An Abaqus plugin for efficient damage initiation hotspot identification in large-scale composite structures with repeated features

Xi Zou[a], Shibo Yan[a,*], Mohammad Reza Ilkhani[a], Louise Brown[a], Arthur Jones[a], Maxime Hamadi[b]

[a] *Composites Research Group, Faculty of Engineering, University of Nottingham, Nottingham, NG7 2RD, United Kingdom*
[b] *Airbus Operations S.A.S., 316 Route de Bayonne, Toulouse, 31300, France*

Abstract

Identifying the hotspots for damage initiation in large-scale composite structure designs presents a significant challenge due to the high modelling cost. For most industrial applications, the finite element (FE) models are often coarsely meshed with shell elements and used to predict the global stiffness and internal loads. Because of the lack of detailed descriptions for the composite materials and 3D stress states, most of the established failure criteria are not applicable. In this work we present an Abaqus plugin tool which implements a framework to identify the hotspots by using a pre-computed database generated for specific, heavily-repeated feature types based on a given structural model. Developed with an object-oriented implementation in Python, this software is split into two main parts, specifically for feature generation and structural analysis. The pre-computed model presents a full 3D description for the considered feature and works as a submodel to the coarse structure model driven by a one-way transfer of the boundary conditions. The presented framework is an analysis tool for efficient sizing of large-scale composite structures, as it enables 3D damage analysis of the structures in critical zones with significant savings of the modelling and computational cost. The results are compared with conventional FE modelling and satisfactory agreement is observed. In addition, the software also enables the pre-computed database to be stored in an HDF5 data file for further reuse on new structures with the same feature.

Keywords: Abaqus plugin, composite structures, damage initiation, object-oriented programming

*Corresponding author
Email address: shibo.yan@nottingham.ac.uk (Shibo Yan)

1. Introduction

In recent years, the applications of fibre reinforced polymer composite materials have moved towards large-scale structures such as aircraft fuselages and megawatt-class wind turbine blades due to their advantages which include high stiffness, high strength and light weight. Modelling of the damage behaviour of composite structures, e.g. delamination, usually requires 3D finite element (FE) discretisation of the structures. Using the conventional 3D FE modelling strategy presents a significant challenge for those large-scale composite structures in terms of very high computational cost. In the meantime, the need for efficient and cost-effective design of such structures necessitates access to affordable modelling tools. In particular, software suites that are able to detect hotspots [1], i.e. potential locations of damage initiation in these structures, are required in the iterative design process.

Conventionally, as a typical "deep dive" approach, the global-local coupling of FE models has been used to address the high computational cost issues resulting from analysing large composite structures [2, 3, 4]. In this method, the large composite structure discretised using linear elastic coarse shell elements works as the global model, and the 3D solid representations of selected sub-regions that require detailed damage analysis are considered as local models. Specific failure criteria are applied at the global level to determine the locations where local analysis should be performed. It will be considered as a two-way global-local method if the degraded material properties obtained from the local analysis are fed into the global model for the following analysis to account for the progressive damage behaviour. With this method, a reduction of approximately 50% in computational time was reported in Ref [5] by comparing with reference results. A multiscale surrogate modelling framework for composite materials considering progressive damage was proposed in Ref [6]. This method employs artificial neural networks and unit cell modelling to generate a macroscale surrogate model for the composite material properties and damage information, which is then used for the damage analysis of composite structures. The method was shown to offer a huge saving of computational cost by using the surrogate model whilst maintaining sufficient accuracy, yet bending-induced delamination, which is an important failure mode for laminates, was not accounted for. Alternatively, the submodelling approach, which is readily implemented in Abaqus, provides a one-way information transfer from the global model to the local model. This approach, without the necessity of coupling and iterative solutions, could save the computational cost with a compromise that only damage initiation can be accounted for.

Typical airframe structures like fuselages have a large number of repeated features, such as fillets and bonded joints, as airframes are large assemblies involving many repetitions of a limited number of parts. These features are usually the locations of potential damage hotspots in loaded structures and therefore subject to damage initiation. For instance, a fillet where two flat panels meet is prone to delamination as it is under a bending-dominated load case. The motivation for the present work is to provide increased agility

in the airframe design process by removing the need for feature-by-feature "deep dive" modelling of the critical high-stress sites. Obviously, performing submodelling analysis for every feature instance in a large scale global model is impractical. An efficient tool has therefore been created to accelerate this process for at least a preliminary filtering of the hotspots, so that the conventional high-fidelity model could be later created in a limited number for further analyses. The main objective of this work is to develop this tool utilising an existing FE code, in this case Abaqus, to facilitate the hotspot filtering or identification in a non-intrusive manner for large-scale composite aero-structures.

As a well-established FE analysis software suite with excellent APIs for user-specified functionality and algorithm development, Abaqus has been used in numerous studies regarding the stress analysis, damage and failure of composite structures, and especially for cohesive zone modelling [7, 8]. However, most of the works involving user-added algorithms or functionalities, such as the extended FE methods, require quite intrusive coding in the form of Fortran subroutines [9, 10, 11, 12, 13]. Sometimes it is necessary to couple Abaqus with third-party codes, and this is mostly seen in the literature that using Fortran subroutines to perform such interactions by coding user-defined interfaces [14, 15].

Alternatively, a few authors are also taking advantage of the Python APIs provided by Abaqus to create user-defined analysis procedures, such as fatigue analysis [16], mesh manipulation and remeshing [17], and multiscale analyses [18, 19, 20]. This approach keeps using the established functions and solvers in a non-intrusive manner, and is also extendible to couple with third-party software for implementing novel algorithms [21]. In this work, we adopt the non-intrusive approach and make use of the object-oriented feature of the Python language for efficient development of the code which is in form of an Abaqus plugin. For numerous code developments including many FE solvers, object-oriented programming (OOP) is the mainstream concept and a large number of previous works have been published as reviewed by Ref [22]. It is noted that there are also studies using both approaches, mixing Fortran and Python codes, to implement complex software, such as Ref [23].

The present study aims to develop an OOP-based software using a nearly non-intrusive approach, with the purpose of providing an efficient and simple design tool for industrial applications in damage initiation hotspot identification for composite material airframes. The paper is structured as follows: Section 2 introduces the underlying theory which serves as the basis of the overall workflow. Design and implementation details are introduced in Section 3 providing information about the OOP and GUI structure, including some key code segments. A case study is presented in Section 4, using a typical fillet feature that is commonly seen in most structures. Finally, the conclusions are drawn in Section 5.

This work is part of the CleanSky 2 project "Multiscale Analysis of AiRframe Structures and Quantification of UncErtaintieS System" (MARQUESS), which also includes a goal-oriented error estimator of the presented workflow [24].

2. Theoretical basis

In this section, we briefly introduce the underlying theory for the plugin. Most of the FE-related theories are already available in Abaqus/Standard, which can be accessed by its Python APIs. However, the failure criteria included in Abaqus are not directly applicable to the prediction of damage initiation within composite materials under 3D stress states. In particular, the delamination criteria need to be implemented by the analyst.

Due to the large size of the structural model, *globally* it may exhibit a nonlinear behaviour with large displacements. For the purpose of simplification, we assume that the *local* response of the material to the load is linear elastic. This is because damage initiation often occurs locally at an early loading stage for most composite structures. Thus the loads, when applied to the feature model, are assumed to elicit a linear response up to damage initiation and hence linear superposition is assumed to be applicable up to that point.

2.1. One-way submodelling

When considering the submodelling approach, there will be a *global or parent model* from which the boundary conditions are extracted and possibly post-processed, and a *local model or submodel* where the extracted information will be enforced at its *driven boundary*. In this paper we will use a large-scale structural model as the global model, and multiple feature models as its submodels. The global model is coarsely meshed with shell elements, while the feature models are 3D models describing all plies in a small local portion of a composite part. To alleviate the likely stress concentrations due to the abrupt mesh density change along the boundary, an intermediate model which contains only the corresponding portion of the same composite part is created with shell elements. This intermediate model is meshed using a density between that of the global model and the submodel, and is inserted in between them, making the whole analysis a two-level submodelling process, namely:

1) Global model to intermediate model;
2) Intermediate model to 3D feature model.

At each level, the latter is the submodel of the former.

In this subsection, to be concise but without losing generality, only a one-level submodelling theory is presented, though extending to multi-level cases is straightforward.

There are typically two types of submodelling according to the information extracted from the global model: node-based and surface-based. For each type, there are also two options: to extract displacement or force as the driven information. In this work we consider only the node-based, displacement-driven approach. This option uses the displacement field to transfer information from global model to the submodel. Starting

from standard FE theory, the basic equation which represents the submodel is given by

$$\mathbf{KU} = \mathbf{F}, \tag{1}$$

where $\mathbf{K}$ is the stiffness matrix of the submodel, $\mathbf{F}$ is the nodal force vector, $\mathbf{U}$ is the nodal DOF vector, which interpolates the displacement field $\boldsymbol{u}(\boldsymbol{x})$ via the element shape functions. The shape functions are denoted as a matrix $\mathbf{N}(\boldsymbol{x})$, it is well-known $\boldsymbol{u}(\boldsymbol{x}) = \mathbf{N}^{\mathrm{T}}(\boldsymbol{x})\mathbf{U}$. In practice, no external forces are applied to the submodel, only displacements on the driven DOFs, denoted as $\mathbf{U}_{\mathrm{d}}$, which is a subvector of $\mathbf{U}$. The remaining free DOFs also form a vector $\mathbf{U}_{\mathrm{f}}$ such that the displacement DOF vector can be rearranged as $\mathbf{U} = [\mathbf{U}_{\mathrm{d}}^{\mathrm{T}}, \mathbf{U}_{\mathrm{f}}^{\mathrm{T}}]^{\mathrm{T}}$. Afterwards, a static condensation can be made to the submodel equation:

$$\begin{bmatrix} \mathbf{K}_{\mathrm{d}} & \mathbf{K}_{\mathrm{df}} \\ \mathbf{K}_{\mathrm{df}}^{\mathrm{T}} & \mathbf{K}_{\mathrm{f}} \end{bmatrix} \begin{bmatrix} \mathbf{U}_{\mathrm{d}} \\ \mathbf{U}_{\mathrm{f}} \end{bmatrix} = \begin{bmatrix} \mathbf{F}_{\mathrm{r}} \\ \mathbf{0} \end{bmatrix}, \tag{2}$$

where $\mathbf{K}_{\mathrm{d}}$, $\mathbf{K}_{\mathrm{df}}$, $\mathbf{K}_{\mathrm{f}}$ are submatrices of the stiffness after the DOF split, and $\mathbf{F}_{\mathrm{r}}$ is the nodal reaction force vector due to the enforcement of displacement.

Solving (2), the submodel's displacement DOF is obtained as

$$\mathbf{U} = \begin{bmatrix} \mathbf{U}_{\mathrm{d}} \\ \mathbf{U}_{\mathrm{f}} \end{bmatrix} = \begin{bmatrix} \mathbf{U}_{\mathrm{d}} \\ -\mathbf{K}_{\mathrm{f}}^{-1}\mathbf{K}_{\mathrm{df}}^{\mathrm{T}}\mathbf{U}_{\mathrm{d}} \end{bmatrix} = \begin{bmatrix} \mathbf{I} \\ -\mathbf{K}_{\mathrm{f}}^{-1}\mathbf{K}_{\mathrm{df}}^{\mathrm{T}} \end{bmatrix} \mathbf{U}_{\mathrm{d}}, \tag{3}$$

where $\mathbf{I}$ is the identity matrix.

Letting $\mathbf{D}$ be the elastic matrix of the submodel, and $\mathbf{B}$ be the well-known strain-displacement matrix, the stress result can be written as

$$\boldsymbol{\sigma} = \mathbf{DBU} = \mathbf{DB} \begin{bmatrix} \mathbf{I} \\ -\mathbf{K}_{\mathrm{f}}^{-1}\mathbf{K}_{\mathrm{df}}^{\mathrm{T}} \end{bmatrix} \mathbf{U}_{\mathrm{d}}. \tag{4}$$

The formulation of (4) can be further simplified as

$$\boldsymbol{\sigma} = \mathbf{MU}_{\mathrm{d}}, \tag{5}$$

with the so-called *M-matrix* defined by

$$\mathbf{M} := \mathbf{DB} \begin{bmatrix} \mathbf{I} \\ -\mathbf{K}_{\mathrm{f}}^{-1}\mathbf{K}_{\mathrm{df}}^{\mathrm{T}} \end{bmatrix}. \tag{6}$$

The submodelling functionality provided by Abaqus includes a so-called *shell-to-solid* submodelling approach, which can perform an automatic redistribution and interpolation for degrees of freedom (DOF) extracted from a shell element to its corresponding solid counterpart. This means the rotational DOF of a shell element can be converted to equivalent translational DOFs for its corresponding elements along the through-thickness direction. For further details, see Section 10.2.2 in [25]. Now letting the displacement

DOF vector extracted from the global model be $\mathbf{U}_\mathrm{G}$, the boundary condition applied to the submodel can be written as

$$\mathbf{U}_\mathrm{d} = \mathcal{I}(\mathbf{U}_\mathrm{G}), \tag{7}$$

where $\mathcal{I}$ denotes the shell-to-solid DOF translation operator which is equivalent to the displacement transformation implicit in Abaqus shell-to-solid submodelling. It can be verified that this operator is linear. Therefore, the stress results of the submodel can be obtained with the M-matrix and the DOF vector $\mathbf{U}_\mathrm{G}$ obtained from the global model. Effectively, this enables fast interrogations for any possible positioning of the submodel on the same global model, ie. arbitrary value of $\mathbf{U}_\mathrm{G}$.

The size of M-matrix depends on:

`nip` the total number of integration points where the stress results are evaluated in the submodel,

`ndof` the total number of DOFs driven nodes, ie. the dimension of $\mathbf{U}_\mathrm{d}$.

Taking a submodel meshed with Abaqus C3D8R elements as an example, the M-matrix has a dimension of $(6 \times \texttt{nip}) \times \texttt{ndof}$. Here the coefficient 6 comes from the number of independent stress components calculated at each integration point. In practice, to obtain the M-matrix, a unit value is applied on each individual DOF in $\mathbf{U}_\mathrm{d}$, resulting in a batch of corresponding stress results which are eventually stored in a Hierarchical Data Format version 5 (HDF5) file as a database for the feature model. This batch running is performed by setting `ndof` independent steps in one Abaqus analysis job that is submitted once, obtaining a result `.odb` file. A sample implementation using Python and NumPy is presented in Listing 1.

Listing 1 Sample code segment for obtaining M-matrix from an Abaqus result file.

```
from odbAccess import *
import numpy as np
odb = openOdb('submodel.odb') # open the odb file
M = np.empty([6*nip, ndof])   # initialise the M-matrix
# obtain values
for i, step in enumerate(odb.steps.keys()):  # loop through all unit DOFs
    frame = odb.steps[step].frames[1]
    for S in frame.fieldOutputs['S'].values: # loop through all integration points
        try:
            stress = np.r_[stress, S.data]
        except NameError:
            stress = S.data
    M[:, i] = stress
```

Once the geometry and material are given, generation of the M-matrix is an independent procedure, thus it can be pre-computed without any information from the global model. Having obtained the database

HDF5 file, the corresponding stress field in the submodel can be computed quickly by prescribing a global DOF vector $\mathbf{U}_{\mathrm{G}}$, whose values are extracted from the interrogation location in the global model. Further post-processing can be carried out upon the stress results obtained through the M-matrix, such as failure analysis based on a particular criterion.

2.2. Failure criteria

The proposed method produces a 3D stress state within features of interest on which well-established unidirectional (UD) composite failure criteria can be readily applied. To demonstrate the workflow, the maximum stress criteria are used in the plugin for the composite materials although other more complex ones can be implemented. The maximum stress theory is the simplest failure criterion for homogenised UD composites, which can be used to predict the failure in the principal material axes. Failure occurs if one of the stresses is greater than the corresponding allowable stress value in that direction. Note that there is no interaction between stresses, i.e. the stress in one material direction would not affect the failure in another material direction. An example for the maximum stress criteria under the plane stress state $(\sigma_1, \sigma_2, \tau_{12})$ is given below:

$$\begin{aligned} -F_1^{\mathrm{c}} < \sigma_1 < F_1^{\mathrm{t}}, \\ -F_2^{\mathrm{c}} < \sigma_2 < F_2^{\mathrm{t}}, \\ |\tau_{12}| < F_{12}, \end{aligned} \tag{8}$$

where F is the material strength, where subscripts 1 and 2 denote the fibre and transverse directions in the local coordinate system of the yarn (fibre), and subscripts c and t denote compression and tension.

The interface damage can also be evaluated after recovery of the 3D stress field in the feature. The initiation of interface damage occurs when the nodal stress components meet a specified criterion. Here a quadratic stress-based criterion for damage onset is adopted [26],

$$f(\boldsymbol{\sigma}) = \left(\frac{\langle \sigma_{\mathrm{n}} \rangle}{\sigma_{\mathrm{n}}^0}\right)^2 + \left(\frac{\tau_{\mathrm{s}}}{\tau_{\mathrm{s}}^0}\right)^2 + \left(\frac{\tau_{\mathrm{t}}}{\tau_{\mathrm{t}}^0}\right)^2 = 1, \tag{9}$$

where $\boldsymbol{\sigma}$ is the stress tensor with components $(\sigma_{\mathrm{n}}, \tau_{\mathrm{s}}, \tau_{\mathrm{t}})$, whilst σ_{n}^0, τ_{s}^0 and τ_{t}^0 denote the damage initiation stress components in the normal, first shear and second shear directions for each of the single-mode delamination modes I, II and III, respectively. The Macaulay operator, a function which takes the value of its argument only when that value is positive, is denoted by brackets $\langle\rangle$. Damage is assumed to initiate when the left-hand side reaches 1 that is $f(\boldsymbol{\sigma}) = f(\sigma_{\mathrm{n}}, \tau_{\mathrm{s}}, \tau_{\mathrm{t}}) = 1$. A sample implementation of the delamination damage initiation evaluation at the interface using Python and NumPy is presented in Listing 2.

Listing 2 Sample code segment for identifying hotspots at the interface.

```
import numpy as np
# extract components from input stress array
sigma_n = nodal_stress[:, 1]
tau_s = nodal_stress[:, 2]
tau_t = nodal_stress[:, 3]
sigma_n[sigma_n < 0.0] = 0.0    # the Macaulay operation
# evaluate quadratic criterion
f = (sigma_n/sigma_n_0)**2 + (tau_s/tau_s_0)**2 + (tau_t/tau_t_0)**2
hotspot_delamination = np.zeros(size(sigma_n))
hotspot_delamination[f>=1] = 1  # set the hotspot index for delamination to 1
```

3. Software design and implementation

The software is developed using Python, taking advantage of its applicability to object-oriented cross-platform programming. Also, it is a key component of the Abaqus package, which provides well-documented APIs for further development. For the best compatibility with Abaqus, the work flow is implemented as a plugin, which integrates all necessary functionalities. There are also many third-party libraries available in repositories such as PYPI and Anaconda, meeting a wide range of requirements. The h5py library [27] is used to handle the input/output data file for this project. This section describes several aspects of the overall design of the software. The main workflow is illustrated in Figure 1.

3.1. Software architecture

As shown in Figure 1 the system is divided into three distinct sections: specifications of input and output data, user interface and software kernel. This section describes the structure of the software kernel with subsequent sections describing how this interacts with the user interface and data.

As far as possible an object-oriented design has been used. The class structure of the kernel is shown in Figure 2. A single `CMarquess` object is generated when the plugin is initialised, providing an interface between the GUI and custom Python classes. It utilises the Abaqus customKernel module which allows a customData member to augment the model database (`mdb`) object to enable data to be transferred between the GUI and the plugin classes.

The GUI described in the following sections allows the user to access the core functions, via the `CMarquess` class, by selection of the appropriate module.

The `CFeatureModel` class is used by the Feature Generation module to create the feature submodels and run the analyses to generate the M-matrix which is then stored in the HDF5 database. Derived classes, e.g. `CFilletFeature`, are created for each type of feature. This facilitates the extension of the system

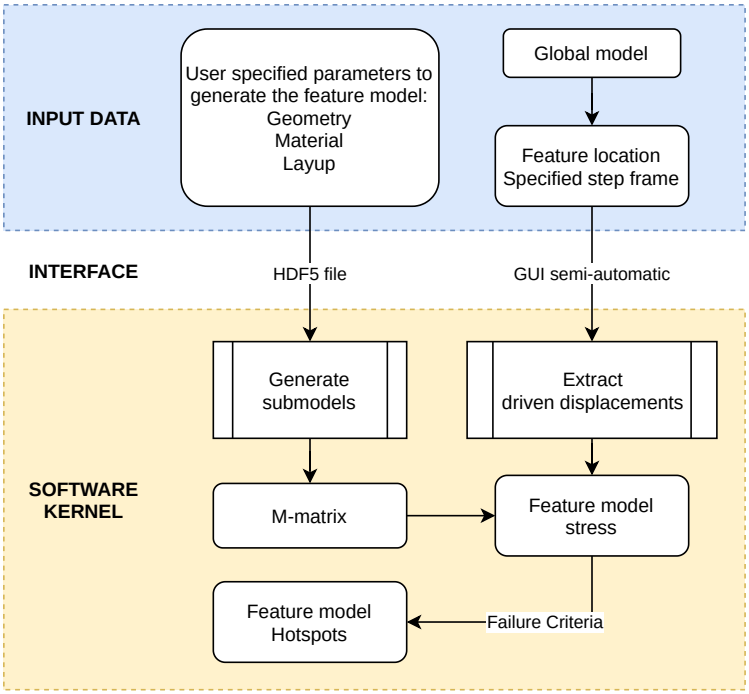

Figure 1: Illustration of the general design for the workflow.

by providing the framework to create new features by creating a new derived feature class containing the information specific to that feature type, as indicated by the `CDerivedFeatures` class in Figure 2. This would typically be geometry and also material and layup information which use the `CMaterial` and `CLayup` classes to contain the relevant information.

The `CFeatureLocation`, `CLocalModelData` and `CVisualization` classes are all used by the Hotspot Analyser module. The `CFeatureLocation` class is used to create the transformation to place the selected feature onto a particular instance of that feature in the global model as described in Section 4.2. There are derived classes for different feature types to take into account their different transformation procedures.

`CLocalModelData` calculates hotspot failure indices using information from the feature location as input. The `CVisualisation` class then saves the results to an odb file in the transformed locations to enable them to be displayed in position on the global model. Both classes require a coordinate transformation between the global and local coordinates systems using the transformation matrix calculated by the `CFeatureLocation` class.

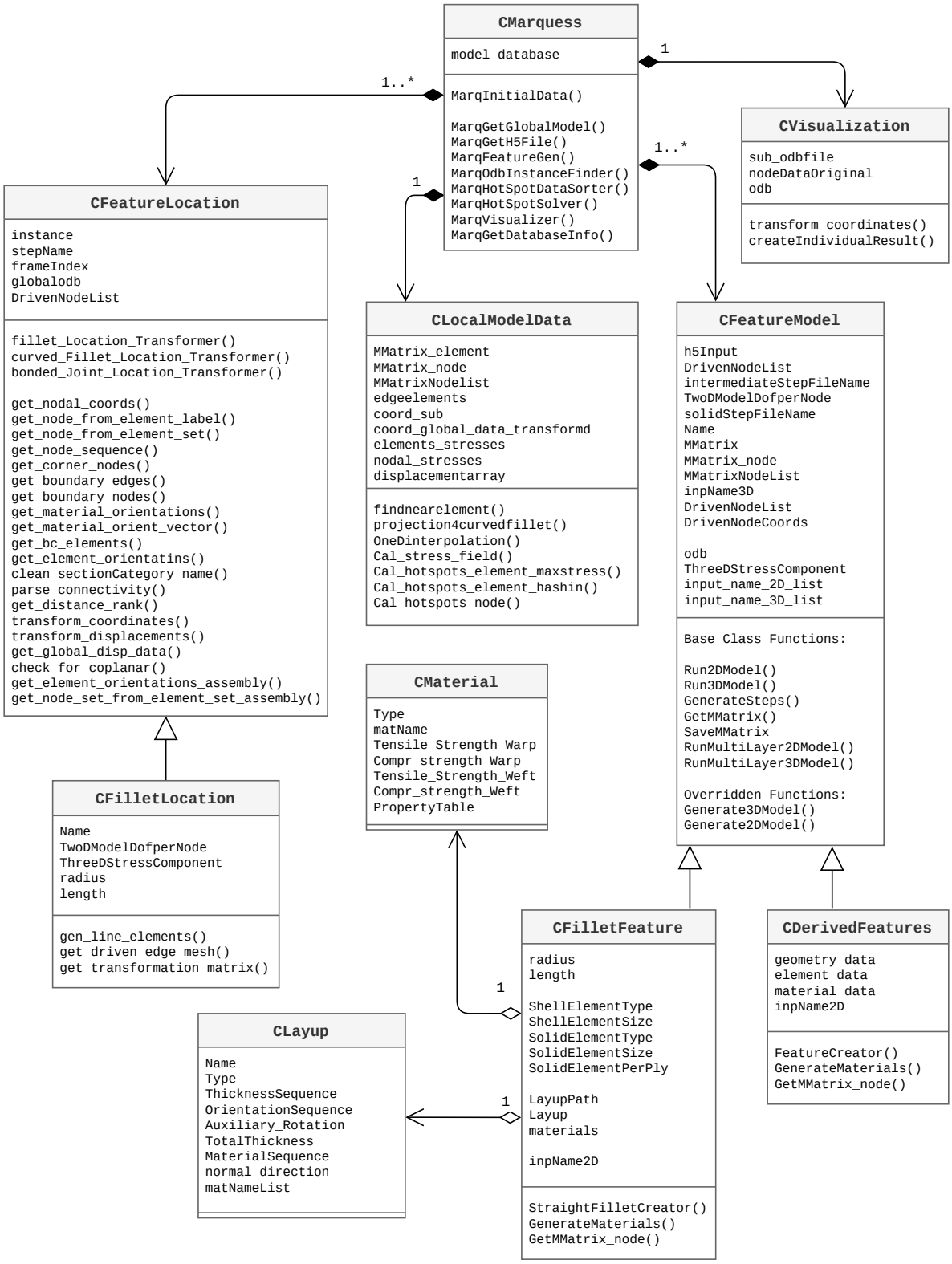

Figure 2: Class diagram for the software kernel.

3.2. Graphical User Interface design for user interaction

As mentioned in the previous section, especially in Figure 1, a graphical user interface (GUI) is designed for interacting with the user. The GUI is designed to facilitate easy transfer of data and information between the user and the developed software kernel, including checking of data compatibility with the kernel requirements and enabling transfer and selection of data through different stages of the analysis process. This is a two-way GUI that has been implemented using the Abaqus GUI toolkit (AFX) which is an extension to the FOX GUI Toolkit [28], a Python wrap of the FOX library based on C++, and Abaqus customised repositories. Because of the large size of the global input models it is important that the user is able to modify input parameters in order that results from previous analysis are not lost. To achieve this goal, classes that are in communication with the GUI classes are inherited from the Abaqus custom repository and FOX classes. Also, data that have been shared between the GUI and kernel are stored in in different Abaqus custom repositories based on their data types.

In addition to being two-way, the GUI has been made so that it is user-friendly, reducing the number of data inputs, for example by providing search abilities among the global model's element sets helping the user to find all of the desired element sets instead of entering them all manually. These functionalities are described in more detail in the following sections.

Because the software runs the hotspot analysis for huge composite structures, user input is checked in the GUI classes before sending data to the kernel classes. In the event of input errors such as data type typo and error in address, warning windows alert the user in order to prevent possible issues during the main failure analysis. Also, a comprehensive HTML user guide section has been prepared that includes a detailed explanation about the workflow, structure and functions of the software and is accessible through a help button at the main window of the software as shown in Figure 3.

To initiate the software, the main menu is launched by clicking the button in the Abaqus plugin menu bar as shown in Figure 3. As listed in the left pane, the software has four main modules, namely: Feature Generator, Hotspot Analyser, Results Visualiser, and Database Viewer. Each module is implemented in an individual pop-up window after selecting the corresponding radio button and clicking the "Next" button.

In the first module, Feature Generator, the pop-up window requires the input file path for the HDF5 file that includes information necessary to generate the feature model. Details about this input file are discussed in Section 3.3.1. Having read the data from the HDF5 file, the user-defined features will be populated in a list as shown in Figure 4. The user can then select the desired features from the populated list to generate the pre-computed FE model and store the corresponding M-matrices in the output feature database which is described in Section 3.3.2. This window is an example of communication between the kernel and GUI that has been developed in this software. The GUI gets data, sends them to the custom repositories, calls the kernel classes, reads the data from the HDF5 file using the h5py library, writes data back to repositories again and populates the feature names in the list in the window.

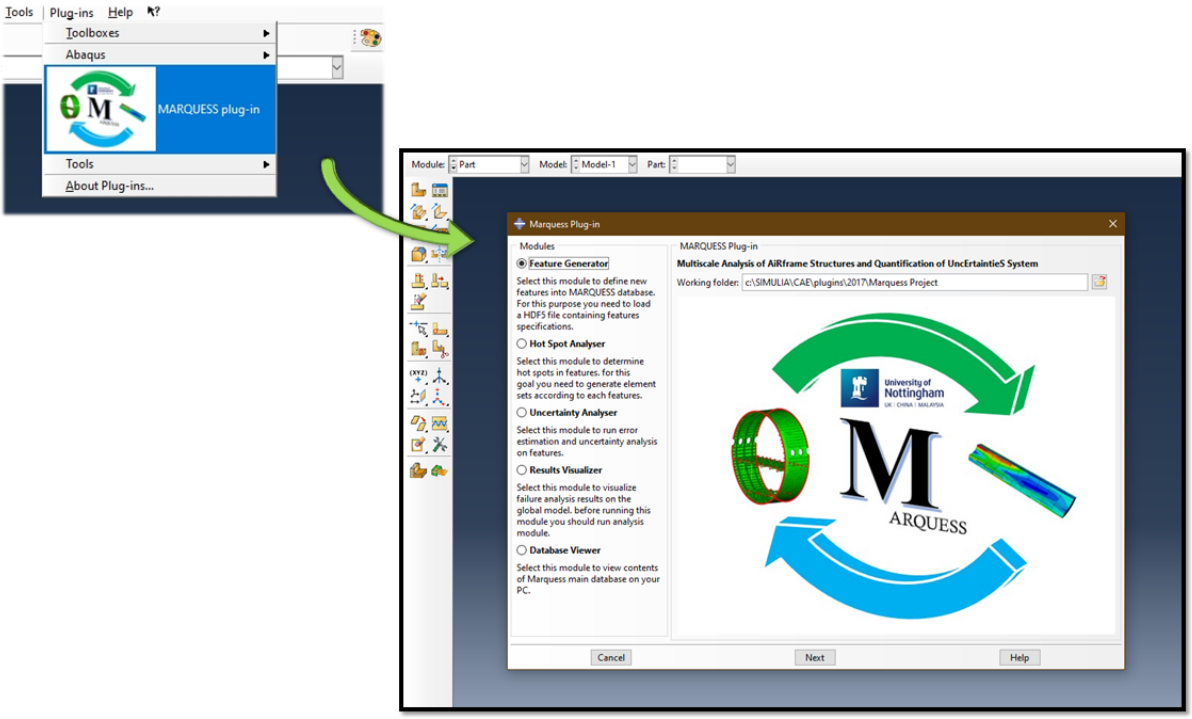

Figure 3: Main menu of the developed software running as a plugin within Abaqus/CAE.

In the second module, Hotspot Analyser, the pop-up window requires the user to select the input HDF5 file and the global model output database (`.odb` file). In addition to these files, the user provides the element set names that are desired to perform the damage initiation analysis. As picking these from the 3D view of the global model would be cumbersome for large structures with numerous element sets, two search fields and an object tree have been provided in the pop-up window to help the user to find all the related element sets and select the desired ones from the object tree, as depicted in Figure 5. In this figure, for instance, part of a stiffened panel with multiple instances of a stabiliser has been considered. Each stabiliser has two element sets representing the straight fillet which are assumed regions of interest. To select the fillets, the full names of all the instances and element sets can be typed into the text box. To speed up the process, the GUI enables the user to input part of the name of the instances fillet element sets in the search fields Figure 5(a) which facilitates the search in the global model. All the instances found and corresponding element sets are used to populate the object tree and the user can select or deselect any of them easily by using the check-boxes as shown in Figure 5(b). This utility simplifies the feature selection procedure and widens the applicability of this software to large composite structures. In the whole procedure, the GUI communicates with the kernel and sends and receives data required for the next steps using the two-way

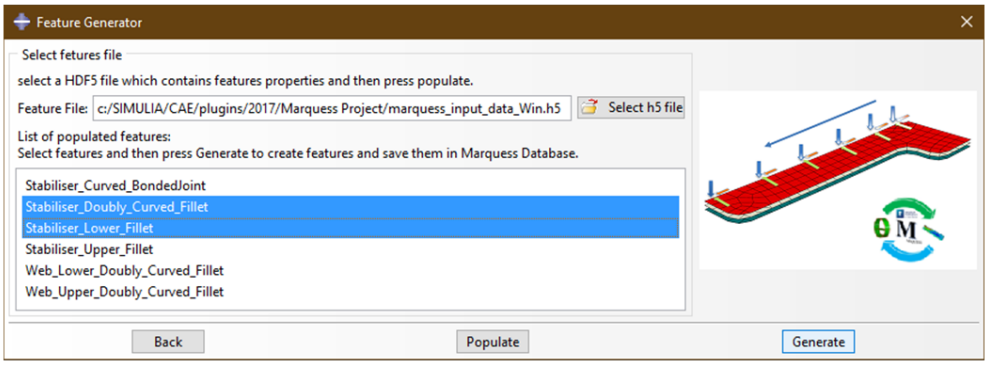

Figure 4: Overview of the feature generator window, defined features are shown in the bottom window.

functionality that has been developed in the GUI.

The Hotspot Analyser is also able to extract the analysis steps from the global model, so that the user can select the desired step name and increment number from the drop-down list generated, as shown in Figure 5(c). This function prevents errors due to typos or missing increment number.

After starting the analysis, another pop-up window shows the status of the analysis in a progress bar. In the meantime, the user can monitor the status of the analysis step-by-step by tracking the messages that are printed in the Abaqus message area.

During the analysis, the proper coordinate transformation of the 3D feature model is performed which arranges the feature in place on the global model. Finally, a result `.odb` file is generated with hotspot indices for visualisation in Abaqus.

In the third module, Results Visualiser, the `.odb` files of both the global model and the hotspot results are read into Abaqus/CAE, and are plotted in an overlaid view, as shown in Figure 6. Indeed, this module presents an overall view of the status of the designed structure and the user can benefit from a straightforward perception of the status of the damage initiation hotspots and can spot the critical locations in the global model. An overlaid plot of an example result is illustrated in Figure 6 within Section 4.3.

The last module in the GUI is the Database Viewer. It is a supplementary tool for checking the HDF5 database that the software has generated during the feature generation, and is especially useful if the user does not have an HDF5 viewer. It visualises all the feature information stored in the database, including the names of the features, dimensions of the matrices, etc. This module helps users who are not familiar with the HDF5 file format to check the status of the input database and to ensure that all required data are included in the database before it being imported into the generation or analysis modules.

3.3. Data storage scheme

The user may wish to parameterise fully the feature model, to give the flexibility to reuse existing Python scripts for FE model generation in Abaqus/CAE. To fulfil this, the software is designed to use an HDF5

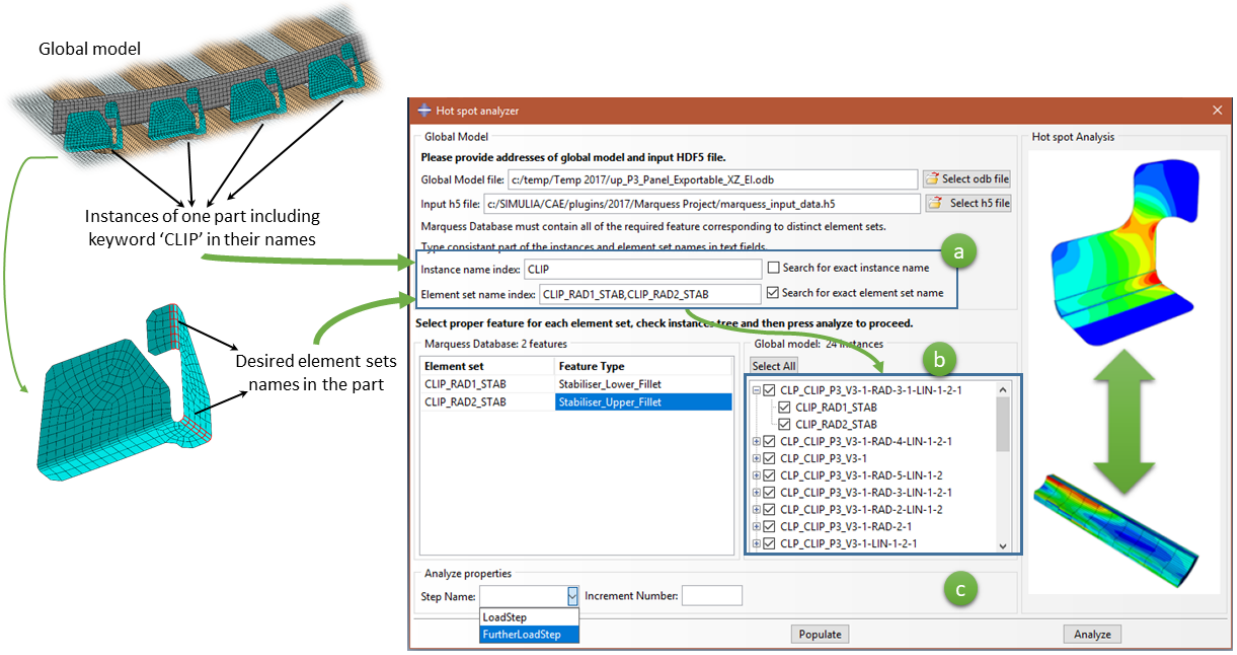

Figure 5: Graphical illustration of the use of search abilities of the Hotspot Analyser window: (a) search fields; (b) tree structure; (c) analysis step specification drop-down list.

database file to store the user inputs. Also, the M-matrix generated via the pre-computed feature models is stored in another HDF5 data file to enable reuse for analysis of the same feature located in a variety of global models.

As mentioned above, the HDF5 file format is adopted for the input/output. Taking advantage of the support for an unlimited variety of data types, it is designed for flexible and efficient I/O and for high volume and complex data. For further details about the HDF5 file format and the related library, the readers are referred to [29]. A Python interface to the HDF5 library is provided by h5py, which enables easy manipulation of the data in terms of NumPy arrays.

3.3.1. Input database

The input data contains two parts: the global model results in the form of an Abaqus `.odb` file, and the HDF5 data file containing user-specified parameters to enable the generation of pre-computed feature models.

The global model results obtained from a standard Abaqus analysis are stored in the `.odb` file. In this file, element sets containing the regions of interest should be defined by the user who specifies the names of the sets. The load step information for the analysis is also stored in the `.odb` file. As the details are portrayed in Section 3.2, the user is able to interact with the GUI with the knowledge of the global model.

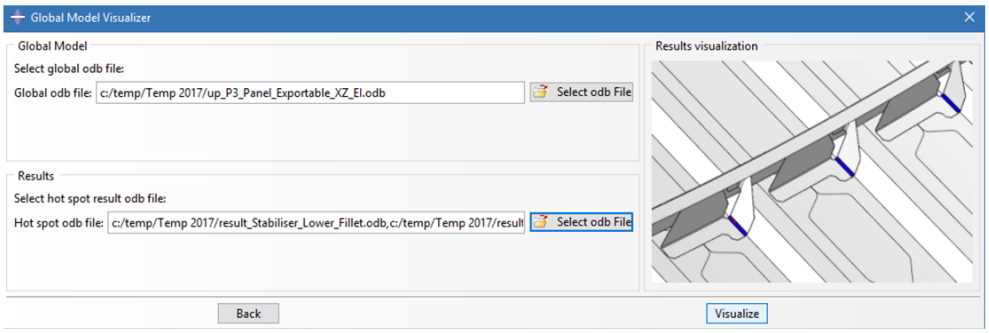

Figure 6: Result visualiser window specifying global model `.odb` file and feature result `.odb` files.

The input HDF5 data file often includes, but is not limited to, geometrical parameters of the feature, material properties such as Young's modulus and strength, as well as the layup information of the composite. An additional parameter `Auxiliary_Rotation` is also introduced to adapt the template feature model's material coordinate system to that of the global model. For instance, a template feature model of a fillet feature is shown in Figure 7. In the template feature model, the 1-direction of the material is parallel to the straight edge of the fillet. In the case that the actual feature has a different material orientation, the `Auxiliary_Rotation` parameter can be specified to adjust the feature model by rotating the material coordinate system along the 3-axis (determined by axes 1 and 2 using the right-handed rule). Since two-scale submodelling is used in this framework, for both the intermediate model and the submodel, the finite element type and size are also specified in the input data. In addition, it is also possible to specify how many solid elements per ply should be used in the 3D submodel. A detailed example code segment is listed in Listing 3. The resulting HDF5 file can be viewed using the HDFView software [30], and a screenshot is shown in Figure 8.

As the actual feature in the global model may occur in the form of multiple instances which are situated in different locations, proper coordinate system transformation is required and this will be explained in Section 4.2 with a practical example. To facilitate the coordinate transformation, some *anchor points* are also defined as shown in Figure 7 with orange dots. These anchor points are implicit input which are only represented in the feature generation codes, however, the user should specify such information when creating a new feature model.

3.3.2. Feature model database

Similar to the structure of the input HDF5 data file, the result data of the feature generation is stored in another HDF5 file. This feature database includes the M-matrix and driven node set for each feature model. During the analysis process, the result HDF5 file is read by the software as it contains the stiffness-like information of each feature. This feature model database can also be reused for variant global models

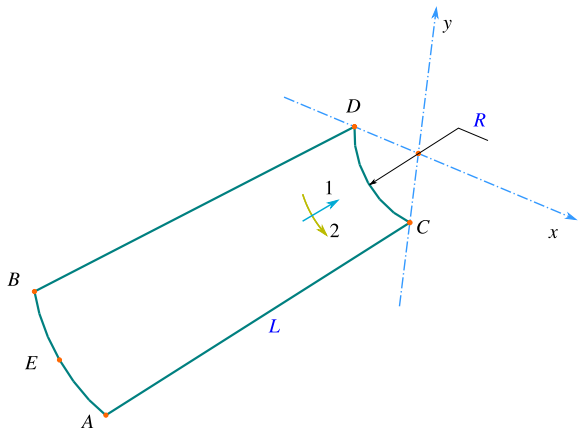

Figure 7: A template feature model for a fillet, showing the shell midsurface. The spatial axes are labelled with letters x and y but not showing z-axis. Material coordinates are labelled with numbers 1 and 2 but not showing 3-direction. Anchor points are marked in orange dots and labelled with letters A, B, etc.

containing the same feature.

3.4. Documentation system

As an important accompaniment to the software, the documentation is generated using Sphinx, which is a commonly used Python library for automatic documentation preparation. The generated documentation is in HTML format and ready to be deployed on web servers.

4. Case study

In this section, we present a case study of the software's workflow on a curved stiffened panel model under bending. **The software focuses on a set number of component features in the global model that have been identified as critical sections of components through user experience. In this way, a stress analyst's time is not devoted to performing tedious and monotonous refined simulations of commonly applied features and a greater emphasis can be placed on verifying location-specific features through traditional "dive deeper" means.** As depicted in Figure 9, the global model comprises four types of parts: the skin of the panel, curved frames, stiffening stringers and stabilisers that connect the frames and other parts. **Critical feature types (potential hotspots) identified by the users are straight fillets of the stabilisers, curved fillets of the stabilisers and curved bounded joints between skin and stringers, as they are the weak points in the structures due to interlaminar stress.** To simplify the presentation here, the region of interest is limited to the straight fillets of the stabilisers. As illustrated in Figure 10, the overall workflow includes three main procedures: feature generation, hotspot analysis and result visualisation. These procedures

Listing 3 Sample code segment for generating input HDF5 file.

```
import h5py
input_file = h5py.File('input_data.h5', 'w')
materials = input_file.create_group('Material')
mat1 = materials.create_group('CFRP')
mat1['E11'] = 94200.                                # unit in MPa
......
layups = input_file.create_group('Layup')
fillet_layup = layups.create_group('Composite_fillet')
fillet_layup['Thickness_Sequence'] = (0.2, 0.2, 0.2) # unit in mm
fillet_layup['Orientation_Sequence'] = (0., 0., 0.)  # unit in degree
fillet_layup['Auxiliary_Rotation'] = 0.              # unit in degree
......
model = input_file.create_group('Model')
feature1 = model.create_group('A_Sample_Fillet')
feature1.attrs['FeatureType'] = 'Fillet'
feature1.attrs['Layup'] = str(fillet_layup.name)
feature1['Radius'] = 4.0                            # unit in mm
feature1['Length'] = 48.0                           # unit in mm
feature1['ShellElementType'] = 'S4R'
feature1['ShellElementSize'] = 3.0                  # unit in mm
feature1['SolidElementType'] = 'C3D8R'
feature1['SolidElementSize'] = 1.0                  # unit in mm
feature1['SolidElementPerPly'] = 1
input_file.close()
```

correspond to the modules in the developed Abaqus plugin. Detailed code segments are provided for the core procedures.

4.1. Feature model generation

As introduced in Section 2.1, the feature model generation is a one-way submodelling process which is an essential part of the two-level submodelling structure. The idea of the workflow presented is to pre-compute the second level of the submodelling process, i.e. the intermediate to 3D feature model, so as to enable 3D FE results to be obtained from the global model by using the pre-computed feature database.

In this example, the straight fillet, the generated intermediate model and the 3D feature model share the same input information from the HDF5 data file. During the generation, each DOF of the driven nodes is applied with a unit value, so that the intermediate model will respond with a corresponding displacement

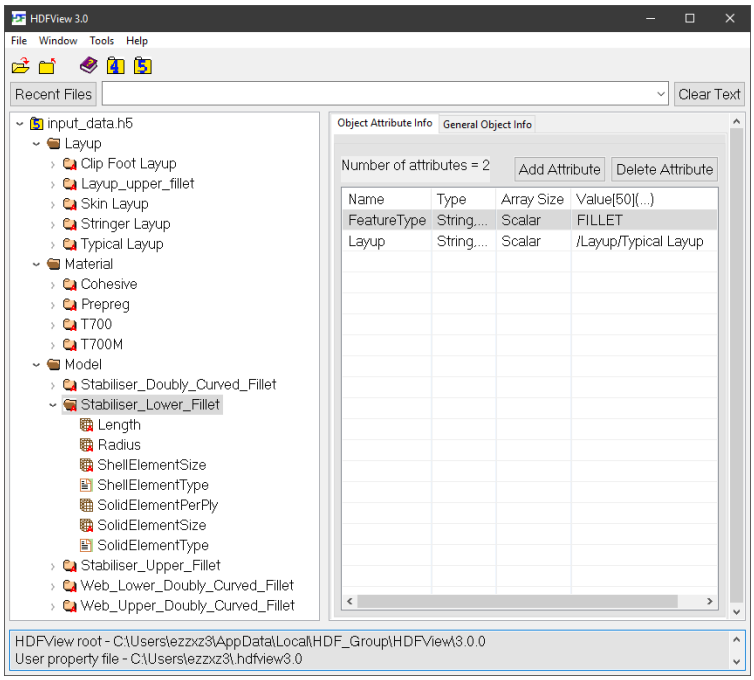

Figure 8: A sample input HDF5 file, opened with HDFView.

field which is later used to drive the 3D feature model, resulting a fictitious stress field. By storing the stress field caused by each unit DOF into the M-matrix, the intermediate and the 3D feature models are condensed into a representative feature, which can be further used to replace the second level of submodelling. To implement this procedure in Abaqus, each unit DOF is applied in an individual step of the same model, so that the results can be obtained through a one-off job submission. Obviously, the number of steps is proportional to the number of driven nodes' DOF, and the generation of the feature model database may take hours, being the most costly part of the overall workflow.

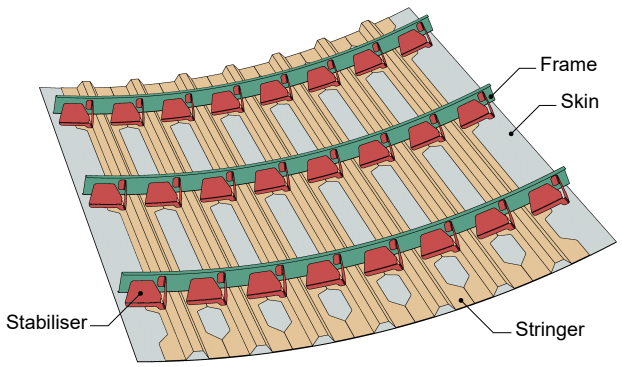

Figure 9: A typical stiffened panel as the global model used in the case study.

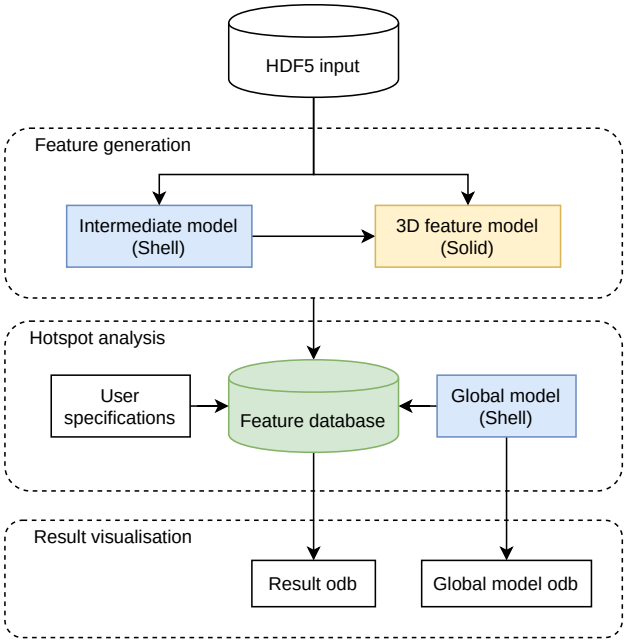

Figure 10: The flowchart of the case study.

4.2. Coordinate transformation

To facilitate the hotspot analysis, it is necessary to perform the proper coordinate transformation between the feature model and the querying feature in the global model. The coordinate system of the feature model, denoted by $\mathcal{F}_\mathrm{f}$, is constructed by

$$\mathcal{F}_\mathrm{f} = \mathrm{span}\{\boldsymbol{i}_\mathrm{f}, \boldsymbol{j}_\mathrm{f}, \boldsymbol{k}_\mathrm{f}, \mathcal{O}_\mathrm{f}\}, \tag{10}$$

where $(\boldsymbol{i}_\mathrm{f}, \boldsymbol{j}_\mathrm{f}, \boldsymbol{k}_\mathrm{f})$ are the unit vectors of the Cartesian frame, and $\mathcal{O}_\mathrm{f}$ is the origin. Similarly, the coordinate system in which the global model was constructed, denoted by $\mathcal{F}_\mathrm{g}$, reads

$$\mathcal{F}_\mathrm{g} = \mathrm{span}\{\boldsymbol{i}_\mathrm{g}, \boldsymbol{j}_\mathrm{g}, \boldsymbol{k}_\mathrm{g}, \mathcal{O}_\mathrm{g}\}. \tag{11}$$

The coordinate transformation can be denoted by an operator $\mathcal{T}$, which is an affine map:

$$\mathcal{T} : \mathcal{F}_\mathrm{g} \to \mathcal{F}_\mathrm{f}. \tag{12}$$

Numerically, the operator $\mathcal{T}$ is represented by a transformation matrix with $\dim \mathcal{T} = 4$. Therefore, to obtain this matrix, the coordinates of four non-coplanar points are required in both $\mathcal{F}_\mathrm{f}$ and $\mathcal{F}_\mathrm{g}$. These points, as already mentioned in Section 3.3.1, are called the anchor points of a certain feature.

Let $\boldsymbol{p}_i \in \mathcal{F}_\mathrm{g}$ and $\boldsymbol{q}_i \in \mathcal{F}_\mathrm{f}$, $i = 1, 2, 3, 4$, wherein exists a one-to-one registration between the global and feature models, namely $\boldsymbol{p}_i \leftrightarrow \boldsymbol{q}_i$. The points can also be written in the form of coordinates as

$$\begin{aligned}
\boldsymbol{p}_i &= p_{i1}\boldsymbol{i}_\mathrm{g} + p_{i2}\boldsymbol{j}_\mathrm{g} + p_{i3}\boldsymbol{k}_\mathrm{g} + \mathcal{O}_\mathrm{g} = \begin{bmatrix} p_{i1} & p_{i2} & p_{i3} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{i}_\mathrm{f} & \boldsymbol{j}_\mathrm{f} & \boldsymbol{k}_\mathrm{f} & \mathcal{O}_\mathrm{f} \end{bmatrix}^\mathrm{T}, \quad i = 1, 2, 3, 4, \\
\boldsymbol{q}_i &= q_{i1}\boldsymbol{i}_\mathrm{f} + q_{i2}\boldsymbol{j}_\mathrm{f} + q_{i3}\boldsymbol{k}_\mathrm{f} + \mathcal{O}_\mathrm{f} = \begin{bmatrix} q_{i1} & q_{i2} & q_{i3} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{i}_\mathrm{f} & \boldsymbol{j}_\mathrm{f} & \boldsymbol{k}_\mathrm{f} & \mathcal{O}_\mathrm{f} \end{bmatrix}^\mathrm{T}, \quad i = 1, 2, 3, 4.
\end{aligned} \tag{13}$$

The coordinate transformation can be facilitated via the matrix form:

$$\boldsymbol{Q} = \mathcal{T}\boldsymbol{P}, \tag{14}$$

where $\boldsymbol{P}$ is formed by p_{ij} with $i, j = 1, 2, 3, 4$ and $p_{i4} = 1$, and $\boldsymbol{Q}$ follows the same rule. Therefore, given the coordinates of the anchor points in both global and feature coordinate systems, the transformation matrix $\mathcal{T}$ can be obtained by solving (14). In practice, this is implemented using NumPy's linear system solver as shown in Listing 4. An additional function `check_for_coplanar` is implemented to check the coordinates of the input nodes to ensure they are non-coplanar. In some circumstances, there may exist some small

Listing 4 Sample code segment for coordinate transformation including a priori checking of non-coplanarity.

```
import numpy as np
P = np.c_[globalCoords, np.ones((4, 1))]
Q = np.c_[localCoords, np.ones((4, 1))]
print 'Checking if global nodes are coplanar...'
check_for_coplanar(P)
print 'Checking if local nodes are coplanar...'
check_for_coplanar(Q)
T = np.linalg.solve(P, Q)
```

discrepancies between the feature model and the real feature of interest in the global model, since the design of the latter is likely to be subject to change within the design iteration process. In this case, the iterative closest point method [31, 32] can be used to obtain an optimal transformation matrix.

Having obtained the coordinate transformation matrix $\mathcal{T}$, it is straightforward to extract the first 3×3 elements to form the displacement transformation matrix $\boldsymbol{T}$. Subsequently, the displacement $\boldsymbol{u}$ and stress $\boldsymbol{\sigma}$ can be transformed by

$$\boldsymbol{u}_{\mathrm{f}} = \boldsymbol{T}\boldsymbol{u}_{\mathrm{g}}, \quad \boldsymbol{\sigma}_{\mathrm{f}} = \boldsymbol{T}\boldsymbol{\sigma}_{\mathrm{g}}\boldsymbol{T}^{\mathrm{T}}. \tag{15}$$

Alternatively, in the case that a user-defined feature model is provided in an Abaqus input (`.inp`) file with known transformation parameters, the transformation matrix can be calculated directly. In particular, the instructions for the transformation of a part instance in the Abaqus input file reads:

```
*Instance, name=Fillet-instance, part=Fillet-part
v_x, v_y, v_z
a_x, a_y, a_z, b_x, b_y, b_z, theta
*End Instance
```

where $\boldsymbol{v} = (v_x, v_y, v_z)^{\mathrm{T}}$ is a vector along which the instance to translate, spatial points $\boldsymbol{a} = (a_x, a_y, a_z)^{\mathrm{T}}$ and $\boldsymbol{b} = (b_x, b_y, b_z)^{\mathrm{T}}$ defines an axis around which the translated instance will rotate by angle value θ. For

instance, a point $\boldsymbol{p}$ in the local coordinate system is transformed to the global coordinate system to be point $\boldsymbol{q}$ by

$$\boldsymbol{q} = \boldsymbol{R}(\boldsymbol{p} + \boldsymbol{v}), \tag{16}$$

where the rotation matrix $\boldsymbol{R}$ is determined by

$$\boldsymbol{R} := (\cos\theta)\boldsymbol{I} + (\sin\theta)[\boldsymbol{b} - \boldsymbol{a}]_\times + (1 - \cos\theta)(\boldsymbol{b} - \boldsymbol{a}) \otimes (\boldsymbol{b} - \boldsymbol{a}), \tag{17}$$

with $\boldsymbol{I}$ being the identity matrix and $[\cdot]_\times$ being the cross product matrix:

$$[\boldsymbol{a}]_\times := \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}. \tag{18}$$

4.3. Hotspot identification

Hotspots in the feature models are identified based on the post-processing of the obtained 3D stress fields by using the failure criteria introduced earlier for composite materials. In this case study, we use the straight fillet feature of the stabilisers in the stiffened panel to demonstrate the proposed workflow. The fillets are referred to as the regions of interest (ROI) as shown in Figure 11. The stabilisers are made of woven fabrics so that the maximum stress criteria are applied to the warp and weft yarn directions only to evaluate material damage.

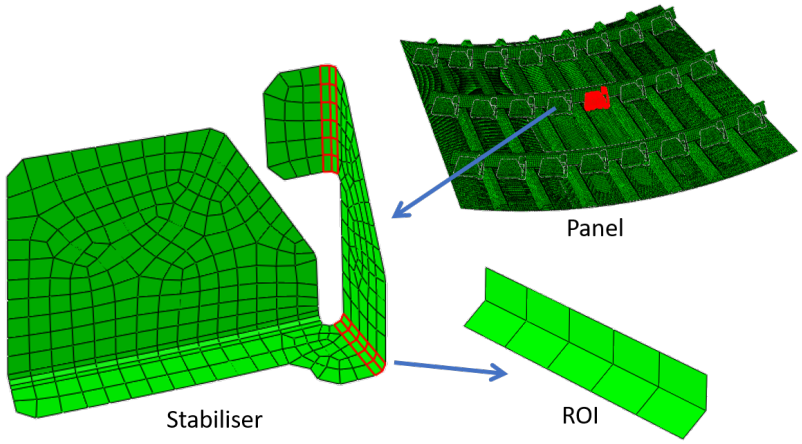

Figure 11: Straight fillet features as region of interest for hotspot identification in the stiffened panel.

Having extracted displacement values in the vicinity of the driven nodes of the intermediate model, and interpolated the displacements to these driven nodes, a displacement vector $\mathbf{U}_\mathrm{d}$ is first formed for a specific feature instance. The resulting stress field is then obtained by multiplying the M-matrix defined by Equation 6 with the displacement vector. The nodal stresses can be obtained in a similar way after

extrapolating the stresses from integration points to nodes to form an M-matrix for nodal stresses. It is noted that global to local coordinate transformation was performed for $\mathbf{U}_{\mathrm{d}}$ before the multiplication as the M-matrix and feature model are defined under the local coordinate system. As an example, Figure 12 shows the results for the 3D stress recovery in comparison with the global model and intermediate model for a straight fillet instance as well as a delamination damage contour. The contours for 3D model are retrospectively created by adding the obtained damage parameters (in this case warp damage, weft damage and delamination onset) or recovered stress field if required as field variables to the 3D feature mesh.

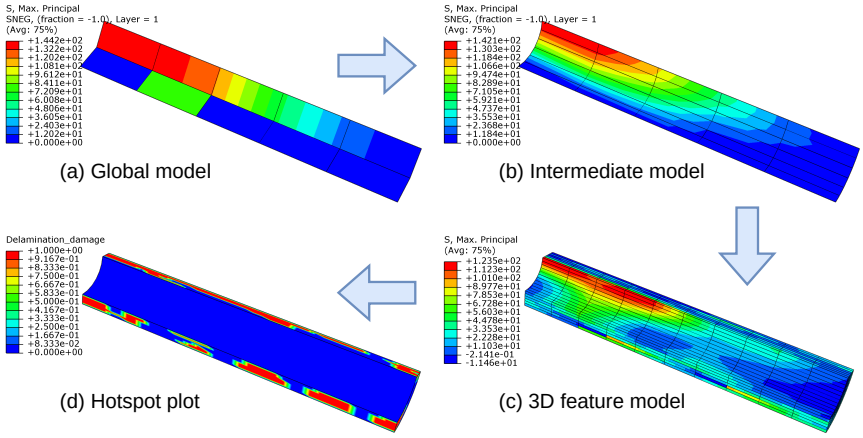

Figure 12: 3D Stresses recovery from global model (a), through intermediate model (b), and 3D feature model (c). The hotspot (delamination) contour (d), a fictitious plot (not corresponding to presented stresses before) for illustration purpose, in a typical fillet feature instance.

To assist the users of this software to evaluate the results in the presence of the global model, the functionality for the overlay plot of the 2D global model and the 3D feature result output database is provided and an example is shown in Figure 13, in which the results of two types of straight fillet, i.e. upper and lower fillets in the stabilisers are present. It should be noted that coordinate transformation from local to global system is required for the 3D feature results to ensure all of models in the visualisation are under the global coordinate system. Here this coordinate transformation is performed by using the inverse of $\boldsymbol{T}$ as in Equation 15. Alternatively, the coordinate transformation can also utilise Equation 16 directly, provided the Abaqus input file is available.

4.4. Discussion

In this case study, the process of 3D stress recovery and hotspot calculation for each straight fillet instance under a specific load increment only takes approximately 10 seconds on a desktop PC. As the example stiffened panel has 24 instances for the lower and upper straight fillet features respectively, the computational acceleration is very significant in comparison with performing direct submodelling analysis

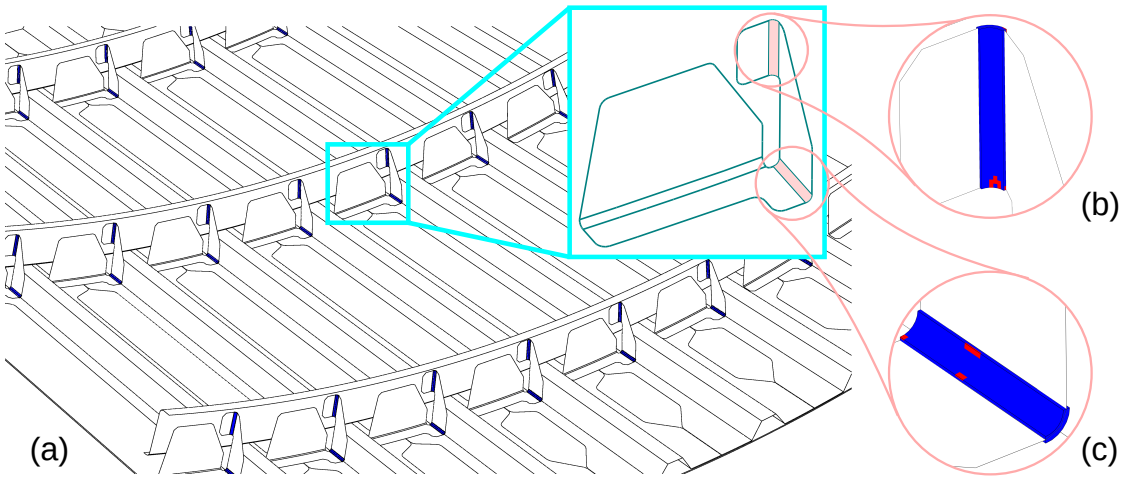

Figure 13: Visualisation of the results: (a) Overview of 3D feature models and the global model through overlaid plot; (b) 3D feature model of a upper fillet with warp damage; (c) 3D feature model of a lower fillet with weft damage.

for every feature instance. This case study has shown that the presented software is a very efficient tool for analysing the critical regions in large-scale composite airframes with repeated features. Without loss of generality, the users can develop their own feature models with regard to the global model they would like to analyse. For example, for analysing the stiffened panel present in this study, we have also developed parameterised feature generation codes for the curved fillet (of the stabiliser) and bonded joint (joining the skin and the stringer) although they have not been demonstrated in this paper. As the aim of this paper is focused on the development of the software, the error quantification part [24] of the workflow (another aspect of the MARQUESS project) is not discussed here. Also, this workflow is based on the one-way submodelling theory so that it could not consider the progressive damage behaviour caused by material property degradation, as its aim is to evaluate the damage initiation and filter hotspots in large composite structures at the design stage.

5. Conclusions

During the iterative design process of large-scale composite structures, a significant challenge for identifying the hotspots for damage initiation is the high modelling cost, which it is imperative to address especially for aerospace applications whilst their experimental testing is more costly. This work presents an efficient Abaqus plugin tool for the structural design stage to identify the hotspots in large-scale composite structures with repeated features, as usually seen in airframes. The software implements a framework using a pre-computed database generated for specific feature types to perform a fast recovery of 3D stress states, based on which failure criteria can be readily applied. Developed with an object-oriented implementation in Python, this software delivers full 3D results for the features of interest via a displacement driven pre-

computed submodel to the coarse structure model. The software architecture and general workflow were discussed in detail in the paper with essential code segments available for other researchers to reuse. The presented software is an analysis tool for efficient sizing of large-scale composite structures, as it has eliminated the conventional tedious feature by feature deep dive and enables 3D damage analysis of the structures in critical zones with significant savings of the modelling and computational cost. The recovered stresses were compared with conventional FE modelling and satisfactory agreement was observed. A further 3D stress error quantification package will also be integrated in the software in the future.

Research data

The software introduced in this paper (MARQUESS Version 1.0) with supporting materials is open access under MIT License and available at: https://doi.org/10.5281/zenodo.3967803.

Acknowledgements

This work is funded by the Clean Sky 2 Joint Undertaking under the European Union's Horizon 2020 research and innovation programme under grant agreement No 754581. Software engineering support was funded by the Engineering and Physical Sciences Research Council RSE Fellowship [Grant number: EP/N019040/1].

References

[1] H. Molker, R. Gutkin, S. Pinho, L. E. Asp, Hot spot analysis in complex composite material structures, Composite Structures 207 (2019) 776–786. doi:10.1016/j.compstruct.2018.09.088.

[2] S. Hühne, J. Reinoso, E. Jansen, R. Rolfes, A two-way loose coupling procedure for investigating the buckling and damage behaviour of stiffened composite panels, Composite Structures 136 (2016) 513–525. doi:https://doi.org/10.1016/j.compstruct.2015.09.056.

[3] M. Akterskaia, E. Jansen, S. Hallett, P. Weaver, R. Rolfes, Analysis of skin-stringer debonding in composite panels through a two-way global-local method, Composite Structures (2018). doi:10.1016/j.compstruct.2018.06.064.

[4] M. Akterskaia, E. Jansen, S. Hühne, R. Rolfes, Efficient progressive failure analysis of multi-stringer stiffened composite panels through a two-way loose coupling global-local approach, Composite Structures 183 (2018) 137–145. doi:https://doi.org/10.1016/j.compstruct.2017.02.011.

[5] M. Akterskaia, E. Jansen, S. R. Hallett, P. M. Weaver, R. Rolfes, Progressive failure analysis using global-local coupling including intralaminar failure and debonding, AIAA Journal (2019) 3078–3089.

[6] S. Yan, X. Zou, M. Ilkhani, A. Jones, An efficient multiscale surrogate modelling framework for composite materials considering progressive damage based on artificial neural networks, Composites Part B: Engineering 194 (2020) 1–11. doi:10.1016/j.compositesb.2020.108014.

[7] D. W. Spring, G. H. Paulino, A growing library of three-dimensional cohesive elements for use in ABAQUS, Engineering Fracture Mechanics 126 (2014) 190–216.

[8] P. Zhang, X. Hu, X. Wang, W. Yao, An iteration scheme for phase field model for cohesive fracture and its implementation in Abaqus, Engineering Fracture Mechanics 204 (2018) 268–287.

[9] E. Giner, N. Sukumar, J. Tarancón, F. Fuenmayor, An Abaqus implementation of the extended finite element method, Engineering fracture mechanics 76 (3) (2009) 347–368.

[10] Y. Lai, Y. J. Zhang, L. Liu, X. Wei, E. Fang, J. Lua, Integrating CAD with Abaqus: a practical isogeometric analysis software platform for industrial applications, Computers & Mathematics with Applications 74 (7) (2017) 1648–1660.

[11] S. Sarkar, I. Singh, B. Mishra, A. Shedbale, L. Poh, A comparative study and ABAQUS implementation of conventional and localizing gradient enhanced damage models, Finite Elements in Analysis and Design 160 (2019) 1–31.

[12] W. Zhang, E. E. Seylabi, E. Taciroglu, An ABAQUS toolbox for soil-structure interaction analysis, Computers and Geotechnics 114 (2019) 103143.

[13] Y.-J. Liang, J. S. McQuien, E. V. Iarve, Implementation of the regularized extended finite element method in Abaqus framework for fracture modeling in laminated composites, Engineering Fracture Mechanics (2020) 106989.

[14] B. Helldörfer, M. Haas, G. Kuhn, Automatic coupling of a boundary element code with a commercial finite element system, Advances in Engineering Software 39 (8) (2008) 699–709. doi:10.1016/j.advengsoft.2007.07.003.

[15] G. Rio, H. Laurent, G. Blès, Asynchronous interface between a finite element commercial software ABAQUS and an academic research code HEREZH++, Advances in Engineering Software 39 (12) (2008) 1010–1022. doi:10.1016/j.advengsoft.2008.01.004.

[16] M. Nesládek, M. Španiel, An Abaqus plugin for fatigue predictions, Advances in Engineering Software 103 (2017) 1–11. doi:10.1016/j.advengsoft.2016.10.008.

[17] S. N. Ullah, L. F. Hou, U. Satchithananthan, Z. Chen, H. Gu, A 3D RITSS approach for total stress and coupled-flow large deformation problems using ABAQUS, Computers and Geotechnics 99 (2018) 203–215.

[18] S. Li, L. Jeanmeure, Q. Pan, A composite material characterisation tool: Unitcells, Journal of Engineering Mathematics 95 (1) (2015) 279–293.

[19] A. MacKrell, Multiscale composite analysis in Abaqus: Theory and motivations, Reinforced Plastics 61 (3) (2017) 153–156.

[20] S. L. Omairey, P. D. Dunning, S. Sriramula, Development of an ABAQUS plugin tool for periodic RVE homogenisation, Engineering with Computers 35 (2) (2019) 567–577. doi:10.1007/s00366-018-0616-4.

[21] X. Zou, M. Conti, P. Díez, F. Auricchio, A nonintrusive proper generalized decomposition scheme with application in biomechanics, International Journal for Numerical Methods in Engineering 113 (2) (2017) 230–251. doi:10.1002/nme.5610.

[22] J. Mackerle, Object-oriented programming in FEM and BEM: a bibliography (1990–2003), Advances in Engineering Software 35 (6) (2004) 325–336. doi:10.1016/j.advengsoft.2004.04.006.

[23] D. Cojocaru, A. M. Karlsson, An object-oriented approach for modeling and simulation of crack growth in cyclically loaded structures, Advances in Engineering Software 39 (12) (2008) 995–1009. doi:10.1016/j.advengsoft.2008.01.009.

[24] M. Bonney, R. Evans, J. Rouse, A. Jones, M. Hamadi, Bayesian reconstruction of goal orientated error fields in large aerospace finite element models, in: Aerospace Europe Conference, Bordeaux, France, 2020.

[25] Dassault Systèmes, Abaqus 2016 Analysis User's Guide (2015).

[26] A. S. Verma, N. P. Vedvik, P. U. Haselbach, Z. Gao, Z. Jiang, Comparison of numerical modelling techniques for impact investigation on a wind turbine blade, Composite Structures 209 (2019) 856–878.

[27] A. Collette, Python and HDF5: Unlocking Scientific Data, O'Reilly Media, Inc., 2013.

[28] Fox toolkit, http://www.fox-toolkit.org/, accessed: 2020-05-01.

[29] M. Folk, G. Heber, Q. Koziol, E. Pourmal, D. Robinson, An overview of the hdf5 technology suite and its applications, in: Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases, 2011, pp. 36–47.

[30] HDF® View - The HDF Group, https://www.hdfgroup.org/downloads/hdfview/, accessed: 2020-05-01.

[31] Y. Chen, G. Medioni, Object modelling by registration of multiple range images, Image and Vision Computing 10 (3) (1992) 145–155, range Image Understanding. doi:10.1016/0262-8856(92)90066-c.

[32] P. J. Besl, N. D. McKay, A method for registration of 3-D shapes, IEEE Transactions on Pattern Analysis and Machine Intelligence 14 (2) (1992) 239–256.