaw and the plate surface is determined. Then the J-integral results for the flaw at this loading increment are extracted. Only a single flaw geometry is considered.

The material stress-strain curve used follows a Ramberg-Osgood relationship:

$$\varepsilon = \frac{\sigma}{E} + \alpha \frac{\sigma}{E}\left(\frac{\sigma}{\sigma_Y}\right)^{n-1}$$

where ε is the true strain, σ is the (uniaxial) true stress, the yield stress σ_Y is taken as 360 MPa, the hardening modulus n is 12 and the yield offset parameter α is 1.6667. The material is modelled using incremental plasticity theory, with a von Mises yield locus and an isotropic hardening characteristic.

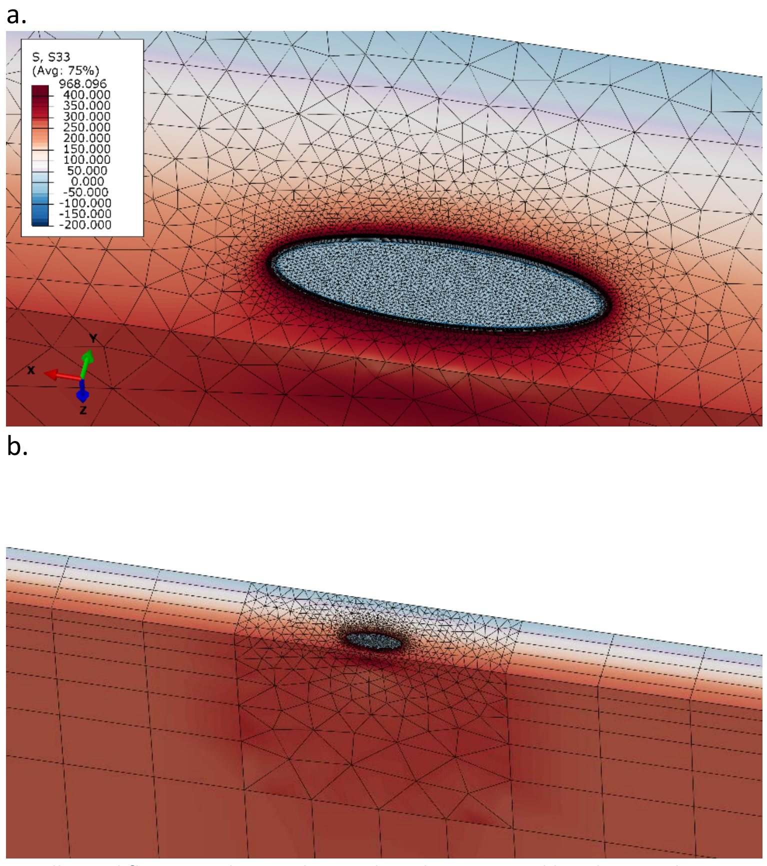


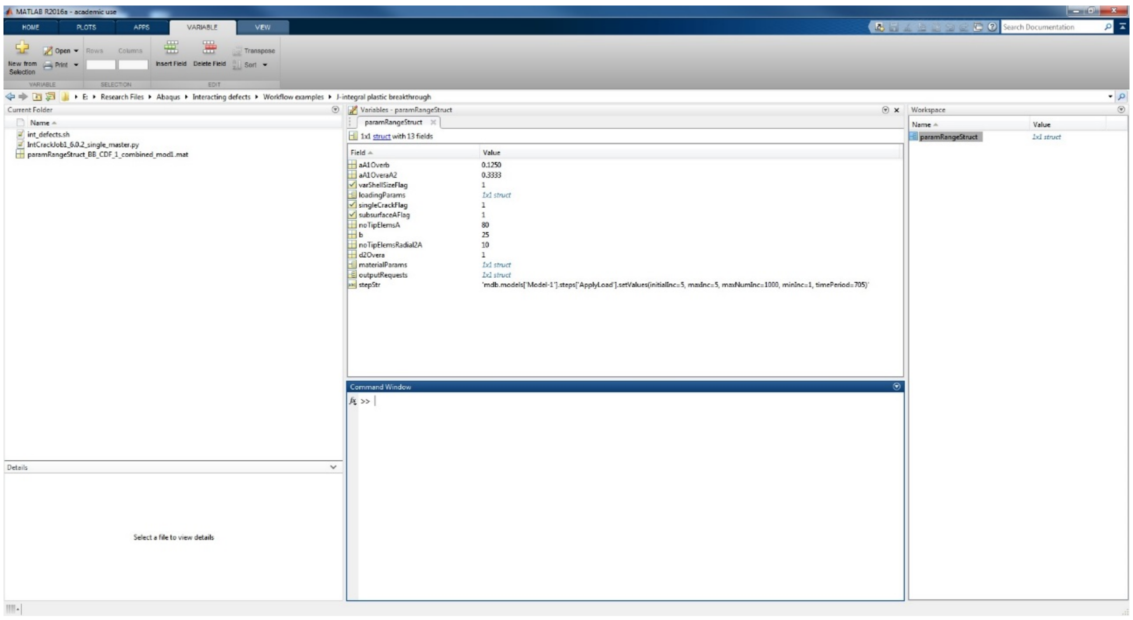
Figure 4: Embedded semi-elliptical flaw in a plate under combined tension and bending. In these images, the loading applied is an equal combination of tension and bending that would, in a perfect linear elastic material, give a linear through-wall distribution of stress: from σ_{33} = 0 MPa on the rear face to 300 MPa on the front face.

The plate is infinitely wide, with a wall thickness is 25 mm. The flaw is an embedded semi-elliptical crack as shown in Figure 3c (Crack B). It has a half-depth in the through-thickness dimension of

3.125 mm (a/b = 0.125), a half-width if 9.376 mm (i.e. an aspect ratio of a/c = 0.3333), and depth from the plate surface of 3.125 mm (i.e. a normalised depth of $d_2/a = 1$).

The loading is applied remotely and is an even combination of tension and bending. The bending moment per unit of plate width is 36.72 kNmm/mm, which would cause remote ligament stresses of ±352.5 MPa at the plate surfaces. The tensile load per unit of plate width is 8.81 kN/mm, which would cause a uniform tensile stress of 352.5 MPa throughout the thickness of the plate. The superposition of these loads causes a through-thickness stress distribution which rises from 0 MPa on the back face of the plate to 705 MPa on the front face. The load and moment are increased in proportion to one-another and ramped linearly over the model time step.

1. Copy the master python file “IntCrackJob1_6.0.4_single_master.py” into the working directory.
2. Using MATLAB, construct the data structure paramRangeStruct defining the models. The material is defined as elastic-plastic using paramRangeStruct.materialParams.plasticTable. Von Mises equivalent plastic strain field output is requested for the whole model using: paramRangeStruct.outputRequests.custom = ‘** OUTPUT REQUESTS
 *El Print, Elset = PlatePartitioned-1.Set-1, Position = Centroidal, Frequency=1
 PEEQ’. No contour integral output was requested: paramRangeStruct.outputRequests.contourTypeA = ‘none’.



E: ▸ Research Files ▸ Abaqus ▸ Interacting defects ▸ Workflow examples ▸ J-integral plastic breakthrough

Current Folder:

```
int_defects.sh
IntCrackJob1_6.0.2_single_master.py
paramRangeStruct_BB_CDF_1_combined_mod1.mat
```

Variables - paramRangeStruct

1x1 struct with 13 fields

Field	Value
aA1Overb	0.1250
aA1OveraA2	0.3333
varShellSizeFlag	1
loadingParams	1x1 struct
singleCrackFlag	1
subsurfaceAFlag	1
noTipElemsA	80
b	25
noTipElemsRadial2A	10
d2Overa	1
materialParams	1x1 struct
outputRequests	1x1 struct
stepStr	'mdb.models['Model-1'].steps['ApplyLoad'].setValues(initialInc=5, maxInc=5, maxNumInc=1000, minInc=1, timePeriod=705)'

Workspace: paramRangeStruct — 1x1 struct

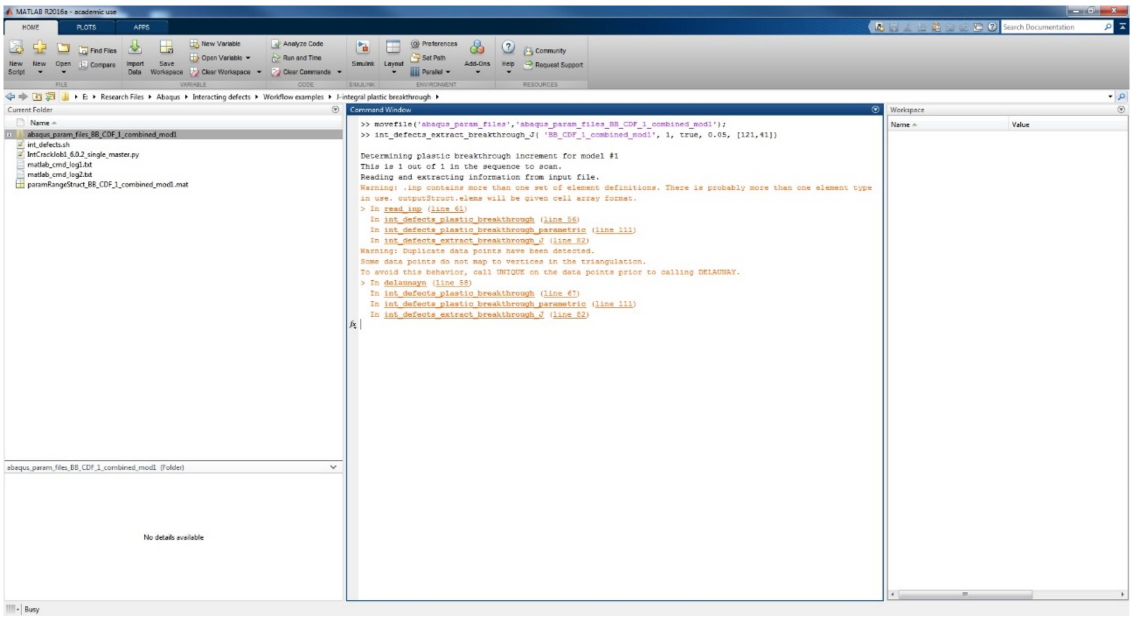
3. Create and run the models. For this example, model creation and execution was performed on a remote Linux server using the following commands:

```
int_defects_write_inp_parametric('paramRangeStruct_BB_CDF_1_combined
_mod1.mat','IntCrackJob1_6.0.4_single_master.py','abaqus',600);

load('abaqus_param_files/workspace_dump.mat');

int_defects_run_parametric_parallel(paramLogStruct,filenamesStruct,'
abaqus',44000,1,8,false,false);
```

4. Rename the “abaqus_param_files” directory to add an identifying element to the name, eg:

```
movefile('abaqus_param_files','abaqus_param_files_BB_CDF_1_combin
ed_mod1');
```

5. Use `int_defects_extract_breakthrough_J` to determine the J-integral result at the time step at which plastic breakthrough (at 5% strain) is first detected. It may take several minutes to determine the plastic breakthrough time step and extract the J-integral data. The results are written into the file "J_results_*my_runName*.mat".


MATLAB R2016a - academic use
HOME
PLOTS
APPS
E: ▸ Research Files ▸ Abaqus ▸ Interacting defects ▸ Workflow examples ▸ J-integral plastic breakthrough ▸
Current Folder
abaqus_param_files_BB_CDF_1_combined_mod1
int_defects.sh
IntCrackJob1_6.0.2_single_master.py
matlab_cmd_log1.txt
matlab_cmd_log2.txt
paramRangeStruct_BB_CDF_1_combined_mod1.mat
abaqus_param_files_BB_CDF_1_combined_mod1 (Folder)
No details available
Command Window
Workspace
Name
Value
Busy

```
>> movefile('abaqus_param_files','abaqus_param_files_BB_CDF_1_combined_mod1');
>> int_defects_extract_breakthrough_J( 'BB_CDF_1_combined_mod1', 1, true, 0.05, [121,41])

Determining plastic breakthrough increment for model #1
This is 1 out of 1 in the sequence to scan.
Reading and extracting information from input file.
Warning: .inp contains more than one set of element definitions. There is probably more than one element type
in use. outputStruct.elems will be given cell array format.
> In read_inp (line 61)
  In int_defects_plastic_breakthrough (line 56)
  In int_defects_plastic_breakthrough_parametric (line 111)
  In int_defects_extract_breakthrough_J (line 82)
Warning: Duplicate data points have been detected.
Some data points do not map to vertices in the triangulation.
To avoid this behavior, call UNIQUE on the data points prior to calling DELAUNAY.
> In delaunayn (line 58)
  In int_defects_plastic_breakthrough (line 67)
  In int_defects_plastic_breakthrough_parametric (line 111)
  In int_defects_extract_breakthrough_J (line 82)
fx |
```

6. Inspecting the contents of J_results_*my_runName*.mat, we can find the following information:
   - 5% plastic breakthrough occurs at a model time of 597.5 (outputArrayNear(1) = 597.5). Note that in this model, the model time is equal to the maximum remote ligament stress in MPa. So a max. remote ligament stress of 597.5 MPa cause 5% plastic breakthrough.
   - At this level of loading, the J-integral is 136.8 N/mm at the point on the crack tip line closest to the front face of the plate (outputArrayNear(2) = 136.8).
   - The J-integral is 91.2 N/mm at the point on the crack tip line closest to the front face of the plate (outputArrayFar(2) = 139.0).
   - The maximum J-integral encountered anywhere on the crack tip line at this level of loading is 139.0 N/mm (outputArrayMax(2) = 139.0).

A.6 Workflow automation using a bash script

When running `int_defects` on a Unix-based system, you can use the shell script `int_defects.sh` to automate the generation and solution of a model set. nB. This script does not automate extraction of the results.

1. Copy the required master python file ("IntCrackJob1_6.0.4_single_master.py" or "IntCrackJob1_6.0.4_master.py") into the working directory.
2. Edit the shell script `int_defects.sh` so that it contains the correct input information on the indicated lines.
3. Run:

```
chmod +x int_defects.sh
bash int_defects.sh
```

An echo of the commands that are being executed in MATLAB is written in matlab_cmd_log1.txt and matlab_cmd_log2.txt.

Appendix B: Hints and tips

B.1 Parameter ranges

- When defining a parameter set (i.e. when constructing paramRangeStruct) for the first time, it is easy to accidentally omit required parameters. In most cases, `int_defects_write_input_parametric` should provide informative errors about what parameters are missing or have unacceptable values. It may be useful to copy a similar parameter set from one of the examples given in Appendix A and then adapt it to the task at hand.

- The range of crack aspect ratios that can be meshed by Abaqus/CAE depends on factors such as the number of mesh seeds and the size of the inner and other crack tip mesh zones, both of which are specified in paramRangeStruct. As a rule of thumb, cracks with an aspect ratio in the range $0.25 \leq a/c \leq 4$ will usually mesh without problems, although much smaller/greater aspect ratios are possible in some cases.

- When defining a parameter set (i.e. when constructing paramRangeStruct) it is not necessary to ensure that all possible combinations of the defined parameters would give a geometrically-valid model. For example, if one combination of parameters defined flaws which overlapped, this model would not be attempted: `int_defects_write_inp_parametric` will simply move on to the next combination. A log of the invalid parameter combinations is made in paramLogStruct, and a summary of invalid combinations will be given in log1.txt.

- For models of interacting flaws, the flaw spacing can be specified in a large number of different ways in paramRangeStruct. It is worth thinking carefully about what parameter the crack spacing should depend on, and the range of parameter values, since this will depend on the objective of the study.

- When setting up `int_defects` to run a large set of models with different parameters, it is often useful to first run a smaller set containing all the extreme parameters. If there are any problems relating to the input parameters, this will make them apparent much more quickly than if the large model set was run straight away.

B.2 Results

- When running sets of models, you can use the input argument abaqusCleanupFlag=false in `int_defects_run_parametric_parallel` to prevent deletion of each model's .odb file. The .odb can then be inspected using Abaqus/CAE, which is useful for troubleshooting or to allow export of images.

- When analysing the results from a set of models, it is often useful to view the parameters of each individual model in a set. Use paramLogStruct in abaqus_param_files\workspace_dump.mat to review the parameters and execution status of a model after the event.

- `int_defects_read_parametric_parallel` (and `int_defects_read_user`, which calls it) returns contour integral data for all available contours reported by Abaqus. For example, outputArray{1,3}.k{2,2}(end,:) contains the SIFs for third model in the series, second crack in the model, outermost available contour. outputArray{1,3}.k{2,2}(end,1) would give the results for the innermost available contour. Use of the outermost available contour is recommended. The number of contours requested can be set in: paramRangeStruct (outputRequests.noContoursA and outputRequests.noContoursB).

B.3 Other

- Always be aware of the resources that are required for running models using Abaqus/Standard – including both no. of CPUs and the amount of available RAM relative the size of the models being run. Set timeouts for `int_defects_write_inp_parametric` and `int_defects_run_parametric_parallel`.
- `int_defects` is designed to allow robust model generation and solution. If one model in a set fails to mesh or solve, or even if many models encounter problems, `int_defects` can still proceed with processing the rest of the model set. This means that with a dedicated machine and with enough time, you can process very large sets of models (thousands). The largest number of models in a single set analysed with `int_defects` so far is 7350 [9].

Appendix C: Known bugs and wish list

C.1 Known bugs

In this context, 'bugs' are defined as functionality which should work but doesn't, rather than limitations of the software or 'missing' features.

- No documented bugs.

C.2 Wish list

- Add explicit definition of the crack extension direction, i.e. using q-vectors rather than a crack-plane normal.
- Add functionality to allow the study of embedded-embedded defect interaction.
- Currently `int_defects_write_inp_parametric` creates models for all possible combinations of the parameters given in paramRangeStruct. In some cases it would be useful to be able to define a subset of all combinations of parameters to analyse, which could reduce the time spent building and solving models for "less interesting" regions of the parameter space.
- Add interoperability with the Warp3D FE solver.
- Improve readability of the scripting interface code Python code that is generated by `int_defects` when creating models. For example, use a consistent and descriptive set of object names.
- Add functionality for using existing Abaqus thermal model solutions as the loading case. nB. This was already done in [15], but has not made its way into the main branch of `int_defects`.
- Add functionality for creep analysis, including model generation options and extraction and post-processing of C* contour integral results.

Appendix D: Development history and acknowledgements

D.1 Development history

I developed the initial versions of this software in late 2015 as part of an EPSRC-funded postdoctoral research fellowship on nuclear structural integrity (EPSRC grant no. EP/M019446/1). It was designed for validating flaw interaction rules used in the R6 structural integrity assessment procedure and was initially a code library targeted at linear-elastic analysis only. Checking the adequacy of these interaction rules required a large number of FE models and so an automatic way to deal with model generation and submission handling was the natural response.

The software was developed intermittently from 2015-2018: the addition of new features was done ad hoc to support specific studies on flaw interaction, some of which are listed in Section 9.2. In late 2018 I decided to release `int_defects` as a MATLAB toolbox and began work on documenting its features and rationalising its code structure.

`int_defects` is written in MATLAB code and packaged as a toolbox. This has the advantages of enabling easy analysis and plotting of results, as well as easy distribution. MATLAB is also the high-level language that I am most familiar with. On the other hand, `int_defects` also uses the Abaqus/CAE scripting interface which is Python-based. Therefore, it would be more elegant for the model generation parts of `int_defects` to be written in Python, which unlike MATLAB is also free and open-source. Since the toolbox works well in its current form, I have no plans to recode any parts in Python.

Harry Coules, 06/2019

D.2 Acknowledgements

The original idea for this toolbox grew out of initial discussions on flaw interaction with Peter Budden, Kisaburo Azuma and Bostjan Bezensek. Part of the testing of int_defects was performed by Molly Probert. Development of this software was enabled by the following funding from the UK Engineering and Physical Sciences Research Council (EPSRC):

- A postdoctoral fellowship in nuclear structural integrity (“Advanced structural analysis for the UK nuclear renaissance”, EP/M019446/1).
- An Impact Acceleration Account early-career kickstarter award (“Exploiting new results in defect analysis to improve structural integrity assessment”).