Thermomechanical modeling of copper-ceramic bonded assemblies for power electronics applications

Théophile Hourdou

Thermomechanical modeling of copper-ceramic bonded assemblies for power electronics applications

Théophile Hourdou[a]
[a]Univ. Grenoble Alpes, CNRS, Grenoble INP, SIMaP, 38000 Grenoble, France

August 30, 2024

This work is supported by the French National Research Agency in the framework of the "Investissements d'avenir" program (ANR-15-IDEX-02)

Abstract: This study examines the thermo-mechanical behaviour and reliability of Direct Bonded Copper (DBC) component in power electronics. Using finite element simulations and experimental data, material properties and interface behaviour has been modelled. Thermal cycling simulation has been done to identify key failure modes, focusing on stress distribution and crack propagation at the copper-ceramic interface. The findings offer crucial insights for improving the design and durability of DBC components in demanding applications.

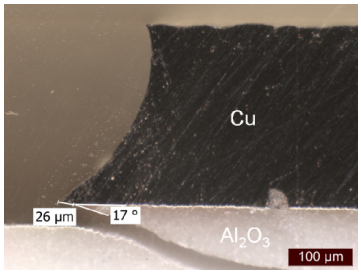
1 Introduction

In the field of power electronics, the demand for highly reliable and efficient thermal management solutions has significantly increased, driven by the rapid development of high-power density applications such as electric vehicles, renewable energy systems, and advanced computing technologies. Direct Bonded Copper (DBC) (Figure 1) components have emerged as a critical one in these applications, offering a combination of high thermal conductivity and electrical insulation in packaging devices.

(a) (b)

Figure 1: Global (a) and sectional view (b) of a power electronics device [Ben Kaabar, 2015]

However, the industrial adoption of DBC components faces a critical challenge due to their susceptibility to failure under thermal cycling conditions. The mismatch in the coefficients of thermal expansion (CTE) between the copper and ceramic layers introduces significant thermo-mechanical stresses during operation, which can lead to the degradation of the material interfaces and ultimately result in device failure.



(a) crack propagation in Al_2O_3 ceramic

(b) ceramic-copper interface decohesion

Figure 2: Failures in a DBC component [Gaiser et al., 2015]

Figure 2 shows the two primary failure mechanisms of the DBC component. A common one is crack initiation and propagation in the underneath ceramic. However, interface decohesion, where the bond between the copper and ceramic layers weakens and separates, is the more prevalent failure mode.

1.1 Objectives

Understanding the behaviour of materials and their interfaces under these conditions is paramount. This study is focused on modelling approaches to predict the main failure mechanisms being ceramic-copper interface decohesion. This involves not only the precise mechanical characterization of the individual materials but also the development of fine numerical models that can simulate the complex interactions between the copper, ceramic, and their interfaces. The outcomes of this study are expected to contribute to the advancement of power electronics with more reliable DBC components.

1.2 Methodology

This study uses experimental data from [Ben Kaabar, 2015] to model the thermo-mechanical behaviour of DBC components and interface. Two methodologies has been used to analyze the behaviour and reliability of the DBC component.

First, a simulation of the DBC component cooling during assembly is conducted to quickly analyse the interactions between the materials under operational conditions. The study then focuses on evaluating interface failure with the coupled criterion, using the analysis of stress levels along the interface and the energy release rate associated with crack propagation, providing insights into the conditions under which cracks may initiate at the interface.

Secondly, a simulation of the DBC component during assembly and thermal cycling during operation is conducted, using the thermo-mechanical non-linear behaviours of the materials and the failure behaviour of the interface. The study then focuses on evaluating interface damage and failure.

2 DBC components behaviour modeling

For both methodologies, thermo-mechanical model behaviour parameters must be identified at first from experimental data. The global behaviour of bulk components is based on the total strain $\underline{\underline{\varepsilon}}$ additive decomposition:

$$\underline{\underline{\varepsilon}} = \underline{\underline{\varepsilon}}_e + \underline{\underline{\varepsilon}}_p + \underline{\underline{\varepsilon}}_{th} \tag{1}$$

with subscript e the reversible (or elastic) strain part, p the irreversible (or plastic) strain part if appropriate and th the thermal strain part.

Reversible strain part is commonly modelled by isotropic linear elasticity (Hooke's law):

$$\underline{\underline{\varepsilon}}_e = \frac{1+\nu}{E}\underline{\underline{\sigma}} - \frac{\nu}{E} I_1(\underline{\underline{\sigma}})\underline{\underline{I}} \tag{2}$$

where E is the Young's modulus and ν the Poisson's ratio.

Irreversible strain part in metallic materials is commonly described by an equivalent stress σ_{eq} determined from a yield criterion and various internal variables.

The equivalent stress is defined as follows:

$$\sigma_{eq}(\sigma_*, R) = \max\{\sigma_* - R, 0\} \tag{3}$$

with σ_* an effective stress determined from the yield criterion and R an isotropic hardening. In the literature, this effective stress is determined by the von Mises yield criterion:

$$\sigma_* = J_2\left(\underline{\underline{\sigma}} - \sum_{i=1}^{n} \underline{\underline{X}}_i\right) \tag{4}$$

with $\underline{\underline{X}}$ kinematic hardenings.

Several evolution functions may be used to described both hardening, however in the present study non-linear ones are used since metallic behaviour exhibit severe non-linearities. The isotropic hardening may be defined as a function of the cumulated plastic strain p:

$$R = R_0 + Q\left(1 - \exp\left(-bp\right)\right) \tag{5}$$

with R_0 the initial yield stress, Q the hardening amplitude and b the saturation rate.

Kinematic hardenings are defined by their rate, depending on the plastic strain rate $\dot{\underline{\underline{\varepsilon}}}_p$:

$$\dot{\underline{\underline{X}}}_i = \frac{2}{3} C_i \dot{\underline{\underline{\varepsilon}}}_p - \gamma_i \underline{\underline{X}}_i \dot{p} \tag{6}$$

with C_i the hardening rate and γ_i the saturation rate.

Thermal strain part is supposed isotropic and depends linearly on the temperature:

$$\underline{\underline{\varepsilon}}_{th} = \alpha\ \Delta T\ \underline{\underline{I}} \tag{7}$$

with α the coefficient of thermal expansion (CTE).

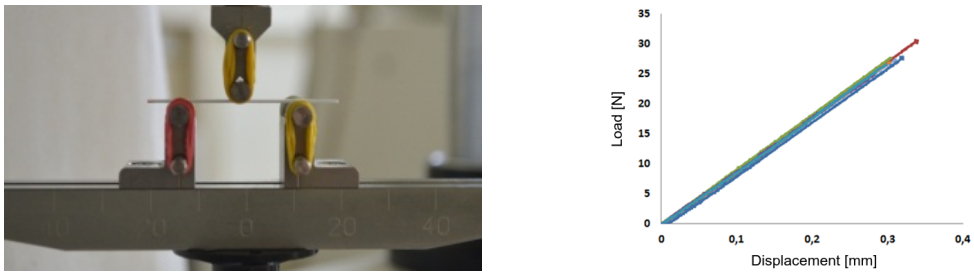
This coefficient was measured experimentally, values were used by [Ben Kaabar, 2015] and are kept as is in this study. Furthermore, an assumption is made about the temperature independence of all mechanical parameters i.e. all values are considered constant.

2.1 Ceramic thermo-mechanical modeling

Ceramic material exhibits a linear response under traction and bending as shown by previous studies [Ben Kaabar, 2015]. Experiments results have already been used to finely model its mechanical behaviour. Results and parameters are only presented here as a recall.

2.1.1 Experimental data

3-point bending were used to characterize the ceramic material. A parallelepiped-shape sample is placed on 2 lateral pins and under a third central one. Figure 3(a) shows the experimental setup. Typical results of ceramic bending, which exhibit a linear evolution of the load depending on the pins displacement up to failure, are shown in Figure 3(b).



(a) 3-point bending setup (b) typical bending results

Figure 3: Ceramic mechanical characterization

2.1.2 Model description

Since experimental results are linear, the material mechanical behaviour may be modelled by a linear elastic model presented in the previous section, in equation (2). The complete thermo-mecanical model is defined with the following material parameters:

- elasticity : E and ν
- thermal : α

2.1.3 Parameters identification

[Ben Kaabar, 2015] study used beam therory to recover E from the following equation:

$$E = \frac{F_R \, L^3}{48 \, \delta \, I_z} \tag{8}$$

with F_R the load at failure, L the length between lateral pins, δ the deflection (here supposed equal to the central pin displacement) and I_z the second moment of area of the beam's cross section depending on the section geometry.

Concerning Poisson's ratio, [Ben Kaabar, 2015] study evaluated its value by homogenization and comparison to the literature.

Final parameters values shown in Table 1 were found to modelled the ceramic material.

E [GPa]	ν [-]	α [e-6/°C]
330	0.22	8

Table 1: Ceramic thermo-mechanical model parameters

2.2 Copper thermo-mechanical modeling

Unlike ceramic in the previous section, copper exhibit a non-linear response under cyclic tensile tests as shown by previous studies [Ben Kaabar, 2015] and was modelled by an elasto-plastic mechanical model with a single cinematic non-linear hardening. In this study, the model has been re-identified to be more representative of experimental data[1].

2.2.1 Experimental data

Cyclic tensile tests were done in [Ben Kaabar, 2015] study using plate dogbone samples. 10 tensions up to a load corresponding to an engineering stress of 120MPa was imposed. Load cell were used to acquire the load and digital image correlation were used to compute longitudinal and transverse extensions in the calibrated length of the sample.

True stress and true strain were computed from load and extensions. Figure 4 shows the resulting cyclic tensile curve for this metallic material. Left side corresponds to the total test, right side shows a zoom around 5% of strain corresponding to additional cycles.

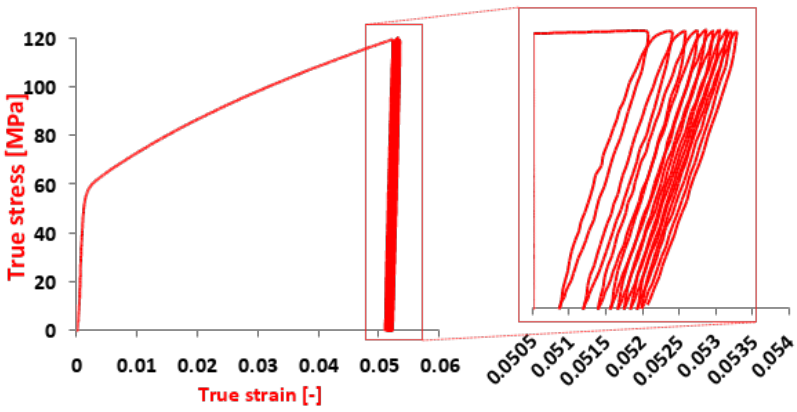


Figure 4: Cyclic tensile test data

During the first tension, the material exhibits a linear response up to ∼50MPa followed by a short severely non-linear curve (up to ∼65MPa), then followed by a relatively long non-linear curve up to 120MPa.

Following cycles show that the severe non-linear curve occurs at each tension. Moreover, this non-linearity tends to start at increasingly stress values cycle after cycle, leading to the closing of the hysteresis loops and the stabilization of the mechanical response.

[1]It may be noted that the data has been recovered graphically from [Ben Kaabar, 2015] thesis since numerical datasets had been lost.

2.2.2 Model description

Since experimental results are non-linear and exhibit reversible and irreversible strains, the material mechanical behaviour may be modelled by an elasto-plastic model presented in previous section.

Elastic part is modelled with Hooke's law in equation (2).

Plastic part is modelled with von Mises criterion and several hardening functions. Two kinematic hardenings are chosen since a short and severe non-linearity occurs at each cycle and a relatively long one is observed in the first one. Moreover, an isotropic hardening is chosen to represent the stress value increase at each cycle.

The complete thermo-mecanical model is defined with the following material parameters:

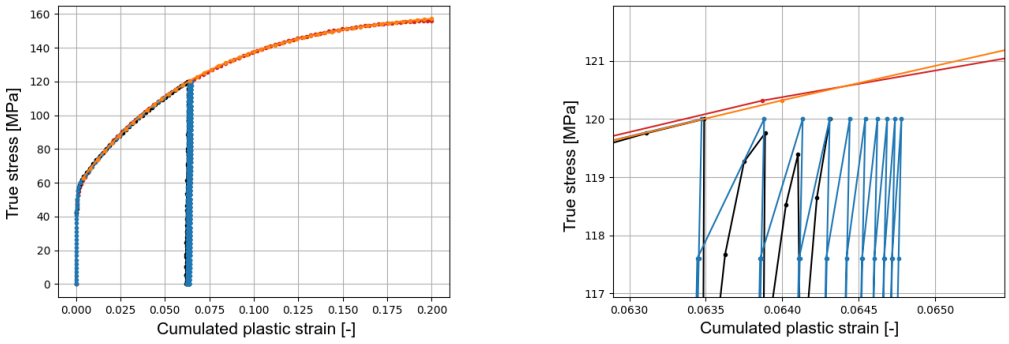
- elasticity: E and ν
- plasticity: R_0, C_1 and γ_1, C_2 and γ_2, Q and b
- thermal: α

2.2.3 Parameters identification

Identification of parameters were done directly from the cyclic tensile curve.

Initial elastic slope allow determination of E. The plastic strain is then determined by removing the elastic part: $\varepsilon_p = \varepsilon - \sigma/E$. From the chart σ *vs.* ε_p, initial yield stress and hardening parameters are roughly evaluated.

An optimization by inverse method is used afterwards to refine the relevance of these parameters. It consists of a minimization of the deviation between experimental and simulated data. Identification results are shown in Figure 5. An excellent agreement between experimental data and numerical results can be seen.



(a) up to failure (b) zoom of additional cycles

Figure 5: Copper cyclic tensile curve

Poisson's ratio and CTE are determined from literature.

Final parameters values shown in Table 2 were found to modelled the copper material.

E [GPa]	ν [-]	α [-/°C]	R_0 [MPa]	C_1 [MPa]	γ_1 [-]	C_2 [MPa]	γ_2 [-]	Q [MPa]	b [-]
127	0.33	16.5e-6	43	27000	1850	1120	15	30	12.8

Table 2: Copper thermo-mechanical model parameters

2.3 Interface mechanical modeling

Last part of the BDC component is the interface between ceramic and copper layers. Interfaces must be modelled differently from bulk materials with an approach linked to fracture mechanics.

When the interface is subjected to relatively low stress values, neither crack occurs nor decohesion propagation. The interface simply represents a discontinuity between to homogeneous sections.

With higher stress levels, the interface begins to open and a degradation of its mechanical properties occurs, at the decohesion front, prior to crack opening. This behaviour is well modelled by the cohesive zone method with a common and simple bilinear one of [Dugdale, 1960] shown in Figure 6.

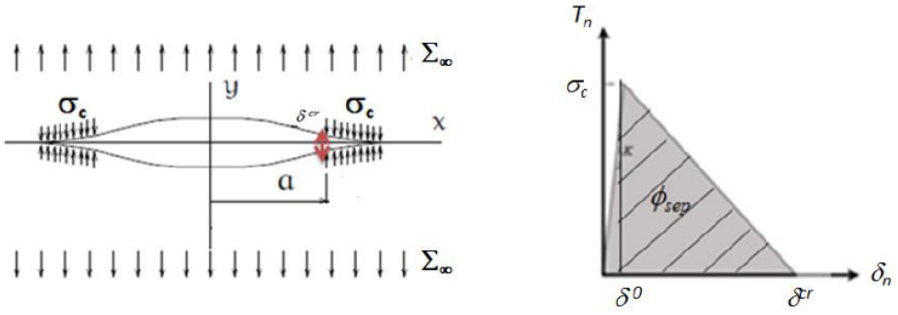


Figure 6: Cohesive zone method [Ben Kaabar, 2015]

In Figure 6 left side, a crack inside a plate under tension is shown. The half-crack length is noted a from its symmetry axis. In this area, crack lips are stress free. Further, a stress field is however visible along a short length even if the crack seems to open in this region, meaning a separation behaviour with non-zero rigidity.

In Figure 6 right side, the proposed separation behaviour is shown with normal stress vector component T_n *vs.* crack opening δ_n. Two stress evolution phases occur during opening. First, the crack is allowed to slightly open reversibly up to a maximum stress value σ_c with a rigidity $k = \sigma_c/\delta_0$. Secondly, damage initiation occurs, and increase, up to a maximum opening at failure δ_{cr}. During this second phase, the material weakens and the stress decrease linearly. The energy used to open the crack Φ_{sep} is simply the integration of the two phases on the figure, also named G_c as the energy release rate.

2.3.1 Experimental data

In order to characterized interface decohesion in DBC component between ceramic and copper layers, 4-point bending tests of these components were done in [Ben Kaabar, 2015] study. A parallelepiped sample with a central layer of ceramic and two layers of copper were bent between 2 inner bottom pins and 2 outer top pins. A notch is machined in the upper copper layer to ensure ceramic crack between inner pins. As seen on Figure 7, the displacement ΔL of the outer ones is imposed and the reaction load F is measure on the inner ones.

Figure 7: DBC bending test

The test is instrumented with an optical camera in order to observe the decohesion propagation during bending. Figure 8 shows the load *vs.* displacement with optical data at specific time during the test. Two phases are visible. A first one corresponding to the bending of the undamaged structure up to point A where the ceramic layer break inside the notch region. At this point, decohesion initiation occurs between ceramic layer and the underneath copper one. Phase B corresponds to the propagation of the decohesion in both direction. However for this specific test, it has been seen a longer decohesion at the right side leading to a significant di-symmetry.

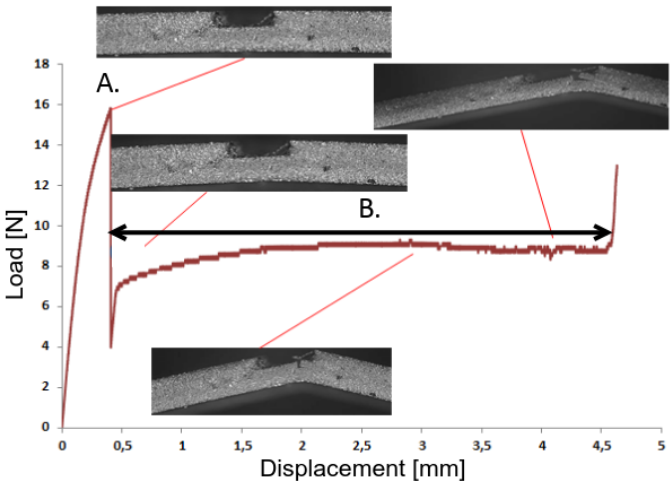


Figure 8: DBC bending test results

A processing of the optical data were done by [Ben Kaabar, 2015] on this right side to evaluate the decohesion length. However an additional analysis were done in the present study to compare left and right propagation rates. The results are shown in the following sections.

2.3.2 Model description

The [Dugdale, 1960] model is well defined for interface behaviour during decohesion between two different materials. Therefore the complete mecanical model is defined with the following interface parameters based on Figure 6:

- rigidity k
- maximum stress σ_c
- energy release rate at failure G_c

2.3.3 Parameters identification

An optimization by inverse method is used to identify these interface model parameters. It consists of a minimization of the deviation between experimental and simulated data. A simulation of the 4-point bending test was done by finite-element method using Simulia Abaqus software. The simulation input script is available in appendix A. The structure was meshed and a partial view, including the initial crack in the ceramic layer and mesh refinement along interface with underneath copper layer, can be seen in Figure 9.

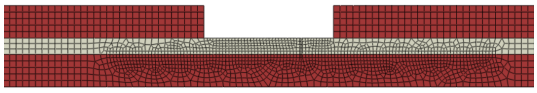


Figure 9: Simulation mesh of the DBC bending test

Identification results are shown in Figure 10 with a comparison of load over sample width *vs.* pin displacement with left and right decohesion length as well. An excellent agreement between experimental data and numerical results can be seen.

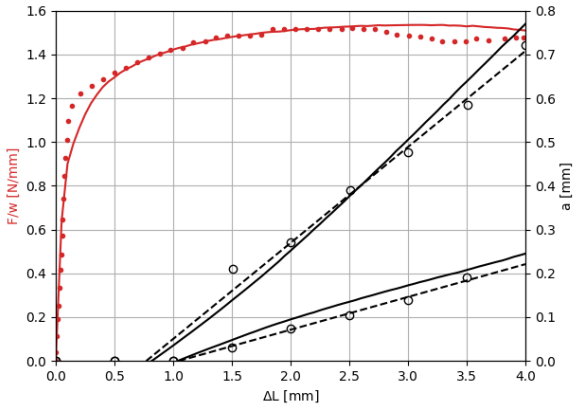


Figure 10: Interface identification results

In order to model well the di-symmetry, left and right interfaces have been defined with different parameters values. Results can be seen in Table 3. The small difference in these values is enough to generate such a propagation di-symmetry.

	k [N/mm]	σ_c [MPa]	G_c [N/mm]
left	1e8	160	0.120
right	1e8	150	0.115

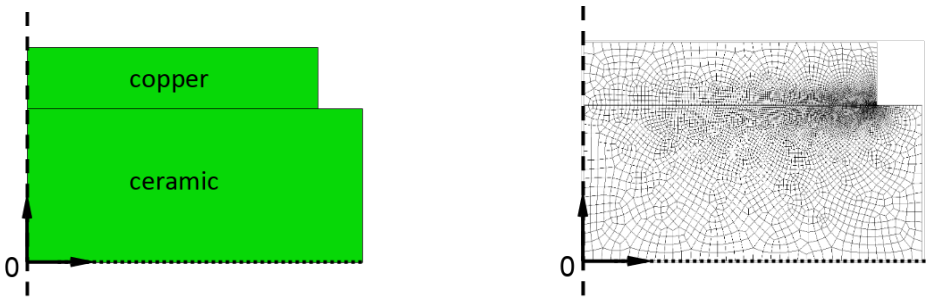
Table 3: Interface mechanical model parameters

3 Simulation of axi-symmetric DBC cooling

After material modelling, the structure can be modelled and simulated as well. In this study, finite element method was used to simulate temperature cooling step during DBC assembly which may involve interface decohesion due to CTE mismatch between ceramic and copper layers.

3.1 Finite element model

A finite element model was developed with simplified geometry and material behaviours. In this model a local part of the DBC structure is considered using Simulia Abaqus software. The simulation input script is available in appendix B.



(a) (b)

Figure 11: Geometry (a) and mesh (b) of the DBC structure

Figure 11 shows the structure with symmetry assumption and the two used materials. Kinematic boundary conditions are imposed:

1. axi-symmetry along left dashed line - $\mathsf{U1} = 0$
2. vertical symmetry along bottom dotted line - $\mathsf{U2} = 0$

Temperature field is imposed homogeneous in the structure, decreasing from 1000°C to 0°C.

Copper and ceramic are modelled as linear elastic materials. Only Young's modulus, Poisson's ratio and CTE are defined in the model. This model is indeed a first evaluation of interface decohesion since a quick and simple determination of stress levels and energy release rate is made, as defined by fracture mechanics.

Stress levels at the end of cooling are directly evaluated at interface nodes. The stress

field is extrapolated at nodes and the stress vector is computed:

$$\underline{T} = \underline{\underline{\sigma}}\,\underline{n} \tag{9}$$

with $\underline{n}$ the norm of the interface at the node. Since the interface is oriented along the X-axis, the stress vector is directly known from the stress tensor components.

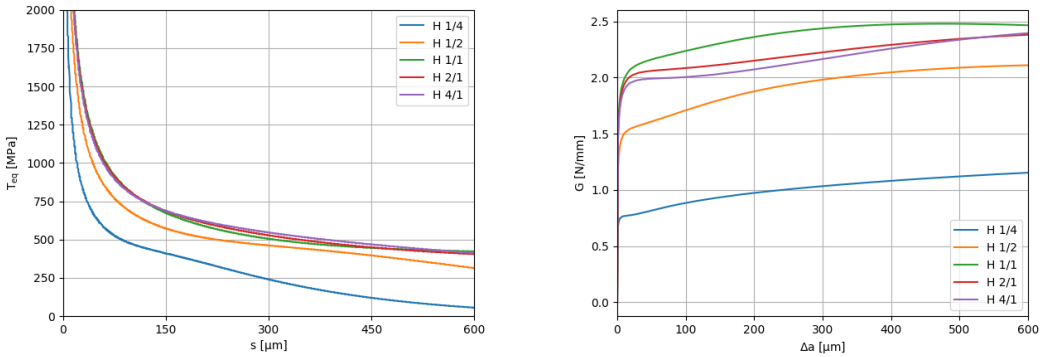
However energy release rate needs additional computation, here by the node release method. The interface nodes are duplicated and linked with kinematic equations. Virtual propagation steps are simulated by deactivating these equations node after node, starting from the right corner toward the axi-symmetry axis. The global strain energy of the structure W_e is computed and extracted for each propagation step. The energy release rate G is then defined as:

$$G(a) = \frac{W_e(a) - W_e(a=0)}{S(a)} \tag{10}$$

with a the length of the crack newly created and S the surface of the crack.

3.2 Results

Simulations of several geometries were done to evaluate their impact on $\underline{T}$ and G. At first, copper thickness, corresponding to 1/4 the ceramic thickness initially, was multiplied by a factor of x2, x4, x8 and x16. The results are shown in Figure 12.



(a) Stress levels

(b) Energy release rate

Figure 12: Axi-symmetric simulation results with copper thickness variation

From Figure 12(a), it may be detrimental to increase copper thickness since the stress levels increase. However from Figure 12(b), G levels increase up to a thickness ratio of 1/1 and then decrease. Therefore, this configuration where identical ceramic and copper layer must not be used.

A second morphology evaluation was done. Copper layer can be machined to add or remove volume of material. Holes (also called dimples) are created by removing copper near the corner. Therefore 4 dimples are meshed and are shown in Figure 13.

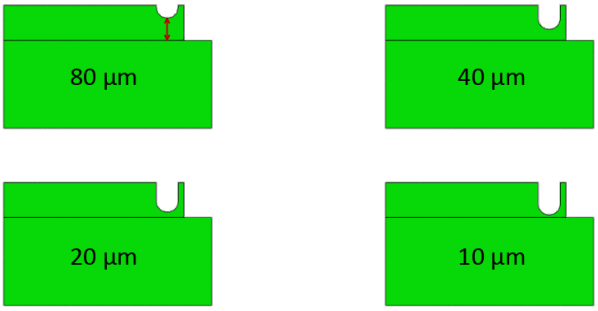
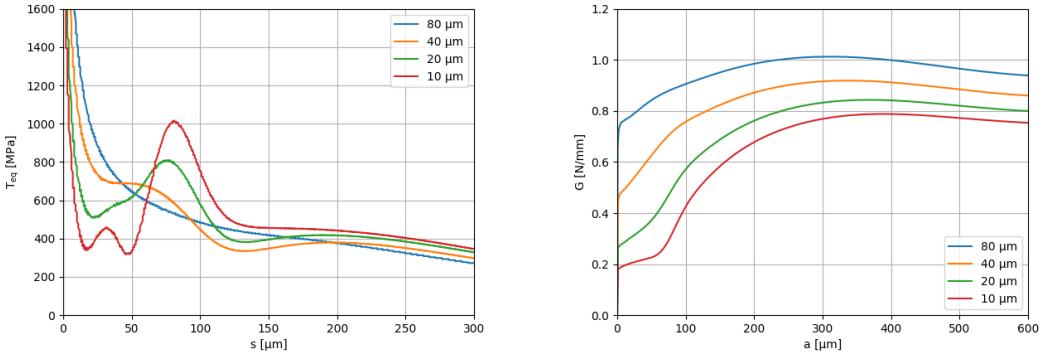


Figure 13: Dimple geometries

Stress levels and energy release rate results are shown in Figure 14. Creating a deeper hole leads to reduce stress levels, however they increase significantly after the hole. The most import result is shown in Figure 14(b) where this dimple highly reduce levels of G globally.



(a) Stress levels

(b) Energy release rate

Figure 14: Axi-symmetric simulation results with dimples

Additional morphology variations may be roughly evaluated based on this study. It has shown its capability to evaluate this criticity using a simple finite element simulation of an axi-symmetric structure with elastic components.

4 Simulation of 3D DBC cooling

To evaluate the decohesion propagation in a finer way, the model has to be more developed. Here the non-linear behaviour of the copper is used and an actual interface is modelled with cohesive zone method in the finite element calculation.

4.1 Finite element model

The structure is here 3D as shown in Figure 15. Symmetry plane at the structure back left, back right and bottom are taken into account.

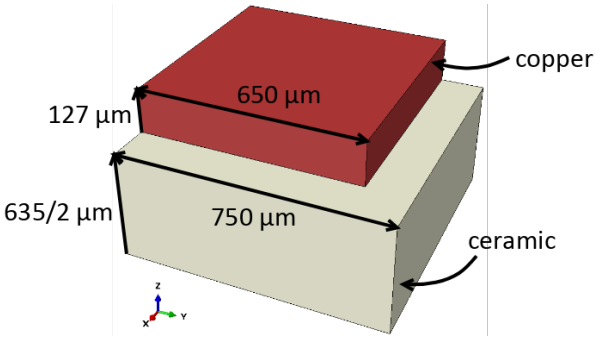


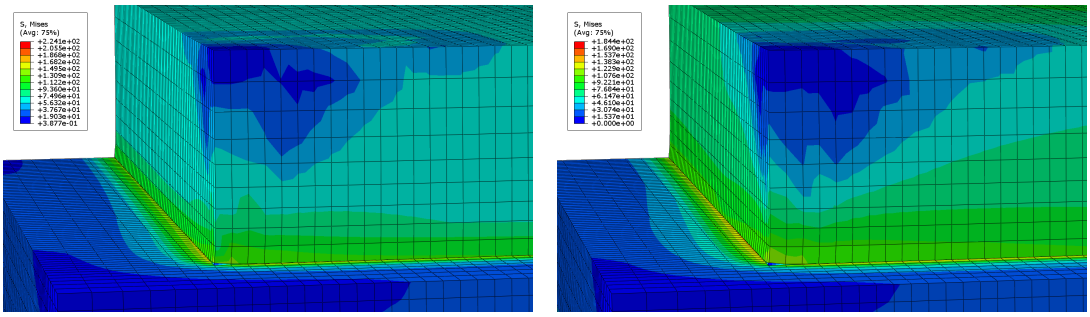
Figure 15: Geometry of the DBC structure

Symmetry boundary conditions are imposed and temperature field is imposed homogeneous in the structure, decreasing from 1000°C to 0°C. 20 heat-cooling cycles are simulated afterwards from -50°C to 250°C. The simulation was created using Simulia Abaqus software. Its input script is available in appendix C.

4.2 Results

Using cohesive zone model, decohesion is supposed to initiate and propagate during thermal cycles.

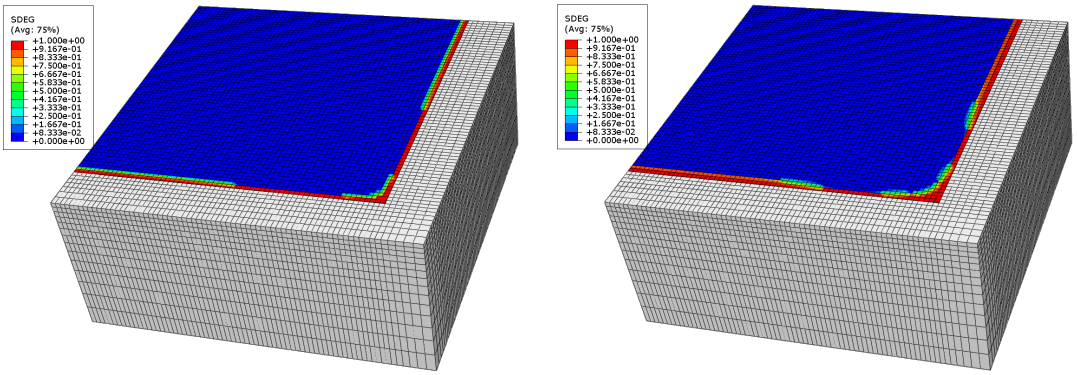
Figure 16 shows equivalent stress field after cooling (a) and after 20 cycles (b). The field is displayed on the deformed configuration. Stress level significantly decrease after cycles but no re-localisation can be clearly seen.



(a) After cooling (b) After 20 cycles

Figure 16: 3D simulation equivalent stress

The deformed configuration shows an opening of the interface at the corner of the structure. Moreover, a comparison of interface damage is shown in Figure 17 between these two steps. The propagation is clearly visible with a concentration at the corner but also along copper edge.



(a) After cooling (b) After 20 cycles

Figure 17: 3D simulation interface damage

From these restuls, this simulation has shown its capability to evaluate the propagation in a quantitative way using finite element method.

5 Conclusion

This study provides significant insights into the thermo-mechanical behaviour and reliability of Direct Bonded Copper (DBC) components, which are critical components in power electronics. Through comprehensive modelling and simulation techniques, including the analysis of material behaviours, interface integrity, and the impact of thermal cycling, this study identifies key factors that contribute to DBC failure. The finite element simulations, coupled with experimental data, offer a detailed understanding of stress distribution and failure mechanisms, particularly at the copper-ceramic interface. The findings underscore the importance of accurate modelling and advanced design strategies to enhance the durability of DBC substrates under demanding operational conditions. This work not only advances the current knowledge of DBC component performance but also provides a robust framework for future studies aimed at improving the reliability of power electronics components.

A Source code of 4-point bending simulation

```
import numpy as _np

from abaqus import mdb as _mdb, session as _session
import part as _part
import mesh as _mesh
import section as _section
import material as _material
import step as _step
import interaction as _interaction
import visualization as _visualization



class geometry:
    sample_length = 55. # mm: sample total length
    crack_position = 0.5 # mm: position from symmetry plane (mid-length) of the initial crack in the ceramic layer
    crack_angle = 90. #degree
    decohesion_length = 0.02 # mm: length of initial decohesion length from crack end

class ceramic:
    thickness = 0.25 # mm
    young = 330000. #GPa
    poisson = 0.22 # -

class copper:
    thickness = 0.5 # mm
    notch = 2. # mm: notch length
    young = 127000. #GPa
    poisson = 0.33 # -
    R0 = 43.2 # MPa
    C1 = 27000. # MPa
    D1 = 1850. # -
    C2 = 1120. # MPa
    D2 = 14.8 # -
    Q = 30. # MPa
    b = 12.8 # -

class _interface:
    length = 1.5 # mm
    k = 1.e8 # N/mm
    Tmax = 150. # MPa
    Gc = 0.01 # N/mm
interfaces = (_interface(), _interface() )

class pins:
    diameter = 4. # mm
    outer_span = 50. # mm
    inner_span = 25. # mm
    inner_lever = 20. # mm

class mesh:
    global_size = 0.05 #0.1 # mm
    interface_size = 0.005 # 0.02 # mm

name = 'simulation'
__model__ = None
```

```
57
58
59
60  def create():
61      '''
62      Create the simulation model in memory.
63
64      This function do not take any parameter but uses the module parameters
65      classes to generate the model.
66
67      Example
68      -------
69          >>> import model
70          >>> model.mesh.interface_size = 0.01 # Change some parameters
71          >>> model.create()
72
73      Warning
74      -------
75          The model is regenerated at each call using current module parameters
76      '''
77      # Delete previous generation and create a new one
78      global __model__
79      if __model__ is not None:
80          _mdb.models.changeKey(fromName=__model__.name, toName='To-Delete')
81      model = _mdb.Model(name=name)
82
83      del _mdb.models['Model-1' if __model__ is None else 'To-Delete']
84
85
86      # Create the base geometry
87      sketch = model.ConstrainedSketch(name='sketch', sheetSize=1.0)
88      sketch.rectangle(
89          point1=(-geometry.sample_length/2., -copper.thickness),
90          point2=(geometry.sample_length/2., ceramic.thickness+copper.thickness)
91      )
92      sample = model.Part(
93          name='Sample',
94          dimensionality=_part.TWO_D_PLANAR,
95          type=_part.DEFORMABLE_BODY
96      )
97      sample.BaseShell(sketch=sketch)
98      del model.sketches['sketch']
99
100     # Create notch
101     sketch = model.ConstrainedSketch(name='sketch', sheetSize=1.0)
102     sketch.rectangle(
103         point1=(-copper.notch/2., ceramic.thickness),
104         point2=(copper.notch/2., ceramic.thickness+copper.thickness)
105     )
106     sample.Cut(sketch=sketch)
107     del model.sketches['sketch']
108
109     # Create partition
110     alpha = _np.radians(geometry.crack_angle) if geometry.crack_angle != 90. else _np.nan
111     crack_end = 0. if _np.isnan(alpha) else ceramic.thickness/_np.tan(alpha)
112     regular_end = 0. if _np.isnan(alpha) else crack_end/ceramic.thickness*mesh.interface_size
113
114     sketch = model.ConstrainedSketch(name='sketch', sheetSize=1.0)
115     sketch.Line( # ceramic top left
```

```
116             point1=(-geometry.sample_length/2., ceramic.thickness),
117             point2=(-copper.notch/2., ceramic.thickness)
118         )
119         sketch.Line( # ceramic top right
120             point1=(copper.notch/2., ceramic.thickness),
121             point2=(geometry.sample_length/2., ceramic.thickness)
122         )
123         sketch.Line( # ceramic bottom left
124             point1=(-geometry.sample_length/2., 0.),
125             point2=(geometry.crack_position-interfaces[0].length, 0.)
126         )
127         sketch.Line( # interfaces
128             point1=(geometry.crack_position-interfaces[0].length, 0.),
129             point2=(geometry.crack_position+interfaces[1].length, 0.)
130          )
131         sketch.Line( # ceramic bottom right
132             point1=(geometry.crack_position+interfaces[1].length, 0.),
133             point2=(geometry.sample_length/2., 0.)
134         )
135         sketch.Line( # underneath ceramic crack
136             point1=(geometry.crack_position, -mesh.interface_size),
137             point2=(geometry.crack_position, 0.)
138         )
139         sketch.Line( # underneath delamination left
140             point1=(geometry.crack_position-geometry.decohesion_length, -mesh.interface_size),
141             point2=(geometry.crack_position-geometry.decohesion_length, 0.)
142         )
143         sketch.Line( # underneath delamination right
144             point1=(geometry.crack_position+geometry.decohesion_length, -mesh.interface_size),
145             point2=(geometry.crack_position+geometry.decohesion_length, 0.)
146         )
147 
148         sketch.Line( # ceramic crack
149             point1=(geometry.crack_position, 0.),
150             point2=(geometry.crack_position+crack_end, ceramic.thickness)
151         )
152         sketch.Line( # delamination left
153             point1=(geometry.crack_position-geometry.decohesion_length, 0.),
154             point2=(geometry.crack_position-geometry.decohesion_length+crack_end, ceramic.thickness)
155         )
156         sketch.Line( # delamination right
157             point1=(geometry.crack_position+geometry.decohesion_length, 0.),
158             point2=(geometry.crack_position+geometry.decohesion_length+crack_end, ceramic.thickness)
159         )
160         sketch.Line( # left interface end
161             point1=(geometry.crack_position-interfaces[0].length, -mesh.interface_size),
162             point2=(geometry.crack_position-interfaces[0].length,  mesh.interface_size),
163         )
164         sketch.Line( # right interface end
165             point1=(geometry.crack_position+interfaces[1].length, -mesh.interface_size),
166             point2=(geometry.crack_position+interfaces[1].length,  mesh.interface_size),
167         )
168         sketch.Line( # regular bottom
169             point1=(geometry.crack_position-interfaces[0].length, -mesh.interface_size),
170             point2=(geometry.crack_position+interfaces[1].length, -mesh.interface_size)
171         )
172         sketch.Line( # regular top
173             point1=(geometry.crack_position-interfaces[0].length, mesh.interface_size),
174             point2=(geometry.crack_position+interfaces[1].length, mesh.interface_size)
```

```
    )
    sketch.Line( # regular left
        point1=(geometry.crack_position-interfaces[0].length-mesh.global_size*3., -copper.thickness),
        point2=(geometry.crack_position-interfaces[0].length-mesh.global_size*3., ceramic.thickness)
    )
    sketch.Line( # regular right
        point1=(geometry.crack_position+interfaces[1].length+mesh.global_size*3., -copper.thickness),
        point2=(geometry.crack_position+interfaces[1].length+mesh.global_size*3., ceramic.thickness)
    )
    sample.PartitionFaceBySketch(faces=sample.faces, sketch=sketch)
    del model.sketches['sketch']

    # Create pins
    alpha=_np.radians(30.)
    radius = pins.diameter/2.-mesh.global_size
    sketch = model.ConstrainedSketch(name='sketch', sheetSize=1.0)
    sketch.ArcByCenterEnds(
        center=(0.0, 0.0),
        point1=( pins.diameter/2.*_np.cos(alpha), pins.diameter/2.*_np.sin(alpha)),
        point2=(-pins.diameter/2.*_np.cos(alpha), pins.diameter/2.*_np.sin(alpha)),
        direction=_part.COUNTERCLOCKWISE
    )
    sketch.ArcByCenterEnds(
        center=(0.0, 0.0),
        point1=( radius*_np.cos(alpha), radius*_np.sin(alpha)),
        point2=(-radius*_np.cos(alpha), radius*_np.sin(alpha)),
        direction=_part.COUNTERCLOCKWISE
    )
    sketch.Line(
        point1=(-radius*_np.cos(alpha), radius*_np.sin(alpha)),
        point2=(-pins.diameter/2.*_np.cos(alpha), pins.diameter/2.*_np.sin(alpha))
    )
    sketch.Line(
        point1=( radius*_np.cos(alpha), radius*_np.sin(alpha)),
        point2=( pins.diameter/2.*_np.cos(alpha), pins.diameter/2.*_np.sin(alpha))
    )
    pin = model.Part(name='Pin', dimensionality=_part.TWO_D_PLANAR, type=_part.DEFORMABLE_BODY)
    pin.BaseShell(sketch=sketch)
    del model.sketches['sketch']


    # Get crack and interface edges
    crack = sample.edges.findAt(
        ((geometry.crack_position+crack_end/2., ceramic.thickness/2., 0.), ), # crack
        ((geometry.crack_position+regular_end/2, mesh.interface_size/2., 0.), ), # regular ligament
        ((geometry.crack_position-geometry.decohesion_length/2., 0., 0.), ), # left decohesion
        ((geometry.crack_position+geometry.decohesion_length/2., 0., 0.), ) # right decohesion
    )
    bonds = (
        sample.edges.findAt((geometry.crack_position-interfaces[0].length/2., 0., 0.), ), # left interface
        sample.edges.findAt((geometry.crack_position+interfaces[1].length/2., 0., 0.), ), # right interface
    )


    # Create the assembly
    assembly = model.rootAssembly
    instance = assembly.Instance(name='Sample', part=sample, dependent=True)
    left_inner_pin = assembly.Instance(name='Left-Inner-Pin', part=pin, dependent=True)
    right_inner_pin = assembly.Instance(name='Right-Inner-Pin', part=pin, dependent=True)
```

```
234     left_outer_pin = assembly.Instance(name='Left-Outer-Pin', part=pin, dependent=True)
235     right_outer_pin = assembly.Instance(name='Right-Outer-Pin', part=pin, dependent=True)
236
237     assembly.rotate(
238         instanceList=('Left-Outer-Pin', 'Right-Outer-Pin'),
239         axisPoint=(0.0, 0.0, 0.0),
240         axisDirection=(0.0, 0.0, 1.0),
241         angle=180.0
242     )
243     assembly.translate(
244         instanceList=('Left-Inner-Pin', ),
245         vector=(-pins.inner_span/2., -copper.thickness-pins.diameter/2., 0.0)
246     )
247     assembly.translate(
248         instanceList=('Right-Inner-Pin', ),
249         vector=( pins.inner_span/2., -copper.thickness-pins.diameter/2., 0.0)
250     )
251     assembly.translate(
252         instanceList=('Left-Outer-Pin', ),
253         vector=(-pins.outer_span/2., ceramic.thickness+copper.thickness+pins.diameter/2., 0.0)
254     )
255     assembly.translate(
256         instanceList=('Right-Outer-Pin', ),
257         vector=( pins.outer_span/2., ceramic.thickness+copper.thickness+pins.diameter/2., 0.0)
258     )
259
260
261     # Mesh the parts
262     elem_type = _mesh.ElemType(elemCode=_mesh.CPE4I, elemLibrary=_mesh.STANDARD)
263     sample.setElementType(regions=(sample.faces,), elemTypes=(elem_type,))
264     sample.setMeshControls(regions=sample.faces, elemShape=_mesh.QUAD)
265
266     # Set global size
267     sample.seedPart(size=mesh.global_size)
268     edges = sample.edges.findAt(
269         ((-geometry.sample_length/4., ceramic.thickness+copper.thickness, 0.), ),
270         (( geometry.sample_length/4., ceramic.thickness+copper.thickness, 0.), ),
271         ((-geometry.sample_length/4., ceramic.thickness, 0.), ),
272         (( geometry.sample_length/4., ceramic.thickness, 0.), ),
273         (( 0., -copper.thickness, 0.), ),
274     )
275     sample.seedEdgeBySize(edges=edges, size=mesh.global_size, constraint=_mesh.FINER)
276
277     # Set interface size
278     edges = sample.edges.findAt(
279         (( geometry.crack_position-interfaces[0].length/2.,  mesh.interface_size, 0.), ),
280         (( geometry.crack_position-interfaces[0].length/2.,  0., 0.), ),
281         (( geometry.crack_position-interfaces[0].length/2., -mesh.interface_size, 0.), ),
282         (( geometry.crack_position+interfaces[1].length/2.,  mesh.interface_size, 0.), ),
283         (( geometry.crack_position+interfaces[1].length/2.,  0., 0.), ),
284         (( geometry.crack_position+interfaces[1].length/2., -mesh.interface_size, 0.), ),
285         (( geometry.crack_position+crack_end/2.-geometry.decohesion_length, ceramic.thickness/2., 0.), ),
286         (( geometry.crack_position+crack_end/2., ceramic.thickness/2., 0.), ),
287         (( geometry.crack_position+crack_end/2.+geometry.decohesion_length, ceramic.thickness/2., 0.), ),
288         (( geometry.crack_position-mesh.interface_size/2.,  -mesh.interface_size, 0.), ),
289         (( geometry.crack_position-mesh.interface_size/2.,  0., 0.), ),
290         (( geometry.crack_position+regular_end-mesh.interface_size/2., mesh.interface_size, 0.), ),
291         (( geometry.crack_position+crack_end-mesh.interface_size/2., ceramic.thickness, 0.), ),
292         (( geometry.crack_position+mesh.interface_size/2.,  -mesh.interface_size, 0.), ),
```

```
        (( geometry.crack_position+mesh.interface_size/2.,  0., 0.), ),
        (( geometry.crack_position+regular_end+mesh.interface_size/2., mesh.interface_size, 0.), ),
        (( geometry.crack_position+crack_end+mesh.interface_size/2., ceramic.thickness, 0.), ),

    )
    sample.seedEdgeBySize(edges=edges, size=mesh.interface_size, constraint=_mesh.FINER)

    # Mesh the sample
    sample.generateMesh()

    # Mesh the pin
    elem_type = _mesh.ElemType(elemCode=_mesh.CPE4R, elemLibrary=_mesh.STANDARD)
    pin.setElementType(regions=(pin.faces,), elemTypes=(elem_type,))

    pin.seedPart(size=mesh.global_size)
    edges = pin.edges.findAt(
        ((0.,  pins.diameter/2., 0.), ),
        ((0.,  radius, 0.), ),
    )
    sample.seedEdgeBySize(edges=edges, size=mesh.global_size/2., constraint=_mesh.FINER)
    pin.generateMesh()


    # Create interface
    etype = _mesh.ElemType(elemCode=_mesh.COH2D4, elemLibrary=_mesh.STANDARD, viscosity=0.001)

    # Create crack: insert cohesive elements and delete them
    nb_elements = len(sample.elements)
    region = _part.Region(side1Edges=crack )
    sample.insertElements(edges=region)

    crack_set = sample.Set(name='Crack-Elements', elements=sample.elements[nb_elements:] )
    sample.setElementType(regions=crack_set, elemTypes=(etype,) )

    sample.deleteElement(sample.elements[nb_elements:])
    del sample.sets['Crack-Elements']

    # Insert cohesive elements in each interface
    for side, interface in zip(('Left', 'Right'), bonds):
        nb_elements = len(sample.elements)
        region = _part.Region(side1Edges=_part.EdgeArray((interface,)) )
        sample.insertElements(edges=region)

        interface_set = sample.Set(name=side+'-Interface-Elements', elements=sample.elements[nb_elements:] )
        sample.setElementType(regions=interface_set, elemTypes=(etype,) )


    # Create sets
    faces = sample.faces.findAt(
        ((-geometry.sample_length*0.45,  ceramic.thickness/2., 0.), ),
        (( geometry.crack_position-interfaces[0].length/2.,  ceramic.thickness/2., 0.), ),
        (( geometry.crack_position+interfaces[1].length/2.,  ceramic.thickness/2., 0.), ),
        (( geometry.crack_position+crack_end/2.+geometry.decohesion_length*2.,  ceramic.thickness/2., 0.), ),
        (( geometry.crack_position+crack_end/2.-geometry.decohesion_length/2.,  ceramic.thickness/2., 0.), ),
        (( geometry.crack_position+crack_end/2.+geometry.decohesion_length/2.,  ceramic.thickness/2., 0.), ),
        (( geometry.sample_length*0.45,  ceramic.thickness/2., 0.), ),
        (( geometry.crack_position+regular_end/2.-geometry.decohesion_length/2.,  mesh.interface_size/2., 0.), ),
        (( geometry.crack_position+regular_end/2.+geometry.decohesion_length/2.,  mesh.interface_size/2., 0.), ),
        (( geometry.crack_position-interfaces[0].length/2.,  mesh.interface_size/2., 0.), ),
```

```
        (( geometry.crack_position+interfaces[1].length/2.,  mesh.interface_size/2., 0.), )
    )
    ceramic_set = sample.Set(name='Ceramic-Set', faces=faces )

    faces = sample.faces.findAt(
        ((-geometry.sample_length*0.45, ceramic.thickness+copper.thickness/2., 0.), ),
        (( geometry.sample_length*0.45, ceramic.thickness+copper.thickness/2., 0.), ),
        ((-geometry.sample_length*0.45, -copper.thickness/2., 0.), ),
        (( geometry.sample_length*0.45, -copper.thickness/2., 0.), ),

        (( geometry.crack_position-geometry.decohesion_length/2.,  -mesh.interface_size/2., 0.), ),
        (( geometry.crack_position+geometry.decohesion_length/2.,  -mesh.interface_size/2., 0.), ),
        (( geometry.crack_position-interfaces[0].length/2.,  -mesh.interface_size/2., 0.), ),
        (( geometry.crack_position+interfaces[1].length/2.,  -mesh.interface_size/2., 0.), ),
        (( geometry.crack_position, -copper.thickness/2., 0.), ),
    )
    copper_set = sample.Set(name='Copper-Set', faces=faces )

    edge = sample.edges.findAt(
        ((-geometry.sample_length/4.,  ceramic.thickness+copper.thickness, 0.), ),
    )
    sample.Surface(name='Top-Left-Surface', side1Edges=edge)

    edge = sample.edges.findAt(
        (( geometry.sample_length/4.,  ceramic.thickness+copper.thickness, 0.), ),
    )
    sample.Surface(name='Top-Right-Surface', side1Edges=edge)

    edge = sample.edges.findAt(
        ((-geometry.sample_length*0.45, -copper.thickness, 0.), ),
        (( geometry.crack_position, -copper.thickness, 0.), ),
        (( geometry.sample_length*0.45, -copper.thickness, 0.), ),
        )
    sample.Surface(name='Bottom-Surface', side1Edges=edge)

    edge = pin.edges.findAt( (( 0., pins.diameter/2., 0.), ), )
    pin.Surface(name='Pin-Surface', side1Edges=edge)

    pin.Set(name='Pin', faces=pin.faces)


    # Assign materials
    mat = model.Material(name='Pin')
    mat.Elastic(table=((copper.young, copper.poisson), ) )
    model.HomogeneousSolidSection(name='Pin-Section', material='Pin', thickness=1)
    pin.SectionAssignment(region=pin.sets['Pin'], sectionName='Pin-Section')

    mat = model.Material(name='Ceramic')
    mat.Elastic(table=((ceramic.young, ceramic.poisson), ) )
    model.HomogeneousSolidSection(name='Ceramic-Section', material='Ceramic', thickness=1)
    sample.SectionAssignment(region=ceramic_set, sectionName='Ceramic-Section')

    mat = model.Material(name='Copper')
    mat.Elastic(table=((copper.young, copper.poisson), ) )
    mat.Plastic(
        hardening=_material.COMBINED,
        dataType=_material.PARAMETERS,
        numBackstresses=2,
        table=((copper.R0, copper.C1, copper.D1, copper.C2, copper.D2), )
```

```
    )
    mat.plastic.CyclicHardening(parameters=True, table=((copper.R0, copper.Q, copper.b), ) )
    model.HomogeneousSolidSection(name='Copper-Section', material='Copper', thickness=1)
    sample.SectionAssignment(region=copper_set, sectionName='Copper-Section')

    for side, interface in zip(('Left', 'Right'), interfaces):
        czm = model.Material(name=side+'-Interface')
        czm.Elastic(table=((interface.k, interface.k, interface.k), ), type=_material.TRACTION)
        czm.QuadsDamageInitiation(table=((interface.Tmax, interface.Tmax, interface.Tmax), ))
        czm.quadsDamageInitiation.DamageEvolution(table=((interface.Gc, ), ), type=_material.ENERGY)

        model.CohesiveSection(
            name=side+'-Interface-Section',
            material=side+'-Interface',
            response=_section.TRACTION_SEPARATION
        )
        sample.SectionAssignment(
            region=sample.sets[side+'-Interface-Elements'],
            sectionName=side+'-Interface-Section'
        )


    # Create simulation step
    step = model.StaticStep(
        name = 'Step',
        nlgeom = True,
        previous='Initial',
        initialInc=0.01,
        maxInc=0.01,
        maxNumInc=100000,
        minInc=1.e-16,
        timePeriod=4
    )
    step.control.setValues(
        allowPropagation=False,
        resetDefaultValues=False,
        timeIncrementation=(4.0, 8.0, 9.0, 16.0, 10.0, 4.0, 12.0, 50.0, 6.0, 3.0, 50.0)
    )
    model.fieldOutputRequests["F-Output-1"].setValues(
        variables=('U', 'RF', 'E', 'PE', 'PEEQ', 'S', 'SDEG', 'STATUS'),
        timeInterval=0.05
    )


    # Create inner pins rigid body
    rpid = assembly.ReferencePoint(point=(0.0, -copper.thickness-pins.diameter/2.-pins.inner_lever, 0.0)).id
    assembly.Set(referencePoints=(assembly.referencePoints[rpid], ), name='RefPoint')

    edges = assembly.instances['Left-Inner-Pin'].edges + assembly.instances['Right-Inner-Pin'].edges
    model.RigidBody(
        name='Constraint-1',
        refPointRegion=_part.Region(referencePoints=(assembly.referencePoints[rpid],)) ,
        pinRegion=_part.Region(edges=edges)
    )


    # Create Boundary Conditions
    model.DisplacementBC(
        name='Inner-Pins',
```

```
        createStepName='Initial',
        region=assembly.sets['RefPoint'],
        u1=0.0,
        u2=0.0
    )
    model.DisplacementBC(
        name='Left-Outer-Pin',
        createStepName='Step',
        region=left_outer_pin.sets['Pin'],
        u1=0.0,
        u2=-4.0
    )
    model.DisplacementBC(
        name='Right-Outer-Pin',
        createStepName='Step',
        region=right_outer_pin.sets['Pin'],
        u1=0.0,
        u2=-4.0
    )


    # Create frictionless contact
    model.ContactProperty('Frictionless-Contact')
    model.interactionProperties['Frictionless-Contact'].NormalBehavior(
        constraintEnforcementMethod=_interaction.AUGMENTED_LAGRANGE
    )

    # Create standard contact
    model.ContactProperty('Contact')
    model.interactionProperties['Contact'].NormalBehavior(
        constraintEnforcementMethod=_interaction.AUGMENTED_LAGRANGE
    )
    model.interactionProperties['Contact'].TangentialBehavior(
        formulation=_interaction.PENALTY,
        table=((0.35, ), ),
        fraction=0.005
    )

    model.SurfaceToSurfaceContactStd(
        name='Left-Inner-Contact',
        createStepName='Initial',
        master=instance.surfaces['Bottom-Surface'],
        slave=left_inner_pin.surfaces['Pin-Surface'],
        sliding=_interaction.SMALL, #FINITE,
        enforcement=_interaction.NODE_TO_SURFACE,
        interactionProperty='Frictionless-Contact',
    )
    model.SurfaceToSurfaceContactStd(
        name='Right-Inner-Contact',
        createStepName='Initial',
        master=instance.surfaces['Bottom-Surface'],
        slave=right_inner_pin.surfaces['Pin-Surface'],
        sliding=_interaction.SMALL, #FINITE,
        enforcement=_interaction.NODE_TO_SURFACE,
        interactionProperty='Frictionless-Contact',
    )
    model.SurfaceToSurfaceContactStd(
        name='Right-Outer-Contact',
        createStepName='Initial',
```

```
        master=instance.surfaces['Top-Right-Surface'],
        slave=right_outer_pin.surfaces['Pin-Surface'],
        sliding=_interaction.SMALL, #FINITE,
        enforcement=_interaction.NODE_TO_SURFACE,
        interactionProperty='Frictionless-Contact',
    )

    model.SurfaceToSurfaceContactStd(
        name='Left-Outer-Contact',
        createStepName='Initial',
        master=instance.surfaces['Top-Left-Surface'],
        slave=left_outer_pin.surfaces['Pin-Surface'],
        sliding=_interaction.SMALL, #FINITE,
        enforcement=_interaction.NODE_TO_SURFACE,
        interactionProperty='Contact',
    )


    # Store the model globaly
    __model__ = model



def update(side, Tmax, Gc):
    '''
    Update simulation model interface parameters.

    Parameters
    ----------
    side: str ('Left', 'Right')
        Side to update

    Tmax: float
        New value of Tmax

    Gc: float
        New value of Gc

    Example
    -------
        >>> import model
        >>> model.create()
        >>> model.update('left', 150, 0.01)
        >>> model.write_input()

    Raises
    ------
        RuntimeError if the model has not been generated
    '''
    if __model__ is None:
        raise RuntimeError('Model not generated')

    interface = side+'-Interface'
    interface = __model__.materials[interface]
    interface.quadsDamageInitiation.setValues(table=((Tmax, Tmax, Tmax), ))
    interface.quadsDamageInitiation.damageEvolution.setValues(table=((Gc, ), ))
```

```
588 def write_input():
589     '''
590     Write the simulation inp file from the created model
591 
592     Example
593     -------
594         >>> import model
595         >>> model.create()
596         >>> model.write_input()
597 
598     Raises
599     ------
600         RuntimeError if the model has not been generated
601     '''
602     if __model__ is None:
603         raise RuntimeError('Model not generated')
604 
605     job = _mdb.Job(name=__model__.name, model=__model__)
606     job.writeInput(consistencyChecking=False)
607     del _mdb.jobs[__model__.name]
608 
609 
610 
611 def post_process(odbname=None):
612     '''
613     Process simulation output data to compute decohesion length for each interface
614 
615     Parameters
616     ----------
617     odbname: str, optional
618         When the model is not created, simulation output filename without extension
619 
620     Example
621     -------
622         >>> from subprocess import run
623         >>> import model
624         >>> model.create()
625         >>> model.write_input()
626         >>> run('abaqus job=simulation interactive', shell=True)
627         >>> model.post_process()
628 
629     Example
630     -------
631         >>> import model
632         >>> model.post_process('previous_run.odb')
633 
634     Raises
635     ------
636         RuntimeError if the model has not been generated and no output file is specified
637     '''
638     if __model__ is None and odbname is None:
639         raise RuntimeError('Model not generated')
640 
641     # Get a dedicated viewport
642     viewport = _session.viewports.values()[0]
643     viewport.makeCurrent()
644 
645     # Open odb file
646     if odbname is None:
```

```
        odbname = __model__.name
    odb = _session.openOdb(odbname+'.odb', readOnly=True, readInternalSets=True)
    viewport.setValues(displayedObject=odb)

    # Get load on inner pins reference point
    data = _session.xyDataListFromField(
        odb=odb,
        outputPosition=_visualization.NODAL,
        variable=(('RF', _visualization.NODAL, ((_visualization.COMPONENT, 'RF2'), )), ),
        nodeSets=("REFPOINT", )
    )
    results = _np.array(data[0])
    with open(odbname+'/load.sim', 'w') as fout:
        fout.write('#displacement[mm] load[N/mm]\n')
        _np.savetxt(fout, results)


    # Get left decohesion length
    instance = odb.rootAssembly.instances['SAMPLE']
    elements = instance.elementSets['LEFT-INTERFACE-ELEMENTS'].elements

    nids = []
    for element in elements:
        nids.extend(element.connectivity[:2])
    nids = _np.array(nids, dtype=int)-1
    coords = _np.array([instance.nodes[nid].coordinates[0] for nid in nids])
    ranks = _np.argsort(coords)
    coords = _np.unique(coords[ranks])

    # - Get data at integration points
    data = _session.xyDataListFromField(
        odb=odb,
        outputPosition=_visualization.INTEGRATION_POINT,
        variable=(('SDEG', _visualization.INTEGRATION_POINT), ),
        elementSets=("SAMPLE.LEFT-INTERFACE-ELEMENTS", )
    )
    data = _np.array(data)[:,:,1][ranks]

    # - Convert to single node position
    crack = _np.empty(( len(coords), data.shape[1] ))
    crack[0] = data[0]
    crack[1:-1] = (data[1:-2:2]+data[2:-1:2])/2.
    crack[-1] = data[-1]

    # - Get crack front position
    for i, data in enumerate(crack.T):
        for xa, xb, ya, yb in zip(coords[:-1], coords[1:], data[:-1], data[1:]):
            if yb>0.999: break
        if yb<0.1: results[i,1] = geometry.crack_position
        elif yb>0.99999: results[i,1] = xb
        else: results[i,1] = xa + (0.999-ya) * (xb-xa) / (yb-ya)

    # - Compute crack length
    results[:,1] = geometry.crack_position-results[:,1]
    with open(odbname+'/crack_left.sim', 'w') as fout:
        fout.write('#displacement[mm] length[mm]\n')
        _np.savetxt(fout, results)
```

```
    # Get right decohesion length
    instance = odb.rootAssembly.instances['SAMPLE']
    elements = instance.elementSets['RIGHT-INTERFACE-ELEMENTS'].elements

    nids = []
    for element in elements:
        nids.extend(element.connectivity[:2])
    nids = _np.array(nids, dtype=int)-1
    coords = _np.array([instance.nodes[nid].coordinates[0] for nid in nids])
    ranks = _np.argsort(coords)
    coords = _np.unique(coords[ranks])[::-1]

    # - Get data at integration points
    data = _session.xyDataListFromField(
        odb=odb,
        outputPosition=_visualization.INTEGRATION_POINT,
        variable=(('SDEG', _visualization.INTEGRATION_POINT), ),
        elementSets=("SAMPLE.RIGHT-INTERFACE-ELEMENTS", )
    )
    data = _np.array(data)[:,:,1][ranks][::-1]

    # - Convert to single node position
    crack = _np.empty(( len(coords), data.shape[1] ))
    crack[0] = data[0]
    crack[1:-1] = (data[1:-2:2]+data[2:-1:2])/2.
    crack[-1] = data[-1]

    # - Get crack front position
    for i, data in enumerate(crack.T):
        for xa, xb, ya, yb in zip(coords[:-1], coords[1:], data[:-1], data[1:]):
            if yb>0.999: break
        if yb<0.1: results[i,1] = geometry.crack_position
        elif yb>0.99999: results[i,1] = xb
        else: results[i,1] = xa + (0.999-ya) * (xb-xa) / (yb-ya)

    # - Compute crack length
    results[:,1] -= geometry.crack_position
    with open(odbname+'/crack_right.sim', 'w') as fout:
        fout.write('#displacement[mm] length[mm]\n')
        _np.savetxt(fout, results)

    odb.close()
```

B Source code of axi-symmetric DBC simulation

```
import numpy as _np

from abaqus import mdb as _mdb, session as _session
import part as _part
import mesh as _mesh
import section as _section
import material as _material
import step as _step
import load as _load
import visualization as _visualization



class ceramic:
    length = 0.750 # mm
    thickness = 0.317 # mm
    young = 330000. #GPa
    poisson = 0.22 # -
    cte = 8.0e-6 # /dC

class copper:
    length = 0.650 # mm
    thickness = 0.127 # mm
    young = 127000. # MPa
    poisson = 0.33 # -
    cte = 16.5e-6 # /dC

class mesh:
    global_size = 0.05 #0.1 # mm
    interface_size = 0.005 # 0.02 # mm

class loads:
    class cooling:
        initial = 1000.0 # dC
        final = 0.0 # dC
        duration = 1.0 # s

name = 'simulation'
__model__ = None



def create():
    '''
    Create the simulation model in memory.

    This function do not take any parameter but uses the module parameters
    classes to generate the model.

    Example
    -------
        >>> import model
        >>> model.mesh.size = 0.01 # Change some parameters
        >>> model.create()

    Warning
```

```
    -------
        The model is regenerated at each call using current module parameters
    '''
    # Delete previous generation and create a new one
    global __model__
    if __model__ is not None:
        _mdb.models.changeKey(fromName=__model__.name, toName='To-Delete')
    model = _mdb.Model(name=name)

    del _mdb.models['Model-1' if __model__ is None else 'To-Delete']

    # Create geometry
    sketch = model.ConstrainedSketch(name='sketch', sheetSize=1.0)

    # Add a contruction line for axisymmetry
    sketch.ConstructionLine(point1=(0.0, -100.0), point2=(0.0, 100.0))

    # Define contour lines
    sketch.Line(
        point1=(0.0, 0.0),
        point2=(ceramic.length, 0.0)
    )
    sketch.Line(
        point1=(ceramic.length, 0.0),
        point2=(ceramic.length, ceramic.thickness)
    )
    sketch.Line(
        point1=(ceramic.length, ceramic.thickness),
        point2=(copper.length, ceramic.thickness)
    )
    sketch.Line(
        point1=(copper.length, ceramic.thickness),
        point2=(copper.length, ceramic.thickness+copper.thickness)
    )
    sketch.Line(
        point1=(copper.length, ceramic.thickness+copper.thickness),
        point2=(0.0, ceramic.thickness+copper.thickness)
    )
    sketch.Line(
        point1=(0.0, ceramic.thickness+copper.thickness),
        point2=(0.0, 0.0)
    )

    # Create the part
    part = model.Part(
        name='DBC',
        dimensionality=_part.AXISYMMETRIC,
        type=_part.DEFORMABLE_BODY
    )
    part.BaseShell(sketch=sketch)
    del model.sketches['sketch']

    # Create interface
    sketch = model.ConstrainedSketch(name='sketch', sheetSize=1.0)
    sketch.Line(
        point1=(0.0, ceramic.thickness),
        point2=(copper.length, ceramic.thickness)
    )
    part.PartitionFaceBySketch(faces=part.faces, sketch=sketch)
```

```
    del model.sketches['sketch']

    # Create regular domain in copper
    size = mesh.interface_size
    sketch = model.ConstrainedSketch(name='sketch', sheetSize=1.0)
    sketch.Line(
        point1=(0.0, ceramic.thickness+size),
        point2=(copper.length, ceramic.thickness+size)
    )
    part.PartitionFaceBySketch(
        faces=part.faces.findAt( (0.0, ceramic.thickness+copper.thickness, 0.0) ),
        sketch=sketch
    )
    del model.sketches['sketch']

    # Create regular domains in ceramic
    sketch = model.ConstrainedSketch(name='sketch', sheetSize=1.0)
    sketch.Line(
        point1=(0.0, ceramic.thickness-size),
        point2=(copper.length, ceramic.thickness-size)
    )
    sketch.Line(
        point1=(copper.length, ceramic.thickness-size),
        point2=(copper.length, ceramic.thickness)
    )
    sketch.Line(
        point1=(copper.length, ceramic.thickness-size),
        point2=(copper.length+size, ceramic.thickness-size)
    )
    sketch.Line(
        point1=(copper.length+size, ceramic.thickness-size),
        point2=(copper.length+size, ceramic.thickness)
    )
    part.PartitionFaceBySketch(faces=part.faces.findAt( (0.0, 0.0, 0.0) ), sketch=sketch)
    del model.sketches['sketch']

    # Create reference points for Generalized Plane Strain
    part.ReferencePoint(point=part.vertices.findAt( (0.0, 0.0, 0.0) ))

    # Create assembly instance
    assembly = model.rootAssembly
    instance = assembly.Instance(name='DBC', part=part, dependent=True)


    # Setup the mesh
    part.setMeshControls(regions=part.faces, elemShape=_mesh.QUAD)
    etype = _mesh.ElemType(elemCode=_mesh.CAX4, elemLibrary=_mesh.STANDARD)
    part.setElementType(regions=(part.faces,), elemTypes=(etype,) )

    # Set element sizes
    part.seedPart(size=mesh.global_size, deviationFactor=0.01, minSizeFactor=0.1)

    edge = part.edges.findAt( (copper.length/2, ceramic.thickness+copper.thickness, 0.0) )
    part.seedEdgeBySize(edges=(edge,), size=mesh.global_size, constraint=_mesh.FINER)

    edge = part.edges.findAt( (ceramic.length/2, 0.0, 0.0) )
    part.seedEdgeBySize(edges=(edge,), size=mesh.global_size, constraint=_mesh.FINER)

    edge = part.edges.findAt( (copper.length, ceramic.thickness+copper.thickness/2, 0.0) )
```

```
    part.seedEdgeByBias(
        end1Edges=(edge,),
        minSize=mesh.interface_size,
        maxSize=mesh.global_size/2,
        biasMethod=_mesh.SINGLE,
        constraint=_mesh.FINER,
    )

    edge = part.edges.findAt( ((copper.length+ceramic.length)/2, ceramic.thickness, 0.0) )
    part.seedEdgeByBias(
        end2Edges=(edge,),
        minSize=mesh.interface_size,
        maxSize=mesh.global_size/2,
        biasMethod=_mesh.SINGLE,
        constraint=_mesh.FINER,
    )

    interface = part.edges.findAt( (copper.length/2, ceramic.thickness, 0.0) )
    part.seedEdgeBySize(edges=(interface,), size=mesh.interface_size, constraint=_mesh.FINER)

    size = mesh.interface_size
    edge = part.edges.findAt( (copper.length/2, ceramic.thickness+size, 0.0) )
    part.seedEdgeBySize(edges=(edge,), size=mesh.interface_size, constraint=_mesh.FINER)

    edge = part.edges.findAt( (copper.length/2, ceramic.thickness-size, 0.0) )
    part.seedEdgeBySize(edges=(edge,), size=mesh.interface_size, constraint=_mesh.FINER)

    # Generate mesh
    part.generateMesh()
    if part.getUnmeshedRegions():
        raise RuntimeError('Mesh generation failed')


    # Create bulk element sets
    elements = part.faces.findAt( (0,0,0) ).getElements()
    elements += part.faces.findAt( (copper.length/2.0, ceramic.thickness-size/2.0, 0.0) ).getElements()
    elements += part.faces.findAt( (copper.length+size/2.0, ceramic.thickness-size/2.0, 0.0) ).getElements()
    part.Set(name='Ceramic-Elements', elements=elements )

    elements = part.faces.findAt( (copper.length/2.0, ceramic.thickness+size/2.0, 0.0) ).getElements()
    elements += part.faces.findAt( (0,ceramic.thickness+copper.thickness,0) ).getElements()
    part.Set(name='Copper-Elements', elements=elements )

    # Generate interface elements
    nb_elements = len(part.elements)

    region = _part.Region(side1Edges=_part.EdgeArray((interface,)) )
    part.insertElements(edges=region)
    interface_set = part.Set(name='Interface-Elements', elements=part.elements[nb_elements:] )

    etype = _mesh.ElemType(elemCode=_mesh.COHAX4, elemLibrary=_mesh.STANDARD, viscosity=0.001)
    part.setElementType(regions=interface_set, elemTypes=(etype,) )

    # Get nodes at interface linked to cohesive zone elements
    nodes1, nodes2 = set(), set()
    for element in interface_set.elements:
        nodes1.update(element.connectivity[:2])
        nodes2.update(element.connectivity[2:])

```

```
    # Remove interface elements
    part.deleteElement(interface_set.elements)
    del part.sets['Interface-Elements']

    # Sort interface nodes by X position from end to symmetry plane
    nodes1 = [part.nodes[nid] for nid in nodes1]
    nodes2 = [part.nodes[nid] for nid in nodes2]
    nid1 = _np.argsort([node.coordinates[0] for node in nodes1])[::-1]
    nid2 = _np.argsort([node.coordinates[0] for node in nodes2])[::-1]
    nodes1 = [nodes1[i] for i in nid1]
    nodes2 = [nodes2[i] for i in nid2]

    # Create node sets
    part.Set(name='Nodes', elements=part.nodes)

    nodes = [node for node in part.nodes if node.coordinates[0] == 0]
    part.Set(name="X0", nodes=_mesh.MeshNodeArray(nodes))

    nodes = _mesh.MeshNodeArray( [node for node in part.nodes if node.coordinates[1] == 0] )
    part.Set(name="Y0", nodes=nodes)

    # Create interface node sets
    part.Set(name='Interface-Nodes_Side1', nodes=_mesh.MeshNodeArray(nodes1) )
    part.Set(name='Interface-Nodes_Side2', nodes=_mesh.MeshNodeArray(nodes2) )

    # Create node sets for all release steps
    for i, (node1, node2) in enumerate( zip(nodes1, nodes2), start=1):
        part.Set(name='Interface-Nodes_Side1-'+str(i), nodes=_mesh.MeshNodeArray([node1]) )
        part.Set(name='Interface-Nodes_Side2-'+str(i), nodes= _mesh.MeshNodeArray([node2]) )


    # Assign copper properties
    mat = model.Material(name='Copper')
    mat.Elastic(table=((copper.young, copper.poisson), ) )
    mat.Expansion(table=((copper.cte, ), ))

    model.HomogeneousSolidSection(name='Copper-Section', material='Copper', thickness=copper.length)
    part.SectionAssignment(
        region=part.sets['Copper-Elements'],
        sectionName='Copper-Section',
    )

    # Assign ceramic properties
    mat = model.Material(name='Ceramic')
    mat.Elastic(table=((ceramic.young, ceramic.poisson), ) )
    mat.Expansion(table=((ceramic.cte, ), ))

    model.HomogeneousSolidSection(name='Ceramic-Section', material='Ceramic', thickness=ceramic.length)
    part.SectionAssignment(
        region=part.sets['Ceramic-Elements'],
        sectionName='Ceramic-Section',
    )


    # Create steps
    step = model.StaticStep(
        name='Cooling',
        nlgeom=True,
        previous='Initial',
```

```
        initialInc=loads.cooling.duration/20.,
        maxInc=loads.cooling.duration/20.,
        maxNumInc=int(1e6),
        minInc=loads.cooling.duration/1e30,
        timePeriod=loads.cooling.duration
    )
    step.control.setValues(
        allowPropagation=False,
        resetDefaultValues=False,
        timeIncrementation=(4.0, 8.0, 9.0, 16.0, 10.0, 4.0, 12.0, 50.0, 6.0, 3.0, 50.0)
    )

    model.fieldOutputRequests["F-Output-1"].setValues(
        variables=('U', 'E', 'PE', 'PEEQ', 'S', 'SDEG', 'STATUS'),
        timeInterval=loads.cooling.duration/20.0,
    )

    num_steps = len(model.parts['DBC'].sets['Interface-Nodes_Side1'].nodes)-1
    previous = 'Cooling'
    for i in range(1, num_steps+1):
        step = 'Release'+str(i)

        model.StaticStep(
            name=step,
            nlgeom=True,
            previous=previous,
            initialInc=0.2,
            maxInc=0.2,
            maxNumInc=int(1e6),
            minInc=1e-30,
            timePeriod=1.,
        )
        previous = step
    model.fieldOutputRequests['F-Output-1'].setValuesInStep(stepName='Release1', timeInterval=1.0)
    model.historyOutputRequests['H-Output-1'].setValuesInStep(stepName='Release1', timeInterval=1.0)


    # Apply symmetry along X axis
    model.DisplacementBC(
        name='X-symmetry',
        createStepName="Initial",
        region=instance.sets['X0'],
        u1=_load.SET,
    )

    # Apply symmetry along Y axis
    model.DisplacementBC(
        name='Y-symmetry',
        createStepName="Initial",
        region=instance.sets['Y0'],
        u2=_load.SET,
    )

    # Apply temperature variation
    model.Temperature(
        name='Cooling',
        createStepName='Initial',
        region=instance.sets['Nodes'],
        magnitudes=(loads.cooling.initial, )
```

```
352         )
353 
354         model.predefinedFields['Cooling'].setValuesInStep(
355             stepName='Cooling',
356             magnitudes=(loads.cooling.final, )
357         )
358 
359         num_steps = len(instance.sets['Interface-Nodes_Side1'].nodes)-1
360 
361         for num in range(1, num_steps+1):
362             # Create reference point for corner node
363             position = model.parts['DBC'].sets['Interface-Nodes_Side1-'+str(num)].nodes[0].coordinates
364             rpid = assembly.ReferencePoint(point=position).id
365             assembly.Set(referencePoints=(assembly.referencePoints[rpid], ), name='RefPoint'+str(num))
366 
367             # Create equation
368             model.Equation(name='Equation'+str(num)+'-1', terms=(
369                 ( 1.0, 'DBC.Interface-Nodes_Side1-'+str(num), 1),
370                 (-1.0, 'DBC.Interface-Nodes_Side2-'+str(num), 1),
371                 (-1.0, 'RefPoint'+str(num), 1)
372             ))
373             model.Equation(name='Equation'+str(num)+'-2', terms=(
374                 ( 1.0, 'DBC.Interface-Nodes_Side1-'+str(num), 2),
375                 (-1.0, 'DBC.Interface-Nodes_Side2-'+str(num), 2),
376                 (-1.0, 'RefPoint'+str(num), 2)
377             ))
378 
379             # Create pinned bc
380             model.DisplacementBC(
381                 name='Pin'+str(num),
382                 createStepName="Initial",
383                 region=assembly.sets['RefPoint'+str(num)],
384                 u1=0.0,
385                 u2=0.0,
386             ).deactivate('Release'+str(num))
387 
388         # Store the model globaly
389         __model__ = model
390 
391 
392 
393 def write_input():
394     '''
395     Write the simulation inp file from the created model
396 
397     Example
398     -------
399         >>> import model
400         >>> model.create()
401         >>> model.write_input()
402 
403     Raises
404     ------
405         RuntimeError if the model has not been generated
406     '''
407     if __model__ is None:
408         raise RuntimeError('Model not generated')
409 
410     job = _mdb.Job(name=__model__.name, model=__model__)
```

```
411         job.writeInput(consistencyChecking=False)
412         del _mdb.jobs[__model__.name]
413 
414 
415 
416     def post_process(odbname=None):
417         '''
418         Process simulation output data to compute strain energy and stress vector along interface
419 
420         Parameters
421         ----------
422         odbname: str, optional
423             When the model is not created, simulation output filename without extension
424 
425         Example
426         -------
427             >>> from subprocess import run
428             >>> import model
429             >>> model.create()
430             >>> model.write_input()
431             >>> run('abaqus job=simulation interactive', shell=True)
432             >>> model.post_process()
433 
434         Example
435         -------
436             >>> import model
437             >>> model.post_process('previous_run.odb')
438 
439         Raises
440         ------
441             RuntimeError if the model has not been generated and no output file is specified
442         '''
443         if __model__ is None and odbname is None:
444             raise RuntimeError('Model not generated')
445 
446         # Get a dedicated viewport
447         viewport = _session.viewports.values()[0]
448         viewport.makeCurrent()
449 
450         # Open odb file
451         if odbname is None:
452             odbname = __model__.name
453         odb = _session.openOdb(odbname+'.odb', readOnly=True, readInternalSets=True)
454         viewport.setValues(displayedObject=odb)
455 
456         # Remove field average to get contours by element
457         viewport.odbDisplay.basicOptions.setValues(averageElementOutput=False)
458 
459         # Use undeformed shape
460         viewport.odbDisplay.commonOptions.setValues(deformationScaling=_part.UNIFORM, uniformScaleFactor=0)
461 
462         instance = odb.rootAssembly.instances['DBC']
463 
464         # Get interface nodes
465         coordinates = [node.coordinates[0] for node in instance.nodeSets['INTERFACE-NODES_SIDE1'].nodes]
466         ranks = _np.argsort( coordinates )[::-1]
467         crack_advance = _np.array(coordinates)[ranks]
468         interface_length = crack_advance[0]
469 
```

```
    crack_advance = interface_length-crack_advance

    # Compute global system energy
    energy = _session.XYDataFromHistory(
        name='Global strain energy',
        odb=odb,
        outputVariableName='Strain energy: ALLSE for Whole Model',
        steps=odb.steps.keys()
    ).data
    energy = _np.array(energy)
    energy = energy[ energy[:,0]>=1. ] # Remove cooling step
    energy = energy[ _np.unique(energy[:,0], return_index=True)[1] ] # Remove time duplicates
    energy[:,0] = crack_advance # Change from time to crack advance

    # Save results to file
    with open('global_energy.dat', 'w') as fout:
        fout.write('#crack_advance[mm] We[mJ]\n')
        _np.savetxt(fout, energy)

    # Clean data
    del _session.xyDataObjects['Global strain energy']


    # Compute stress vector components along interface
    step = 'Cooling'
    frame = odb.steps[step].frames[-1]
    _session.odbData[odb.name].setValues(activeFrames=(('Cooling', (-1, )), ))

    # Get stress field for profiles
    field = frame.fieldOutputs['S']

    # Save nodal data in copper side
    values = field.getSubset(
        region=instance.nodeSets['INTERFACE-NODES_SIDE1'],
        position=_visualization.ELEMENT_NODAL
    ).values

    T1 = _np.array([(0, value.data[1], value.data[3]) for value in values])[ranks]
    T1[:,0] = crack_advance

    with open('stress_vector-side1.dat', 'w') as fout:
        fout.write('#crack_advance[mm] Tn[MPa] Tt[MPa]\n')
        _np.savetxt(fout, T1)

    # Save nodal data in ceramic side
    values = field.getSubset(
        region=instance.nodeSets['INTERFACE-NODES_SIDE2'],
        position=_visualization.ELEMENT_NODAL
    ).values

    T2 = _np.array([(0, value.data[1], value.data[3]) for value in values])[ranks]
    T2[:,0] = crack_advance

    with open('stress_vector-side2.dat', 'w') as fout:
        fout.write('#crack_advance[mm] Tn[MPa] Tt[MPa]\n')
        _np.savetxt(fout, T2)

    odb.close()
```

C Source code of 3D DBC simulation

```
from abaqus import mdb as _mdb
import part as _part
import mesh as _mesh
import section as _section
import material as _material
import step as _step



class ceramic:
    length = 0.750 # mm
    thickness = 0.317 # mm
    young = 330000. #GPa
    poisson = 0.22 # -
    cte = 8.0e-6 # /dC

class copper:
    length = 0.650 # mm
    thickness = 0.127 # mm
    young = 127000. # MPa
    poisson = 0.33 # -
    cte = 16.5e-6 # /dC
    R0 = 43.2 # MPa
    C1 = 27000. # MPa
    D1 = 1850. # -
    C2 = 1120. # MPa
    D2 = 14.8 # -
    Q = 30. # MPa
    b = 12.8 # -

class interface:
    k = 1.e8 # N/mm
    Tmax = 150. # MPa
    Gc = 0.115 # N/mm
    viscosity = 0.001

class mesh:
    size = 0.010 # mm

class loads:
    class cooling:
        initial = 1000.0 # dC
        final = 0.0 # dC
        duration = 1.0 # s

    class cycles:
        mininal = -50.0 # dC
        maximal = 250.0 # dC
        duration = 1.0 # s (per cycle)
        count = 100 # cycles

name = 'simulation'
__model__ = None



```

```
57  def create():
58      '''
59      Create the simulation model in memory.
60
61      This function do not take any parameter but uses the module parameters
62      classes to generate the model.
63
64      Example
65      -------
66          >>> import model
67          >>> model.mesh.size = 0.01 # Change some parameters
68          >>> model.create()
69
70      Warning
71      -------
72          The model is regenerated at each call using current module parameters
73      '''
74      # Delete previous generation and create a new one
75      global __model__
76      if __model__ is not None:
77          _mdb.models.changeKey(fromName=__model__.name, toName='To-Delete')
78      model = _mdb.Model(name=name)
79
80      del _mdb.models['Model-1' if __model__ is None else 'To-Delete']
81
82
83      # Create the geometry
84      # - generate the ceramic part
85      sketch = model.ConstrainedSketch(name='sketch', sheetSize=1.0)
86      sketch.rectangle(
87          point1=(0., 0.),
88          point2=(ceramic.length, ceramic.thickness)
89      )
90
91      sample = model.Part(
92          name='Sample',
93          dimensionality=_part.THREE_D,
94          type=_part.DEFORMABLE_BODY
95      )
96      sample.BaseSolidExtrude(sketch=sketch, depth=ceramic.length)
97      del model.sketches['sketch']
98
99      # - generate the copper part
100     transform = sample.MakeSketchTransform(
101         sketchPlane=sample.faces[1],
102         sketchUpEdge=sample.edges[4],
103         sketchPlaneSide=_part.SIDE1,
104         sketchOrientation=_part.RIGHT,
105         origin=(0., ceramic.thickness, 0.)
106     )
107     sketch = model.ConstrainedSketch(
108         name='sketch',
109         sheetSize=1.0,
110         transform=transform
111     )
112     sketch.rectangle(
113         point1=(0., 0.),
114         point2=(copper.length, copper.length)
115     )
```

```
    sample.SolidExtrude(
        sketchPlane=sample.faces[1],
        sketchUpEdge=sample.edges[4],
        sketchPlaneSide=_part.SIDE1,
        sketchOrientation=_part.RIGHT,
        sketch=sketch,
        depth=copper.thickness
    )

    # - create partitions
    sample.PartitionCellByExtrudeEdge(
        line=sample.edges[2],
        cells=sample.cells,
        edges=(sample.edges[6], ),
        sense=_part.REVERSE
    )
    sample.PartitionCellByExtendFace(
        extendFace=sample.faces[4],
        cells=sample.cells.getSequenceFromMask(mask=('[#1 ]', ), )
    )
    sample.PartitionCellByExtendFace(
        extendFace=sample.faces[8],
        cells=sample.cells.getSequenceFromMask(mask=('[#2 ]', ), )
    )
    sample.PartitionCellByExtendFace(
        extendFace=sample.faces[12],
        cells=sample.cells.getSequenceFromMask(mask=('[#2 ]', ), )
    )


    # Create the assembly
    assembly = model.rootAssembly
    instance = assembly.Instance(name='Sample', part=sample, dependent=True)


    # Mesh the volume
    elem_type = _mesh.ElemType(elemCode=_mesh.C3D8, elemLibrary=_mesh.STANDARD)
    sample.setElementType(regions=(sample.cells,), elemTypes=(elem_type,))
    sample.setMeshControls(regions=sample.cells, elemShape=_mesh.QUAD)

    sample.seedPart(size=mesh.size)
    sample.generateMesh()

    # Create the interface mesh
    elem_count = len(sample.elements)
    elem_type = _mesh.ElemType(
        elemCode=_mesh.COH3D8,
        elemLibrary=_mesh.STANDARD,
        viscosity=interface.viscosity
    )
    region = _part.Region(
        side1Faces=sample.faces.getSequenceFromMask(mask=('[#2000 ]', ), )
    )
    sample.insertElements(faces=region)

    interface_set = sample.Set(
        name='INTERFACE',
        elements=sample.elements[elem_count:]
    )
```

```
175     sample.setElementType(regions=interface_set, elemTypes=(elem_type,) )
176
177
178     # Create sets
179     bulk_set = sample.Set(name='BULK', nodes=sample.nodes)
180     ceramic_set = sample.Set(name='CERAMIC', cells=sample.cells[:4])
181     copper_set = sample.Set(name='COPPER', cells=sample.cells[4:])
182
183     get_faces = sample.faces.getSequenceFromMask
184     sample.Set(name='X0', faces=get_faces(mask=('[#84800 ]', ), ))
185     sample.Set(name='Y0', faces=get_faces(mask=('[#400224 ]', ), ))
186     sample.Set(name='Z0', faces=get_faces(mask=('[#1008080 ]', ), ))
187
188
189     # Assign materials
190     # - ceramic elastic behavior
191     mat = model.Material(name='CERAMIC')
192     mat.Elastic(table=((ceramic.young, ceramic.poisson), ) )
193     mat.Expansion(table=((ceramic.cte, ), ))
194     model.HomogeneousSolidSection(name='Ceramic-Section', material='CERAMIC')
195     sample.SectionAssignment(region=ceramic_set, sectionName='Ceramic-Section')
196
197     # - copper elasto-plastic behavior
198     mat = model.Material(name='COPPER')
199     mat.Elastic(table=((copper.young, copper.poisson), ) )
200     mat.Expansion(table=((copper.cte, ), ))
201     mat.Plastic(
202         hardening=_material.COMBINED,
203         dataType=_material.PARAMETERS,
204         numBackstresses=2,
205         table=((copper.R0, copper.C1, copper.D1, copper.C2, copper.D2), )
206     )
207     mat.plastic.CyclicHardening(
208         parameters=True,
209         table=((copper.R0, copper.Q, copper.b), )
210     )
211     model.HomogeneousSolidSection(name='Copper-Section', material='COPPER')
212     sample.SectionAssignment(region=copper_set, sectionName='Copper-Section')
213
214     # - interface behavior
215     czm = model.Material(name='INTERFACE')
216     czm.Elastic(
217         table=((interface.k, interface.k, interface.k), ),
218         type=_material.TRACTION
219     )
220     czm.QuadsDamageInitiation(
221         table=((interface.Tmax, interface.Tmax, interface.Tmax), )
222     )
223     czm.quadsDamageInitiation.DamageEvolution(
224         table=((interface.Gc, ), ),
225         type=_material.ENERGY
226     )
227     model.CohesiveSection(
228         name='Int-Section',
229         material='INTERFACE',
230         response=_section.TRACTION_SEPARATION
231     )
232     sample.SectionAssignment(region=interface_set, sectionName='Int-Section')
233
```

```
234
235     # Create simulation steps
236     # - first cooling
237     step = model.StaticStep(
238         name = 'COOLING',
239         nlgeom = True,
240         previous='Initial',
241         initialInc=loads.cooling.duration/20.,
242         maxInc=loads.cooling.duration/20.,
243         maxNumInc=int(1e6),
244         minInc=loads.cooling.duration/1e30,
245         timePeriod=loads.cooling.duration
246     )
247     step.control.setValues(
248         allowPropagation=True,
249         resetDefaultValues=False,
250         timeIncrementation=(4.0, 8.0, 9.0, 16.0, 10.0, 4.0, 12.0, 50.0, 6.0, 3.0, 50.0)
251     )
252     model.fieldOutputRequests["F-Output-1"].setValues(
253         variables=('U', 'E', 'PE', 'PEEQ', 'S', 'SDEG', 'STATUS'),
254         timeInterval=loads.cooling.duration/10.
255     )
256
257     # - cycles
258     kwargs = dict(
259         nlgeom=True,
260         initialInc=loads.cycles.duration/10.,
261         maxInc=loads.cycles.duration/10.,
262         maxNumInc=int(1e6),
263         minInc=loads.cycles.duration/1e30,
264         timePeriod=loads.cycles.duration/2.
265     )
266
267     previous = 'COOLING'
268     for i in range(1, loads.cycles.count+1):
269         step = 'CYCLE'+str(i)
270         model.StaticStep(name=step+'_MIN', previous=previous, **kwargs)
271         model.StaticStep(name=step+'_MAX', previous=step+'_MIN', **kwargs)
272
273         previous = step+'_MAX'
274
275     if loads.cycles.count:
276         model.fieldOutputRequests['F-Output-1'].setValuesInStep(
277             stepName='CYCLE1_MIN',
278             timeInterval=loads.cycles.duration/2.
279         )
280         model.historyOutputRequests['H-Output-1'].setValuesInStep(
281             stepName='CYCLE1_MIN',
282             timeInterval=loads.cycles.duration/2.
283         )
284
285
286     # Create Boundary Conditions
287     model.DisplacementBC(
288         name='U1',
289         createStepName='Initial',
290         region=instance.sets['X0'],
291         u1=0.0
292     )
```

```
293     model.DisplacementBC(
294         name='U2',
295         createStepName='Initial',
296         region=instance.sets['Y0'],
297         u2=0.0
298     )
299     model.DisplacementBC(
300         name='U3',
301         createStepName='Initial',
302         region=instance.sets['Z0'],
303         u3=0.0
304     )
305 
306     # Apply temperature
307     field = model.Temperature(
308         name='TEMP',
309         createStepName='Initial',
310         region=instance.sets['BULK'],
311         magnitudes=(loads.cooling.initial, )
312     )
313     field.setValuesInStep(
314         stepName='COOLING',
315         magnitudes=(loads.cooling.final, )
316     )
317     for i in range(1, loads.cycles.count+1):
318         field.setValuesInStep(
319             stepName='CYCLE'+str(i)+'_MIN',
320             magnitudes=(loads.cycles.mininal, )
321         )
322         field.setValuesInStep(
323             stepName='CYCLE'+str(i)+'_MAX',
324             magnitudes=(loads.cycles.maximal, )
325         )
326 
327     # Store the model globaly
328     __model__ = model
329 
330 
331 def write_input():
332     '''
333     Write the simulation inp file from the created model
334 
335     Example
336     -------
337         >>> import model
338         >>> model.create()
339         >>> model.write_input()
340 
341     Raises
342     ------
343         RuntimeError if the model has not been generated
344     '''
345     if __model__ is None:
346         raise RuntimeError('Model not generated')
347 
348     job = _mdb.Job(name=__model__.name, model=__model__)
349     job.writeInput(consistencyChecking=False)
350     del _mdb.jobs[__model__.name]
```

References

[Ben Kaabar, 2015] Ben Kaabar, A. (2015). *Durabilité des assemblages céramique-métal employés en électronique de puissance.* Thesis, INSA de Lyon.

[Dugdale, 1960] Dugdale, D. S. (1960). Yielding of steel sheets containing slits. *Journal of The Mechanics and Physics of Solids*, 8:100–104.

[Gaiser et al., 2015] Gaiser, P., Klingler, M., and Wilde, J. (2015). Fracture mechanical modeling for the stress analysis of dbc ceramics. *2015 16th International Conference on Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics and Microsystems*, pages 1–6.