Abaqus2Matlab: a suitable tool for finite element post-processing

George Papazafeiropoulos[a], Miguel Muñiz-Calvente[b], Emilio Martínez-Pañeda[c,*]

[a]*Department of Structural Engineering, National Technical University of Athens. 15780 Zografou, Athens, Greece*
[b]*Department of Construction and Manufacturing Engineering, University of Oviedo, Gijón 33203, Spain*
[c]*Department of Mechanical Engineering, Solid Mechanics, Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark*

Abstract

A suitable piece of software is presented to connect Abaqus, a sophisticated finite element package, with Matlab, the most comprehensive program for mathematical analysis. This interface between these well-known codes not only benefits from the image processing and the integrated graph-plotting features of Matlab but also opens up new opportunities in results post-processing, statistical analysis and mathematical optimization, among many other possibilities. The software architecture and usage are appropriately described and two problems of particular engineering significance are addressed to demonstrate its capabilities. Firstly, the software is employed to assess cleavage fracture through a novel 3-parameter Weibull probabilistic framework. Then, its potential to create and train neural networks is used to identify damage parameters through a hybrid experimental-numerical scheme, and model crack propagation in structural materials by means of a cohesive zone approach. The source code, detailed documentation and a large number of tutorials can be freely downloaded from www.abaqus2matlab.com.

Keywords:
Abaqus2Matlab, Post-processing, Finite Element Method, Weibull stress model, Inverse analysis

---

*Corresponding author.
  Email address: mail@empaneda.com (Emilio Martínez-Pañeda)

1. Introduction

Partial Differential Equations (PDEs) govern the physics of most engineering systems. As analytical solutions are limited and generally restricted to idealized cases, the development of efficient and robust numerical methods marks a milestone in the solution of boundary value problems in structural mechanics, electromagnetism, heat transfer, mass diffusion and fluid dynamics, among many other disciplines. The Finite Element Method (FEM) has become the leading numerical technique for solving PDEs in the mechanical, civil, aeronautical and bioengineering industries. Among the wide range of packages available, Abaqus [1] is undoubtedly one of the most popular finite element tools for academics and practitioners.

However, practical applications often require considering non-linear conditions, where uncertainties hinder high fidelity numerical predictions. In such circumstances, the use of advanced analysis methodologies - such as inverse approaches, statistical tools or hybrid experimental-numerical techniques - has proven to compensate the lack of information, yielding results that are otherwise unobtainable. Matlab [2], a multi-paradigm computing environment, is generally considered to be the most powerful software in this regard due to its advanced capabilities in statistics, machine learning, neural networks, curve fitting, model-based calibration and optimization. Yet, a connection between the two most used packages in, respectively, finite element modeling and mathematical analysis, is still lacking. To fill this gap, a novel software tool is here proposed: *Abaqus2Matlab*, which allows to run Abaqus directly from Matlab and to post-process the results, providing a link between the two well-known packages in a non-intrusive and versatile manner. The present proposal enjoys the benefits of Matlab's user friendly and centralized environment, as opposed other powerful tools like Python, which require add-on libraries.

Abaqus2Matlab is distributed as source code with the aim of facilitating research. Numerous codes have been made freely available through the years, positively impacting the computational mechanics community. For instance, Sigmund and co-workers presented an efficient topology optimization implementation [3, 4], Bordas and collaborators [5–7] described an object-oriented

programming library for the extended finite element method (X-FEM) and meshless methods, Giner *et al.* [8] implemented the X-FEM in Abaqus through a user subroutine, Parks and Paulino [9] described the numerical implementation of the PPR potential-based cohesive zone model, Nguyen [10] proposed an open source program to generate zero-thickness cohesive elements and Martínez-Pañeda and Gallego [11] provided a user subroutine to effectively define the material property variation of functionally graded materials in Abaqus. Other open-source software that has recently contributed to scientific progress includes FReET [12], a code to conduct statistical, sensitivity and reliability assessment; FraMePID-3PB [13], a tool to identify fracture parameters in concrete through inverse analysis; NiHu [14], an open source C++ library for the boundary element method; ESFM [15], a general framework for meshless methods; NOSA-ITACA [16], a finite element code for masonry structures; PCLab [17], an object-oriented Monte Carlo/Finite Element software; and, μMECH [18], an open source C/C++ library of analytical solutions to classical micromechanical problems.

The present manuscript is organized as follows. The software framework and architecture are explained in the following section. Then, Section 3 provides usage instructions through simple examples. Section 4 shows the capabilities of the toolbox by addressing two relevant engineering applications; namely, probabilistic analysis of cleavage fracture and inverse identification of damage parameters through neural networks. Finally, the work is summarized in Section 5.

2. Abaqus2Matlab

The main internal characteristics of *Abaqus2Matlab* are described below. The structure of Abaqus results (*.fil) file is briefly described in the first place, as it is necessary to understand how the presented software stores Abaqus results. The reading procedure is then detailed and insight is given into the software architecture.

2.1. Creating and processing Abaqus' results (*.fil) file

The results (*.fil) file can be used to transfer Abaqus analysis results to other packages. The aforementioned file can be written in binary or ASCII

3

format, depending on the need for porting results between dissimilar operating systems. ASCII format is chosen in the present approach due to its versatility.

2.1.1. Generation of Abaqus results (*.fil) file

The Abaqus results file is obtained in ascii format by defining specific options in the input (*.inp) or restart (*.res) files. The results file generation procedure differs between Abaqus/Standard and Abaqus/Explicit, *FILE FORMAT, ASCII must be specified in the former and *FILE OUTPUT in the latter. The reader is referred to Abaqus documentation for more details.

2.1.2. Output

The following output types can be written to the results file: element, nodal, energy, modal, contact surface, element matrix, substructure matrix and cavity radiation factor. Nodes and elements are numbered globally in models that have been defined as an assembly of part instances. A map between user-defined numbers and internal numbers is printed to the data file (*.dat) if any output requested includes node and element numbers. Set and surface names that appear in the results file are given along with their corresponding assembly and part instance names, separated by underscores.

2.1.3. Record format

The results (*.fil) file is a sequential file that must be read up to the location of the desired data. All data items are converted into equivalent character strings and written in (logical) records. Each single line contains a series of 80 string characters, which may contain the full record or part of it. In the latter case, after completely filling the first line, the record string continues at subsequent lines. The beginning of each record is indicated by an asterisk (*) and the data items are arranged immediately behind each other within each record. Each record has the format shown in Table 1.

Table 1: Format of a record written in an Abaqus results file

Location	Length	Description
1	1	Record Length (L)
2	1	Record type key
3	(L-2)	Attributes

The location number denotes the position in the record where a series of consecutive data items are written. The number of data items in each series is denoted by the length number. The first data item is an integer denoting the number of data items in the record. The second one defines the record type key, an indicator denoting the type of data. And finally the attributes are contained in a series of L-2 data items, at the 3rd position of a record.

2.1.4. Data item format

Integer numbers are denoted by the character I, followed by a two digit integer which shows the number of the digits of the integer with the value of the integer following. On the other hand, floating point numbers begin with the character D, followed by the number in the format E22.15 or D22.15, depending on the precision. And character strings begin with the character A, followed by eight characters. If the length of a character string is less than 8, then the trailing positions are filled with blank spaces. If the length of a character string is larger than 8, then the character string is written in consecutive character strings, eight characters at a time.

2.2. Reading Abaqus results files with Abaqus2Matlab

A function named `Fil2str` is defined in Matlab to read the Abaqus results (*.fil) file by considering the data as a string and concatenating lines horizontally, as shown in listing 1.

Listing 1: Function Fil2str.m to read Abaqus results (*.fil) file

```matlab
1  function Rec = Fil2str(ResultsFileName)
2  % Open the results file for reading
3  fileID = fopen(ResultsFileName,'r');
4  % Read data as a string and assign them to a cell array
5  % Concatenate each line without specifying delimiter, white
6  % space or end of line characters
7  try
8  C = textscan (fileID, '%s', 'CollectOutput', '1', ...
9          'delimiter','','whitespace','','endofline','');
10 catch
11 C = textscan (fileID, '%s', 'CollectOutput', 1, ...
12          'delimiter','','whitespace','','endofline','');
13 end
14 % Close the results file
15 fclose(fileID);
16 % Assign A
17 A = C{1}{1};
18 % Remove newline characters
19 A1 = strrep(A,sprintf('\n'),'');
20 % Remove carriage return characters
21 Rec = strrep(A1,sprintf('\r'),'');
```

The function is programmed so as to allow compatibility between different MATLAB versions. The information from the results file is stored in a cell array C containing a single line string. That single line string subsequently enters an *ad hoc* function that depends on the results that the user wishes to post-process. Thus, more than 50 different functions are already available in *Abaqus2Matlab*, covering the vast majority of results types that can be obtained in Abaqus; new record functions can be easily generated from the existing template. An appropriate naming convention is adopted, where each function is defined by the word Rec followed by the record key of the particular type of results. Record keys for each specific set of results can be found in Abaqus documentation. For example, nodal coordinates (record key 1901) are obtained through function Rec1901.m, whose code is shown in listing 2.

The programming of such functions follows a similar structure. First, the record length is obtained for preallocation purposes, using Matlab's intrinsic function strfind to find the positions of the records in Rec. For each case, the first 8 characters in front of each position are stored in Rec2. After-

6

wards, the record length is identified and stored in the column vector `NW` by first converting from string to double format using Matlab's built-in function `str2num`. Subsequently, the elements of `ind` (i.e. position of the second data item of records giving node definition results) are scanned and for each element the number of digits of the node number is determined first, then the node number, and finally the nodal coordinates, by the insertion of a `for` loop within each record definition. Finally, the node numbers and the node coordinates are concatenated horizontally to form the output array. Hence, the `Rec1901` function takes as input a one-row string containing the ASCII code of the ABAQUS results (*.fil) file and provides as output a matrix with as many rows as nodes in the model and with the node number in the first column and the nodal coordinates in the subsequent columns.

Listing 2: Example of a specific record function

```matlab
1  function out = Rec1901(Rec)
2  ind = strfind(Rec,'I 41901'); % record key for node output
3  if isempty(ind)
4      out=[];return;
5  end
6  nextpos=numel('I 41901')+1;
7  % Initialize
8  NodeNum=zeros(numel(ind),1);
9  NW=zeros(numel(ind),1);
10 for i=1:numel(ind)
11     % find the record length (NW)
12     Rec2=Rec(ind(i)-7:ind(i));
13     indNW=strfind(Rec2,'*'); % record starts with *
14     % ensure record existence and appropriate type key ...
                location
15     if isempty(indNW) || indNW>3
16         ind(i)=NaN;continue;
17     end
18     % number of digits of the record length
19     ind1=indNW+2; ind2=indNW+3;
20     a1=str2num(Rec2(ind1:ind2));
21     % Record length (NW)
22     ind1=ind1+2; ind2=ind2+a1;
23     NW(i)=str2num(Rec2(ind1:ind2));
24 end
25 NodeCoords=zeros(numel(ind),max(NW)-4);
26 for i=1:numel(ind)
27     % number of digits of the node number
28     ind1=ind(i)+nextpos; ind2=ind(i)+nextpos+1;
29     a1=str2num(Rec(ind1:ind2));
30     % Node number
31     ind1=ind1+2; ind2=ind2+a1;
32     NodeNum(i)=str2num(Rec(ind1:ind2));
33     % Node coordinates
34     for j=1:NW(i)-4
35         % node coordinate
36         ind1=ind2+2;
37         ind2=ind2+23;
38         NodeCoords(i,j)=str2num(Rec(ind1:ind2));
39     end
40 end
41 % Assemply of matrices for output
42 out=[NodeNum NodeCoords];
43 end
```

3. Usage instructions

A brief description of the source code assembly is given first to ease the understanding of the software and allow for personalized developments. The most relevant operations are then detailed and finally a simple optimization example is described to show the use of *Abaqus2Matlab*. Comprehensive documentation can be found in www.abaqus2matlab.com.

3.1. Organization of the source code

Source code files are organized through the following folders:

- OutputAnalysis folder, which contains the functions required to post-process analysis type results (e.g. node definitions, element connectivity, eigenfrequencies and eigenvalues, etc.).

- OutputNodes folder, which contains the functions required to post-process nodal type results (e.g. node displacements, concentrated forces, nodal temperatures, etc.).

- OutputElements folder, which contains the functions required to post-process element type results, i.e. results at integration points or centroids (e.g. stresses, strains, section forces and moments, etc.).

- Verification folder, which contains numerous Matlab scripts to verify the `Rec` functions corresponding to each result type.

- AbaqusInputFiles folder, which contains the input files that are run by Abaqus to verify each of the results functions.

- Html and Help folders, which contain all the documentation files of *Abaqus2Matlab*.

3.2. Main usage instructions

Firstly, the `Documentation.m` file must be compiled in order to add all *Abaqus2Matlab* specific functions to Matlab's libraries. Emphasis has been placed in the development of user-friendly and intuitive software. As a consequence, only three steps are required; namely, (i) generate Abaqus' input file, (ii) run the finite element analysis and (iii) extract the results required. Thus, one should first indicate in Abaqus' input file the specific data that should be stored in the results file (*.fil); e.g. nodal displacements,

Listing 3: Commands required to store the specific information in the *.fil file

```
1  *FILE FORMAT, ASCII
2  *NODE FILE
3  U
```

the code described in Listing 3 should be included at the end the step defini-
tion (i.e., at the end of the document if it is a one-step analysis), just before
the command *END STEP.

The remaining steps are performed entirely inside *Abaqus2Matlab* frame-
work within Matlab. A template script is shown in listing 4. First, the
finite element job is run for a specific Abaqus input file. Both the script file
and Abaqus' input file must be stored in the same folder (i.e., the working
directory). The information in the results file (*.fil) is then read and clas-
sified according to the desired output data through `Fil2str` and - for the
displacement field - `Rec101`, respectively.

Listing 4: Example of a specific *Abaqus2Matlab* script

```
1  %% Template script Abaqus2Matlab
2  % Run the corresponding input file 101.inp with Abaqus
3  Inp_file='101';
4  system(['abaqus job=' Inp_file]);
5  % Pause Matlab execution to create the lck file
6  pause(10)
7  % If the lck file exists then halt Matlab execution
8  while exist([Inp_file '.lck'],'file')==2
9  pause(0.1)
10 end
11 %% Postprocess Abaqus results file with Matlab
12 % Assign all lines of the fil file in an one-row string
13 Rec = Fil2str( [Inp_file '.fil']);
14 % Obtain the desired output data
15 Displacements= Rec101(Rec);
```

Template scripts for the most often used types of Abaqus results have
been developed and can be found in the Verification folder.

3.3. Simple example: truss optimization problem

A simple optimization example is shown to display *Abaqus2Matlab* functi-
oning. The weight of the 2-bar plane truss shown in Fig. 1 will be optimized

10

by minimizing the members' cross-sectional area. The truss is characterized by the following quantities: Young's modulus $E = 68.948$ GPa, density $\rho = 2767.99$ kg/m^3, bar length $L = 9.144$ m and applied concentrated forces $P = 444.974$ kN. Material costs can be lowered by reducing the weight, which implies - for a constant density and bar length - minimizing the sum of the cross-sectional areas, the design variables under consideration. Constraints are imposed on the displacements, where their maximum value should be limited to $d_{max} = 0.0508$ m in both directions, and stresses, where their magnitudes should be lower than $\sigma_{max} = 172.369$ MPa in both tension and compression (absolute value). The design variables are the cross section area of each member in the interval $[0.00365, 0.02258]$ m^2, with the lower bound being a consequence of the upper limit imposed on the axial stress.
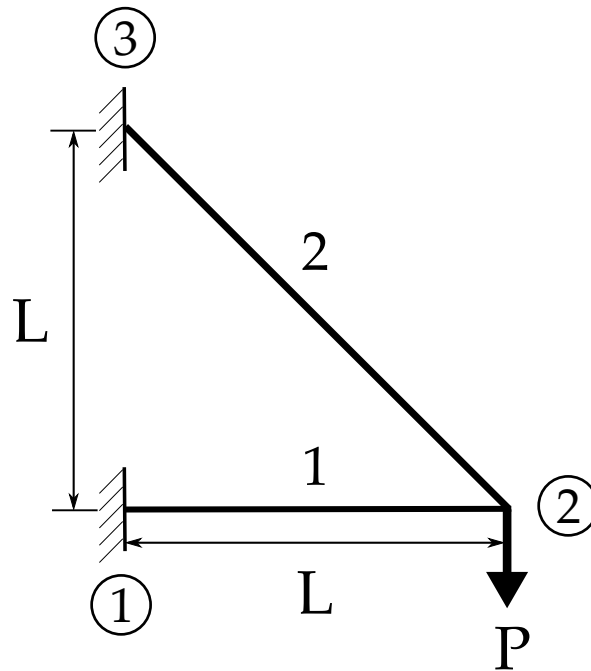
Figure 1: Optimization example: a 2-bar plane truss, including element and node (circles) numbering.

The main script employed to solve the problem is shown in listing 5. The number of elements is first specified and an initial guess for the 2 cross-sectional areas assigned. Before calling the main optimization function, design variable limits are defined and tolerances provided; the latter include an

11

upper bound on the variation of the objective function (i.e., truss weight) during a step and an upper bound on the magnitude of the constraint functions.

Listing 5: Main code for the optimization example

```
1  % Specify the number of elements of the truss
2  NumElements=2;
3  % Make a starting guess for the solution.
4  x0 = [0.0037; 0.0049];
5  % Set the lower and upper limits
6  AreaMin=0.003650822800775; % P*sqrt(2)/maxstress
7  AreaMax=0.0225806;
8  lb=AreaMin*ones(1,NumElements);
9  ub=AreaMax*ones(1,NumElements);
10 % Set FunctionTolerance and StepTolerance
11 options=optimset('fmincon');
12 options.Display='iter-detailed';
13 options.TolFun=1e-3;
14 options.TolCon=1e-3;
15 % Perform constrained optimization of the truss
16 [X,fval,exitflag,output,lambda]=fmincon(@TrussObjfun,x0,...
17     [],[],[],[],lb,ub,'TrussConfun',options)
```

TrussObjfun (listing 6) is a simple function that provides as output the weight of the truss for given values of the design variables. TrussConfun.m, shown in listing 7, is employed to construct the Abaqus input file and subsequently perform the calculations. The input file is created through TrussInpFileConstr.m on every constraint evaluation, changing the data lines corresponding to the cross section area. Next, postprocessing of the aforementioned results takes place, which concludes in the formation of the inequality and equality vectors required as an output of the constraint function in Matlab (c and ceq, respectively). All the required information is then available for Matlab's function fmincon to perform the optimization analysis.

Listing 6: TrussObjfun function

```
1  function f = TrussObjfun(x)
2  u=9.144; % Horizontal length
3  f = 9.81*2767.990471*x'*u*[1;sqrt(2)]; % total weight
4  end
```

Results show that design variables at the local minimum equal [0.00365, 0.00482] m^2, while the minimum truss weight renders 2.5987 kN. For the particular example under consideration, this has been achieved after 6 iterations and 21 objective function evaluations. Optimization methodologies, like the one outlined here, can be used at the concept stage of the design process to achieve a proposal that best fits performance and manufacturability requirements. Such tools have proven to substantially reduce design development costs by avoiding expensive and time consuming design iterations.

Listing 7: Constraint function TrussConfun.m

```matlab
function [c,ceq] = TrussConfun(x)
% Set the displacement limits of the 2-bar truss
Dmaxhor=0.0508;Dmaxver=0.0508;
% Construct the Abaqus input file TrussABAQUS.inp
TrussInpFileConstr(x)
% Run the input file TrussABAQUS.inp with Abaqus
!abaqus job=TrussABAQUS
% Pause Matlab execution to create the TrussABAQUS.lck file
pause(10)
while exist('TrussABAQUS.lck','file')==2
    pause(0.1)
end
% Assign all lines of the TrussABAQUS.fil file in an ...
    one-row string
Rec = Fil2str('TrussABAQUS.fil');
% Obtain the nodal displacements
out2 = Rec101(Rec);
NodalDisplacements=out2(:,2:3);
% Delete the files of last Abaqus run to avoid rewriting them
delete('TrussABAQUS.fil');delete('TrussABAQUS.prt');
delete('TrussABAQUS.com');delete('TrussABAQUS.sim');
% Calculate the maximum nodal displacements
maxNodDisplX1=max(abs(NodalDisplacements(:,1)));
maxNodDisplY1=max(abs(NodalDisplacements(:,2)));
% Assemble the constraints
c = [maxNodDisplY1-Dmaxver;
    maxNodDisplX1-Dmaxhor];
ceq = [];
end
```

Results can be obtained in a few minutes and all the necessary files can

be freely downloaded from *Abaqus2Matlab* website.

4. Applications

The range of applications of *Abaqus2Matlab* is enormous, as it provides a non-intrusive connection between a sophisticated finite element package and the most comprehensive mathematical analysis tool. For demonstration purposes, two problems of particular interest from the scientific and engineering perspective will be addressed here. On the one hand, the toolbox proposed is used to estimate cleavage fracture in metals, where a probabilistic approach is needed due to the statistical nature of the micromechanisms involved. On the other hand, an advanced inverse-optimization methodology is employed to obtain the parameters governing the traction-separation law that describes deformation and fracture.

4.1. Cleavage fracture

Cleavage fracture originates from microcracks that nucleate from defects (carbides, cracks arrested at grain boundaries, etc.). The location of these defects is statistical by nature and hence modeling efforts rely on probabilistic analysis. Since the seminal work by Beremin [19], cleavage fracture toughness estimations are based on Weibull statistics and the weakest link model, where the probability of failure equals the probability of sampling (at least) one critical fracture-triggering particle. Grounded on this approach, we propose a novel probabilistic framework that takes advantage of the advanced statistical tools of MATLAB to estimate all Weibull-related parameters without any *a priori* assumptions.

For a given Weibull stress σ_w and a threshold stress for crack growth σ_{th}, the cumulative probability of failure P_f, in terms of the modulus m and scaling parameter σ_u is given by,

$$P_f = 1 - \exp\left[-\left(\frac{\sigma_w - \sigma_{th}}{\sigma_u}\right)^m\right] \tag{1}$$

where the Weibull stress can be defined as,

$$\sigma_w = \sigma_{th} + \left[\sum_{i=1}^{n_e}\left(\sigma_1^i - \sigma_{th}\right)^m (V_i/V_0)\right]^{(1/m)} \tag{2}$$

14

Here V_0 is a reference volume, V_i is the volume of the i*th* material unit (finite element) in the fracture process zone experiencing a maximum principal stress σ_1^i and n_e is the number of finite elements/material units in the fracture process zone. The parameter σ_{th} is needed due to the fact that cracks do not propagate below a certain threshold energy value. However, the concurrent estimation of the threshold, modulus and shape parameters remains a complicated task; a common approach lies in assuming a value for σ_{th} and estimating m and σ_u from a set of experiments. Here, all three parameters (σ_{th}, m and σ_u) will be obtained by means of a novel iterative procedure involving least squares estimates of the cumulative distribution functions.

The capabilities of *Abaqus2Matlab* to model cleavage fracture will be benchmarked with an extensive experimental data set developed within the Euro toughness project [20]. As in the experiments, a quenched and tempered pressure vessel steel DIN 22NiMoCr37 steel will be investigated; only tests where significant ductile crack growth is not observed will be considered and the reference experimental data will be that obtained at -40°C with a compact tension specimen of size 1T. Comprehensive details of the material tensile properties, specimen size and failure loads are provided in the original experimental article [20] and will not be reproduced here for the sake of brevity.

The algorithm methodology is described in Fig. 2. First, the finite element results are computed by running the corresponding Abaqus job *inside* the proposed toolbox. A finite element mesh of approximately 2000 quadratic plane strain quadrilateral elements with reduced integration is employed, with the elements being progressively smaller as the crack tip is approached. The values of the volume element V_i and the maximum principal stress σ_1^i are read and stored for each finite element and load level of interest. The latter is characterized through the J-integral and the pin displacement, that are also read in the *Abaqus2Matlab* environment. The statistical analysis is then conducted. The probability of failure for each failure load reported experimentally is first computed through,

$$P_f = \frac{j - 0.3}{n_j + 0.4} \tag{3}$$

where n_j denotes the number of experiments and j the rank number. Afterwards, an iterative procedure is conducted to simultaneously estimate σ_{th}, m and σ_u. In each iteration the Weibull stress is computed from the values

of m and σ_{th} from the previous iteration and subsequently inserted in Eq. (1) to compute the values of σ_u, m and σ_{th} in the current iteration by fitting a univariate distribution through the least squares method. Convergence is achieved when the relative norm of the change in the solution is below an appropriate tolerance value. Therefore, taking advantage of Matlab capabilities, Weibull parameters are calculated by finding the distribution whose cumulative function best approximates the empirical cumulative distribution function of the experimental data.
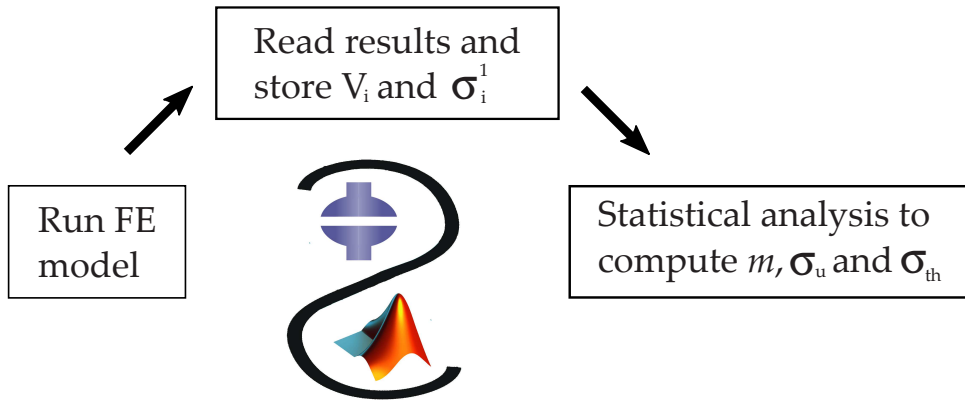
Figure 2: Schematic overview of the use of *Abaqus2Matlab* to estimate the probability of cleavage failure.

The results obtained for the particular case considered (Euro toughness data set, T1, -40°C) are displayed in Fig. 3. The figure shows the probability of failure versus the external load from the experimental study and the current statistical model. The calibrated Weibull stress parameters are also embedded in the figure. As it can be observed, a good agreement is attained between the failure probability estimated from Eq. (1) and the experimental results.
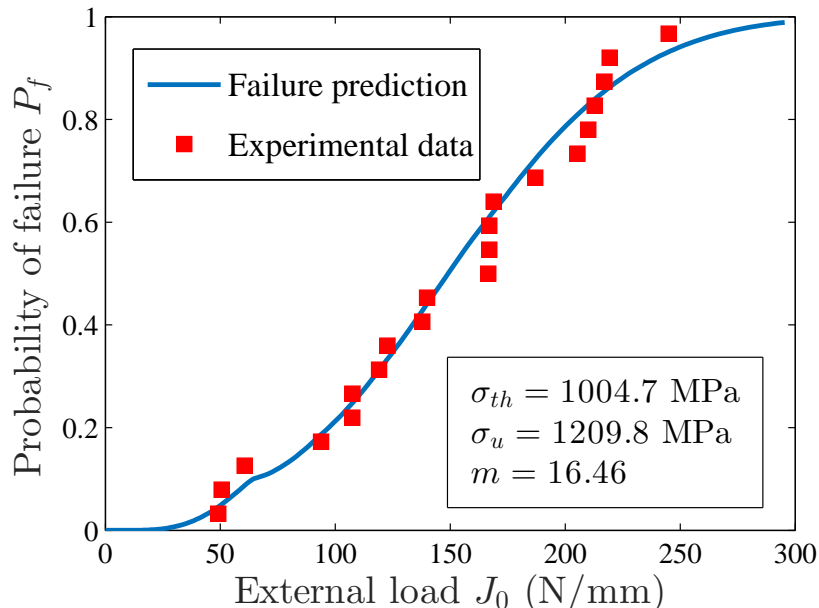
Figure 3: Failure probability as a function of the external load. The figure includes the experimental data for 22NiMoCr37 steel (EuroData project [20]) and the predictons from the present statistical model for the values of σ_{th}, σ_u and m displayed.

Results indicate that, for the particular case under consideration, a 50% probability of failure will be attained for an external load of approximately $J_0 = 150$ N/mm, while the 5% and 95% probability bonds are attained at $J_0 = 60$ N/mm and $J_0 = 250$ N/mm, respectively. Weibull-parameters estimation reveals that stresses lower than $\sigma_{th} = 1004.7$ MPa are innocuous and that a failure probability of 64% in a unit element is attained at a stress level of $\sigma_u + \sigma_{th} = 2214.5$ MPa.

More insight into the local failure probability can be gained by means of a *hazard map*. A hazard map highlights the areas being affected or vulnerable to a certain type of failure, providing visual information on the failure probability at each particular unit element [21]. Thus, the local probability of failure (i.e., P_f for a local σ_w) is shown in Fig. 4 in logarithmic scale. The mesh can be easily constructed by reading the nodal coordinates and the element connectivity through *Abaqus2Matlab*.
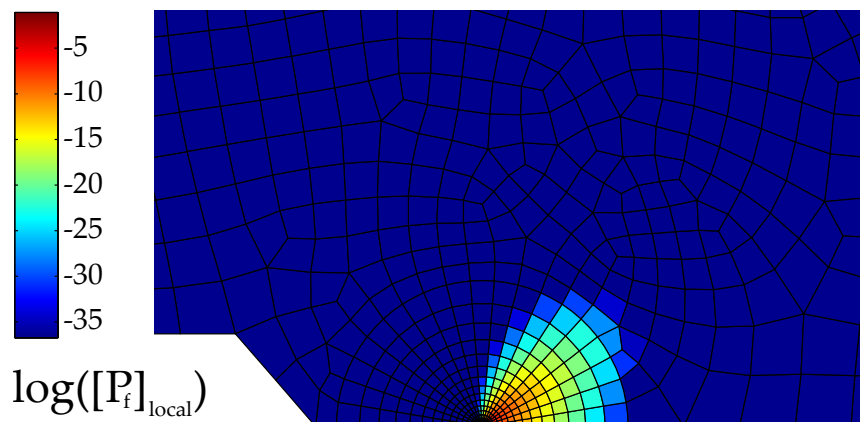
Figure 4: Hazard map. The legend shows the local probability of failure.

Statistical tools are indispensable to assess cleavage fracture as experimental data tends to be widely scattered; two identical specimens of the same material may have very different toughness values due to the random location of the fracture-triggering particle. *Abaqus2Matlab* enables the usage of Matlab's in-built statistical capabilities to estimate all Weibull parameters without any prior assumptions. This novel iterative framework allows for more precise estimations of failure probabilities, a crucial aspect in risk quantification and operational decision making in engineering applications.

4.2. Cohesive zone modeling

Classic fracture mechanics is an indispensable design tool that provides the basis for structural integrity assessment in engineering standards. The need to design components that exploit material performance to its maximum has however shifted scientific research from stationary cracks to crack propagation and damage. This is particularly true in ductile metals or composites, where a stable crack propagation stage precedes catastrophic failure. Among the many damage mechanics tools available, cohesive zone models are particularly attractive to characterize the reserve strength of the system once cracking has occurred, and to design accordingly [22]. The pivotal ingredient of cohesive zone modeling is the traction-separation law that governs material degradation and separation. As depicted in Fig. 5, for a given shape of the traction-separation curve, the cohesive response can be fully characterized through two parameters, the cohesive energy Γ_c and the critical cohesive strength T_c. Thus, for the bi-linear law of Fig. 5, the cohesive energy can be

18

expressed as a function of the critical separation δ_c and the critical cohesive strength T_c,

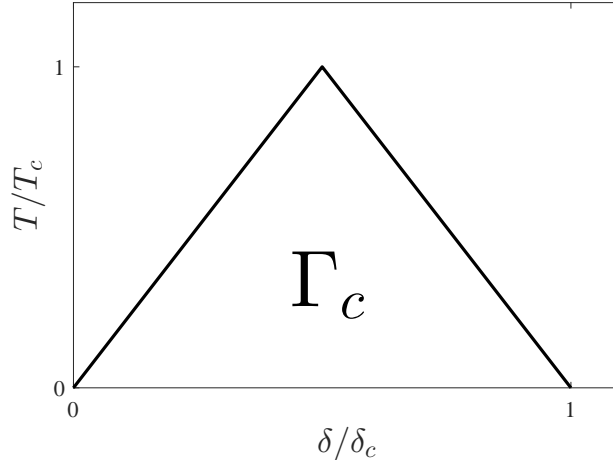$$\Gamma_c = \frac{1}{2} T_c \delta_c \tag{4}$$

Figure 5: Bi-linear traction separation law characterizing the cohesive zone model.

The two parameters governing the cohesive response can be inferred from experiments. Generally, a trial and error procedure is followed, but such methodology is time consuming and error-prone. Here, a novel technique that builds on inverse analysis and neural network optimization is proposed to estimate the parameters governing the traction-separation law. *Abaqus2Matlab* enables such an approach, not only by linking the advanced optimization tools available in Matlab with Abaqus damage modeling outcomes, but also by allowing to read and modify Abaqus input files accordingly. Thus, not only is *Abaqus2Matlab* useful for post-processing purposes but it can be also used to optimize and pre-process through a *two-way* interaction between Abaqus and Matlab. This is done by creating a Matlab function that reads the input file from Abaqus (*.inp) and, for this particular example, overwrites the line where the magnitude of δ_c and T_c are defined. The function can be downloaded from *Abaqus2Matlab*'s website and easily re-written to edit any other specific command.

The material under consideration in the present study is Aluminum 2024. Both uniaxial and Compact Tension tests have been performed. The former lead to a Young's modulus of $E = 85826$ MPa (Poisson's ratio $\nu = 0.33$) while the plastic behavior can be fitted through a Hollomon's law $\sigma = k\varepsilon_p^n$ with $k = 733$ and $n = 0.157$. As depicted in Fig. 6, the specimen has a width of $W = 50$ mm, a thickness of $B = 20$ and a total crack length of $a = 17.323$ mm, being the fatigue pre-crack equal to $a_0 = 7.323$ mm.
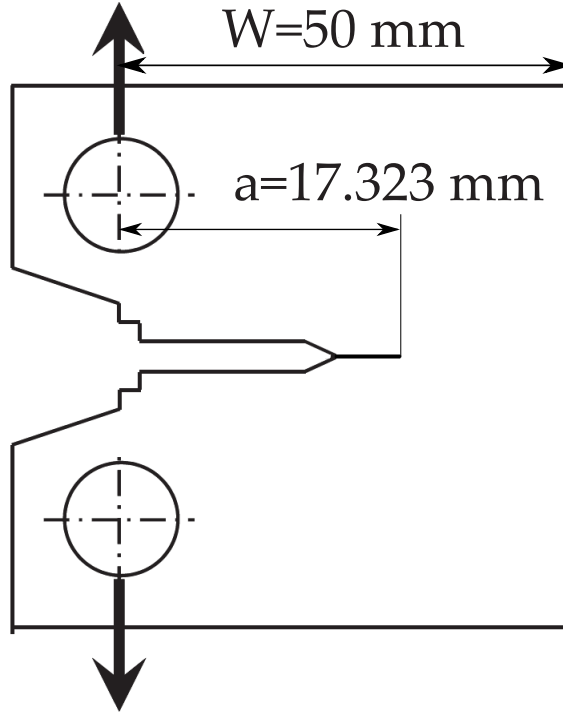
Figure 6: Geometry and dimensions of the Al2024 Compact Tension specimen.

The optimization procedure proposed correlates numerical results and experimental data of load versus crack mouth opening displacement (CMOD) by following the flowchart shown in Fig. 7. Thus, the first step involves assigning a set of initial values to T_c and Γ_c. These initial values should be chosen so as to span a considerably wide range, ensuring that the optimal solution falls inside. The more numerous the merrier, as the performance of the neural network increases with the number of points. Nevertheless, only 5 pairs of T_c vs Γ_c points will be employed in this example to show the model capabilities even with a few initial values (see Fig. 8a).
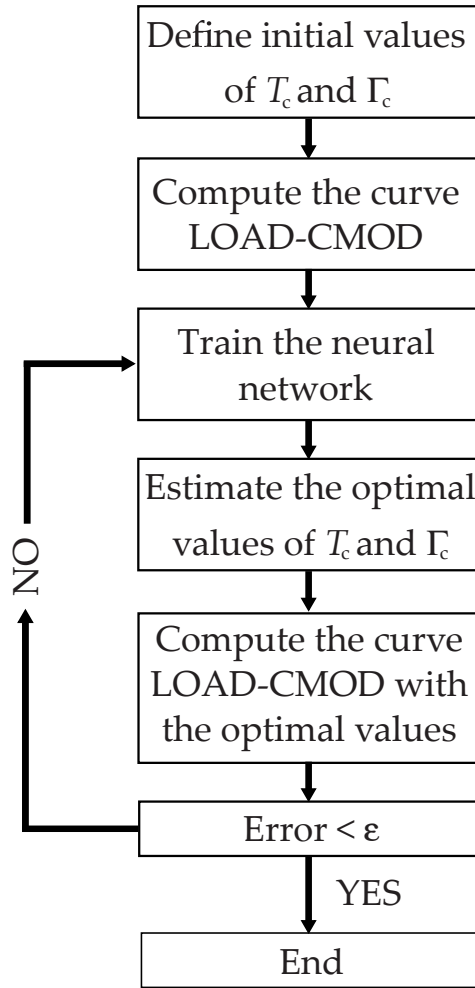
Figure 7: Neural network optimization flowchart.

The finite element calculations are then performed, where Abaqus capabilities to model cohesive zone damage are employed and a very refined mesh of quadrilateral quadratic plane strain elements with reduced integration is adopted. The curve load versus CMOD is obtained in *Abaqus2Matlab* by reading the nodal reaction forces and the displacement in particular sets (`Rec104` and `Rec101` functions). Computations are efficiently performed for each pair of T_c-Γ_c values by taking advantage of *Abaqus2Matlab* capabilities to read and modify Abaqus' input file. The results obtained in each case are shown in Fig. 8b; each curve is characterized by 12 equally distant points so

21
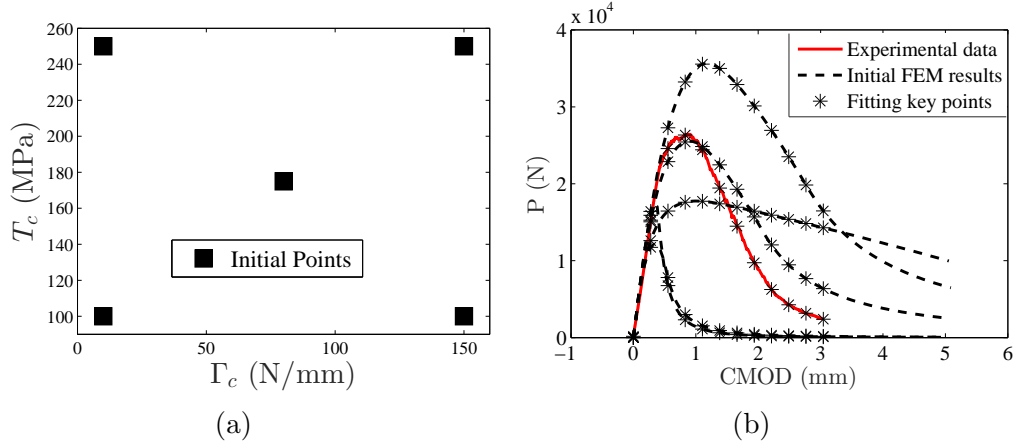
as to correlate with the experimental data.

(a)   (b)

Figure 8: Initial neural network training steps, (a) First set of T_c-Γ_c values, and (b) corresponding load versus crack mouth opening displacement curves.

The next step involves training the neural network based on the input (T_c and Γ_c values) and output (load versus CMOD curves) information. The network is composed of 10 hidden layers and is trained by employing the Bayesian Regulation Method available in Matlab (see Fig. 9); full advantage of the *Neural Net Fitting Matlab App* can be gained with *Abaqus2Matlab*. In this example, 80% of the models have been employed to train the network, 15% of them have been used for validation purposes and the remaining 5% serve to test the final solution obtained.
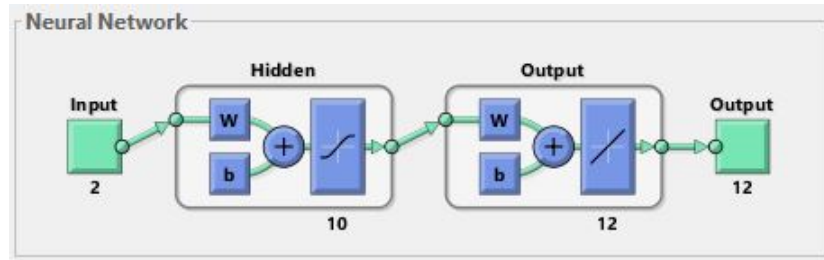

Figure 9: Graphical summary of the characteristics of the Neural Network employed.

Once the neural network is fitted and tested, it is used to estimate - through least squares fitting - the optimal values of T_c and Γ_c by minimizing the differences between the load-CMOD curve obtained from the model and

its experimental counterpart. To assess the quality of the neural network prediction, the optimized values of the cohesive strength and the cohesive fracture energy are provided as input to the finite element model. The outcome of this new finite element analysis is compared to the experimental data. If the norm of the differences between the curves is higher than a given tolerance, the neural network is trained again by adding new input and output information from the previous iteration. Fig. 10 shows the optimal values of the strength and the cohesive energy obtained in each iteration.
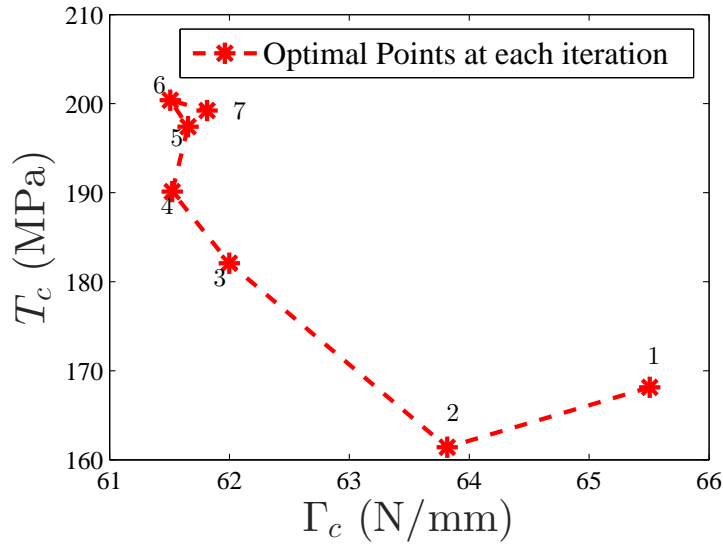

Figure 10: Cohesive strength T_c and fracture energy Γ_c estimations at each iteration.

In the present example convergence is achieved after 7 iterations and the final outcome is shown in Fig. 11, together with the experimental result. As it can be seen in the figure, the optimal values ($T_c = 199.2$ MPa and $\Gamma_c = 61.81$ N/mm) lead to a very good quantitative agreement with the load versus CMOD curve obtained experimentally.
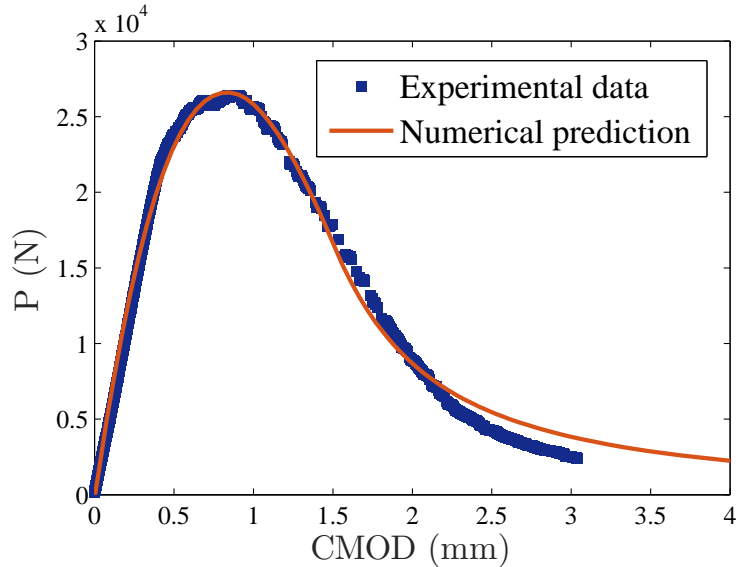
Figure 11: Experimental and numerically optimized predictions of load versus crack mouth opening displacement in Al2024.

Hence, quantitative insight into the initiation and subsequent propagation of damage can be obtained through neural network optimization and a hybrid experimental-numerical strategy, in what is usually referred to as a top-down approach [23, 24]. Thus, *Abaqus2Matlab* largely facilitates structural integrity assessment by taking advantage of advanced damage models available in Abaqus and modern optimization capabilities of Matlab. Moreover, its usage can be easily extended to a wide range of non-linear problems, where inverse analysis is an indispensable tool.

A detailed description, and the associated codes of the novel approaches employed in the two challenging engineering problems addressed, can be downloaded from *Abaqus2Matlab* website.

5. Conclusions

A novel toolbox has been presented to couple Abaqus and Matlab. Its development is motivated by the need of an open source package that provides a non-intrusive link between the highly-developed finite element capabilities of Abaqus and the comprehensive analysis tools of Matlab. The software, conveniently titled *Abaqus2Matlab*, unfolds an extensive range of

24

modeling possibilities. Its capabilities are particularly attractive from the post-processing perspective, enabling to complement advanced finite element simulations with the numerous graphical and mathematical analysis options of Matlab and its toolboxes.

Two practical cases with important implications in structural integrity assessment are investigated to illustrate the potential of *Abaqus2Matlab*. First, cleavage fracture is examined by means of a three-parameter Weibull approach. A novel statistical framework is proposed to estimate the modulus, the scaling and the threshold parameters through *Abaqus2Matlab* without any preceding assumptions. The software is also employed to model crack propagation in Al2024 by extracting the cohesive parameters from the experimental data through inverse analysis. *Abaqus2Matlab* plays a fundamental role by enabling model manipulation and genetic algorithm optimization. The general structure of the code facilitates its application to numerous engineering problems with minimum coding effort. Diverse examples (including the ones described here), comprehensive documentation and the source code can be downloaded from www.abaqus2matlab.com.

6. Acknowledgments

References

[1] ABAQUS Version 2016 Documentation. Dassault Systèmes, Simulia Corp., Providence (2016).

[2] MATLAB R2016a. MathWorks, Inc., Natick, MA (2016).

[3] Sigmund, O., 2001. A 99 line topology optimization code written in Matlab. Struct. Multidisc. Optim. 21, 120-127.

[4] Andreassen, E., Clausen, A., Schevenels, M., Lazarov, B.S., Sigmund, O., 2011. Efficient topology optimization in MATLAB using 88 lines of code. Struct. Multidisc. Optim. 43, 1-16.

[5] Bordas, S., Nguyen, P.V., Dunant, C., Guidoum, A., Nguyen-Dang, H., 2007. An extended finite element library. Int. J. Numer. Methods Eng. 71, 703-732.

[6] Nguyen, P.V., Rabczuk, T., Bordas, S., Duflot, M., 2008. Meshless methods: a review and computer implementation aspects. Math. Comput. Simul. 79, 763-813.

[7] Martínez-Pañeda, E., Natarajan, S., Bordas, S., 2017. Gradient plasticity crack tip characterization by means of the extended finite element method. Comput. Mech. doi: 10.1007/s00466-017-1375-6

[8] Giner, E., Sukumar, N., Taracón, J.E., Fuenmayor, F.J., 2009. An Abaqus implementation of the extended finite element method. Eng. Fract. Mech. 76, 347-368.

[9] Park, K., Paulino, G.H., 2012. Computational implementation of the PPR potential-based cohesive model in ABAQUS: Educational perspective. Eng. Fract. Mech. 93, 239-262.

[10] Nguyen, P.V., 2014. An open source program to generate zero-thickness cohesive interface elements. Adv. Eng. Software 74, 27-39.

[11] Martínez-Pañeda, E., Gallego, R., 2015. Numerical analysis of quasistatic fracture in functionally graded materials. Int. J. Mech. Mater. Des. 11, 405-424.

[12] Novák, D., Vořechovský, M., Teplý, B., 2014. FReET: Software for the statistical and reliability analysis of engineering problems and FReET-D: Degradation module. Adv. Eng. Software 72, 179-192.

[13] Lehký, D., Keršner, Z., Novák, D., 2014. FraMePID-3PB software for material parameter identification using fracture tests and inverse analysis. Adv. Eng. Software 72, 147-154.

[14] Fiala, P., Rucz, P., 2014. NiHu: An open source C++ BEM library. Adv. Eng. Software 75, 101-112.

[15] Hsieh, Y.-M., Pan, M.-S., 2014. ESFM: An Essential Software Framework for Meshfree Methods. Adv. Eng. Software 76, 133-147.

[16] Girardi, M., Padovani, C., Pellegrini, D., 2015. The NOSA-ITACA code for the safety assessment of ancient constructions: A case study in Livorno. Adv. Eng. Software 89, 64-76.

[17] Liu, Y., Cheng, L., Zeng, Q., Feng, Z., Zhang, L., 2015. PCLab A software with interactive graphical user interface for Monte Carlo and finite element analysis of microstructure-based layered composites. Adv. Eng. Software 90, 53-62.

[18] Svoboda, L., Šulc, S., Janda, T., Vorel, J., Novák, J., 2016. μMECH micromechanics library. Adv. Eng. Software 100, 148-160.

[19] Beremin, F.M., 1983. A local criterion for cleavage fracture of a nuclear pressure vessel steel. Metall. Mater. Trans. A 14, 2277-2287.

[20] Heerens, J., Hellmann, D., 2002. Development of the Euro fracture toughness dataset. Engng. Fract. Mech. 69, 421-449.

[21] Muñiz-Calvente, M., Ramos, A., Shlyannikov, V., Lamela, M.J., Fernández-Canteli, A., 2016. Hazard maps and global probability as a way to transfer standard fracture results to reliable design of real components. Eng. Failure Anlysis. 69, 135-146.

[22] Cornec, A., Scheider, I., Schwalbe, K.-H., 2003. On the practical application of the cohesive model. Engng. Fract. Mech. 70, 1963-1987.

[23] Martínez-Pañeda, E., García, T.E., Rodríguez, C., 2016. Fracture toughness characterization through notched small punch test specimens. Mater. Sci. Eng., A 657, 422-430.

[24] Martínez-Pañeda, E., Cuesta, I.I., Peñuelas, I., Díaz, A., Alegre, J.M., 2016. Damage modeling in small punch test specimens. Theor. Appl. Fract. Mech. 86A, 51-60.