

**UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS**

Eduardo Junio do Lago Batista

**Determinação das propriedades efetivas de materiais compósitos a
partir de microestruturas artificiais**

São Carlos

2019

Eduardo Junio do Lago Batista

Determinação das propriedades efetivas de materiais compósitos a partir de microestruturas artificiais

Monografia apresentada ao Curso de Engenharia Aeronáutica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Aeronáutico.

Orientador: Prof. Dr. Ricardo Afonso Angélico

São Carlos

2019

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO,
POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS
DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da
EESC/USP com os dados inseridos pelo(a) autor(a).

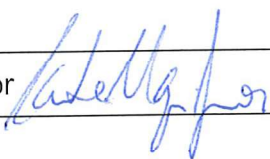
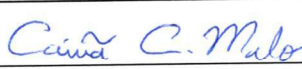
BB333d
d Batista, Eduardo Junio do Lago
 Determinação das propriedades efetivas de
materiais compósitos a partir de microestruturas
artificiais / Eduardo Junio do Lago Batista; orientador
Ricardo Afonso Angélico. São Carlos, 2019.

Monografia (Graduação em Engenharia Aeronáutica)
-- Escola de Engenharia de São Carlos da Universidade
de São Paulo, 2019.

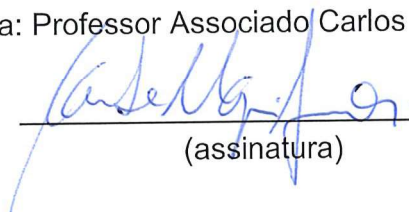
1. Propriedades mecânicas efetivas. 2.
Microestruturas artificiais. 3. Método dos elementos
finitos. 4. Volumes Elementares Representativos. I.
Título.

FOLHA DE APROVAÇÃO

Candidato: Eduardo Junio do Lago Batista
Título do TCC: Determinação das propriedades efetivas de materiais compósitos a partir de microestruturas artificiais
Data de defesa: 29/11/2019

Comissão Julgadora	Resultado
Professor Associado Carlos De Marqui Junior 	Aprovado
Instituição: EESC - SAA	
Pesquisador Caiuã Caldeira de Melo 	Aprovado
Instituição: DEMa - UFSCar	

Presidente da Banca: Professor Associado Carlos De Marqui Junior


(assinatura)

AGRADECIMENTOS

Ao concluir o Ensino Médio em escola pública muitas incertezas pairavam sobre meu futuro. Entre estas, a mais significativa fosse, talvez, a de se eu conseguiria ingressar em um curso superior e realizar um sonho que carregava por anos: o de estudar em uma universidade de ponta em um curso de excelência. Foram muitos os sacrifícios que teriam de ser enfrentados, mas foram muitos os amigos e incentivadores que encontrei durante a caminhada.

Primeiramente devo agradecer à USP, por ter possibilitado e financiado o meu ingresso e permanência durante todo este tempo; aos professores que, mesmo exigentes, sempre se mostraram compreensíveis e disponíveis; ao corpo técnico, que possibilitou o aprendizado em tantas ocasiões quanto foram possíveis.

Devo lembrar em especial de meus pais que, apesar de toda a limitação financeira, permitiram me aventurar por este mundo até então desconhecido; da senhora Lígia Saiki que foi uma das minhas maiores, senão a maior, incentivadoras; do professor Ricardo Afonso Angélico que sempre se dispôs a me ajudar em todos os momentos em que precisei, por toda a sua orientação e aconselhamento. Devo ainda agradecer a meus amigos de infância que acreditaram em minha capacidade em todos os momentos; aos meus amigos de faculdade que, apesar do pouco tempo de convívio, tornaram esta uma das jornadas mais divertidas e memoráveis de minha vida.

Por fim devo agradecer à minha namorada, Midori, por sempre se preocupar com minha saúde e por ter se tornado um dos pilares de meu desenvolvimento.

A todos os que cruzaram meu caminho nos últimos anos, muito obrigado.

“O sucesso é ir de fracasso em fracasso sem perder entusiasmo.”

Winston Churchill

RESUMO

Batista, E. J. L. **Determinação das propriedades efetivas de materiais compósitos a partir de microestruturas artificiais.** 2019. 76p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2019.

A utilização de materiais compósitos na indústria está relacionada ao fato de estes materiais possuírem propriedades mecânicas, térmicas ou químicas efetivas superiores àquelas das fases constituintes isoladas para uma dada aplicação de interesse. Entretanto, geralmente, estes materiais possuem um comportamento mecânico mais complexo, dadas as diferenças entre propriedades físicas, químicas e mecânicas de suas fases. Quando os materiais constituintes possuem coeficientes de expansão térmica diferentes entre si, ocorre o surgimento de esforços internos na estrutura do material compósito quando este é submetido a uma variação de temperatura, principalmente nas regiões de interface entre as fases constituintes. Estas regiões, então, se tornam mais susceptíveis à nucleação e propagação de trincas, podendo culminar na falha de componentes constituídos de tais materiais. O comportamento destes materiais pode ser simulado numericamente, fornecendo uma resposta representativa do comportamento observado experimentalmente. Neste contexto, o objetivo da presente monografia é a geração de um experimento numérico, realizado a partir do método dos elementos finitos, que seja capaz de incorporar os efeitos térmicos à resposta mecânica e gerar resultados estatisticamente relevantes para as propriedades efetivas de materiais compósitos. Para tal, foram analisados três modelos de microestruturas artificiais de materiais compósitos do tipo inclusões esféricas de alumina (Al_2O_3) imersas em matriz vítrea (G1, G2, G3); dispostas em arranjos regulares do tipo Cúbico, Cúbico de Corpo Centrado e Cúbico de Face Centrada; e com proporções volumétricas de 15%, 30% e 45%. Cada um destes modelos apresenta particularidades com relação às propriedades mecânicas da matriz vítrea bem como à diferença entre coeficientes de expansão térmica entre matriz e inclusões. Foram geradas rotinas automatizadas em *python* possibilitando a reprodução dos experimentos numéricos de forma rápida e intuitiva. Os resultados gerados pela análise em MEF foram, então, aplicados em uma técnica de homogeneização para a extração das propriedades efetivas e, posteriormente, comparados com resultados teóricos e experimentais apresentados na literatura. Os campos de tensão, deformação e as propriedades efetivas resultantes observados apresentaram grande proximidade com os resultados experimentais usados para a validação do método, com a ressalva de que os modelos numéricos não apresentam efeitos de trincas, presentes nos experimentos. Os resultados do presente trabalho se apresentam como uma abordagem inicial para a investigação dos fenômenos de surgimento de esforços internos em materiais compósitos submetidos a variação de temperatura, servindo como comparativo tanto para futuros estudos experimentais como numéricos.

Palavras-chave: propriedades mecânicas efetivas, microestruturas artificiais, comportamento mecânico, incompatibilidade de propriedades físicas.

ABSTRACT

Batista, E. J. L. **Determination of effective properties of composite materials from artificial microstructures**. 2019. 76p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2019.

The application of composite materials in the industry is related to the fact that this kind of material presents superior mechanical, thermal, or chemical properties when compared to those of the separate constitutive phases. However, in general, these materials exhibit a more complex mechanical behavior, given the differences between the physical, chemical, and mechanical properties of their phases. When the constitutive materials have different thermal expansion coefficients, internal stress appears on the composite material structure when this material is submitted to a temperature variation, especially on the interface of its constitutive phases. These regions become more susceptible to the nucleation and propagation of cracks, which can culminate in the failure of components. The behavior of these materials can be numerically simulated, predicting a material's representative response, such as the one observed in experiments. The goal of the present monography is to generate a numerical experiment based on the finite element method, which includes the thermal effects into the mechanical response and compute statistically relevant results for the effective properties of composite materials. For this, three artificial microstructures models were analyzed, all of them composed of alumina (Al_2O_3) spherical inclusions dispersed inside a glass matrix (G1, G2, G3); disposed in regular arrangements of the kind primitive Cubic, Body-Centered Cubic or Face-Centered Cubic; and with volume fractions of 15%, 30% and 45%. Automated *python* scripts were generated to make it easier to reproduce the numerical experiments in a fast and intuitive way. The results acquired by the FEM analysis were applied to a homogenization technique for the extraction of the effective properties of the models and, afterward, compared to theoretical and experimental results presented on reference literature. The stresses and deformation fields, as well as the macroscopic properties computed, present significant agreement to the experimental results in the region without a majority of cracks. The numerical models do not account for the crack effects; thus, it is applicable only without the presence of a considerable amount of cracks. The results of the current work are presented as an initial approach for the investigation of the phenomena of nucleation of internal stress in composite materials submitted to a temperature gradient and serve as a comparative baseline for future studies, both experimental and numerical.

Keywords: effective mechanical properties, artificial microstructures, mechanical behavior, properties mismatch.

LISTA DE FIGURAS

Figura 1 – Aplicações de materiais compósitos na indústria. Fonte: [Zago, 2019].	22
Figura 2 – Composição dos materiais utilizados na construção de aeronaves Boeing [Zhang et al., 2018].	22
Figura 3 – Trincas em uma matriz metálica de titânio reforçada com fibras de carvão de silício (créditos: J G Robinson and British Petroleum) [King, 2018].	23
Figura 4 – Classificação dos materiais heterogêneos. Apresenta tanto os materiais reais quanto os modelos virtuais associados. Fonte: [Bargmann et al., 2018]	28
Figura 5 – Diferentes geometrias para as inclusões da fase dispersa. Fonte: [Bargmann et al., 2018]	31
Figura 6 – Modelo com a nomenclatura utilizada para o desenvolvimento das equações de periodicidade. Fonte: [Fan and Wang, 2017]	33
Figura 7 – Metodologia adotada para o desenvolvimento do trabalho de conclusão de curso.	37
Figura 8 – Evolução do Módulo de Young e expansão térmica dos materiais constituintes. Fonte: [Tessier-Doyen et al., 2006]	39
Figura 9 – Modelos computacionais gerados de acordo com o arranjo microestrutural e a fração volumétrica. Inclusões de Alumina e matriz G2.	40
Figura 10 – Fluxograma do algoritmo de automatização da geração dos modelos	42
Figura 11 – Fluxograma de execução dos algoritmos em forma de <i>plug-ins</i> internos ao ambiente Abaqus®	43
Figura 12 – Geração da oitava parte do modelo CFC-30	45
Figura 13 – Modelo CFC-30 resultante, já com a aplicação da malha simétrica de elementos finitos	46
Figura 14 – Aplicação das condições de periodicidade ao modelo CFC-30, de matriz G3 .	46
Figura 15 – Campo residual de tensões devido à variação de temperatura de 470°C a 25°C aplicada ao modelo CFC-30, matriz G3	47
Figura 16 – Tensão máxima principal resultante para casos de carga de deformação global aplicados nos <i>dummy nodes</i>	48
Figura 17 – Comparativo entre os resultados da modelagem MEF com os teóricos e experimentais de [Tessier-Doyen et al., 2006] para os modelos de matriz G3 e $\phi = 15\%$	49
Figura 18 – Comparativo entre os resultados da modelagem MEF com os teóricos e experimentais de [Tessier-Doyen et al., 2006] para os modelos de matriz G3 e $\phi = 30\%$	50
Figura 19 – Comparativo entre os resultados da modelagem MEF com os teóricos e experimentais de [Tessier-Doyen et al., 2006] para os modelos de matriz G3 e $\phi = 45\%$	50
Figura 20 – Comparativo entre os resultados da modelagem MEF com os teóricos e experimentais [Tessier-Doyen et al., 2006] para os modelos de matriz G1 e $\phi=30\%$.	51
Figura 21 – Comparativo entre os resultados da modelagem MEF com os teóricos e experimentais [Tessier-Doyen et al., 2006] para os modelos de matriz G2 e $\phi=30\%$.	51

LISTA DE TABELAS

Tabela 1 – Erros percentuais entre os valores obtidos pela simulação MEF e os valores teóricos para a matriz G3.	52
--	----

LISTA DE SÍMBOLOS

–	Alfabeto Latino
$\mathbf{C}$	Tensor constitutivo de rigidez
E	Módulo de elasticidade
VRE	Volume Representativo Elementar
CUR	Célula Unitária Repetitiva
MEF	Método dos Elementos Finitos
T	Temperatura
u	Deslocamento horizontal
v	Deslocamento vertical
V_f	Volume da fase
V	Volume total do compósito
–	Alfabeto grego
α	Coeficiente de expansão térmica
ϵ	Tensor de deformações
ν	Coeficiente de Poisson
σ	Tensor de tensões
Φ	Fração volumétrica
U	Energia interna de deformação
–	Sistemas de coordenada
(x, y, z)	Coordenadas cartesianas

SUMÁRIO

	Lista de tabelas	15
	1 INTRODUÇÃO	21
1.1	Objetivos	24
1.2	Visão geral do documento	24
	2 REVISÃO BIBLIOGRÁFICA	27
2.1	Materiais compósitos e volumes elementares representativos	27
2.2	Geração de microestruturas artificiais	29
2.3	Métodos dos elementos finitos aplicados na identificação de propriedades efetivas	32
	3 METODOLOGIA	37
3.1	Geração dos modelos de volumes representativos elementares	38
3.2	Organização das rotinas de geração dos modelos de microestruturas	42
	4 RESULTADOS	45
4.1	Apresentação dos modelos resultantes	45
4.2	Comparação com dados experimentais e numéricos	49
	5 CONCLUSÕES E PERSPECTIVAS	53
	REFERÊNCIAS	55
	A CÓDIGO EM PYTHON PARA A GERAÇÃO DAS GEOMETRIAS MODELO E APLICAÇÃO DAS CONDIÇÕES DE PERIODICIDADE	59
A.1	Cell	59
A.2	Mirror	66
A.3	Constraints	67
A.4	Analysis	74
1.5	Report	75

CAPÍTULO 1

INTRODUÇÃO

A evolução da história humana demonstra a importância que os materiais e a sua manipulação representam para vida cotidiana, sendo tão relevantes que mesmo as eras são nomeadas a partir dos materiais mais utilizados para a confecção de ferramentas, como a idade do bronze ou a do ferro [Ashby, 2005]. O desenvolvimento da ciência dos materiais possibilitou que os materiais utilizados pelos homens fossem melhor estudados e analisados, tendo suas propriedades determinadas em maior detalhe e direcionando as suas aplicações para atividades e processos onde fossem melhor explorados. Desta forma, os materiais foram classificados como metais, cerâmicas e polímeros, a depender de suas propriedades e formas físicas.

A combinação de materiais é antiga e precede a denominação de materiais compósitos, sendo exemplos a utilização de concretos [Sparavigna, 2011]. Somente no final da Segunda Guerra Mundial, entretanto, os materiais compósitos poliméricos começaram a ser utilizados na indústria, com aplicações na fabricação dos primeiros radomes de alojamento dos equipamentos eletrônicos de radar. Também na indústria aeronáutica tem-se a produção de alguns componentes a partir de polímeros reforçados por fibra de vidro. Entretanto esses materiais eram extremamente caros na época, impossibilitando sua aplicação em larga escala [Hollaway, 2000].

Desde então, o estudo e utilização destes materiais em diversos ramos da indústria possibilitou a ampliação do conhecimento sobre as propriedades destes materiais bem como sobre seus processos de fabricação. Tudo isto impulsionou a diversificação da aplicação dos materiais compósitos, como pode ser visto na Figura 1. Em especial, a indústria aeronáutica apresenta uma crescente utilização destes materiais na composição de componentes das aeronaves, como pode ser observado na Figura 2. Esta migração ao uso de materiais compósitos pode ser atribuída às suas relações resistência/peso superiores (mecânica, à corrosão e fadiga). Dadas tais vantagens, atualmente existem aeronaves que apresentam constituição em massa de materiais compósitos superior ao de materiais metálicos, caso do Boeing 787 Dreamliner [Zhang et al., 2018].

Além disto, materiais compósitos de matriz cerâmica apresentam capacidade comprovada de aplicação sob condições operacionais de alta temperatura (acima de 1400°C), por exemplo, em estruturas de bocais de exaustão de foguetes e motores de aeronaves, escudos de reentrada

atmosférica e bordos de ataque de aeronaves supersônicas. Tais estruturas demandam de seus materiais componentes propriedades de alta resistência térmica e mecânica [Zhang et al., 2018]



Figura 1 – Aplicações de materiais compósitos na indústria. Fonte: [Zago, 2019].

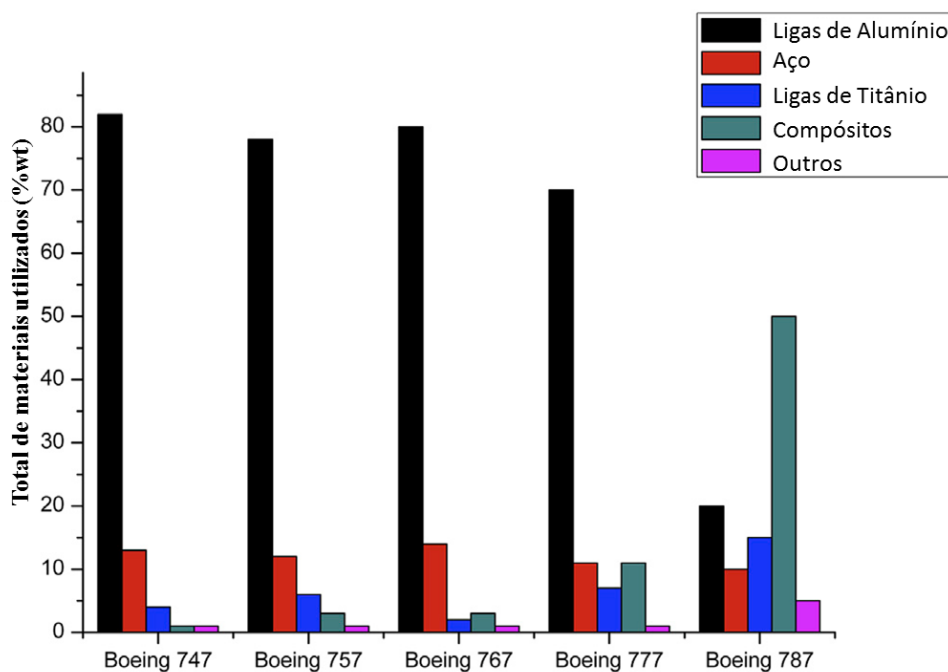


Figura 2 – Composição dos materiais utilizados na construção de aeronaves Boeing [Zhang et al., 2018].

Entretanto, a utilização de materiais compósitos formados por materiais de diferentes propriedades físicas implica em certas dificuldades. Cada uma das fases constituintes destes mate-

riais compósitos reagem a estímulos térmicos ou mecânicos de acordo com as suas características intrínsecas. Tais fenômenos são responsáveis pela geração de tensões internas no material, as quais ao atingir determinado nível, podem culminar na falha do componente. Como exemplo pode-se citar o fenômeno de que trata o estudo deste texto, as tensões internas geradas pela variação de temperatura em materiais compósitos por fases constituintes de diferentes coeficientes de expansão térmica. Diante das inúmeras possibilidades de criação de novos materiais compósitos, é fundamental o uso de ferramentas computacionais que permitam simular diferentes cenários, desde o projeto do material à avaliação do comportamento desse mediante as condições de serviço.

O desenvolvimento de métodos numéricos mais eficientes, bem como de computadores mais robustos, permite o desenvolvimento de ferramentas computacionais voltadas à determinação das propriedades dos materiais, utilizando modelagem via método dos elementos finitos. Este tipo de modelagem se mostra especialmente importante para o estudo de casos onde a aplicação de tais materiais é de alta complexidade de reprodução experimental. Durante as últimas décadas, este tipo de modelagem numérica apresentou grande desenvolvimento, especialmente no estudo dos efeitos termo-mecânicos em materiais compósitos [Luchini et al., 2016, Luchini et al., 2017]. Claramente, é indispensável que os resultados obtidos por tais ferramentas computacionais sejam validados a partir de resultados experimentais.

Na Figura 3, é representada uma falha em um compósito de titânio reforçado com fibras de carbetto de silício. Este material possui aplicação em estruturas aeronáuticas, mísseis e motores. De acordo com King, essas falhas ocorrem tanto por tensões térmicas residuais do processo de fabricação, quanto por tensões resultantes do componente em aplicação [King, 2018].

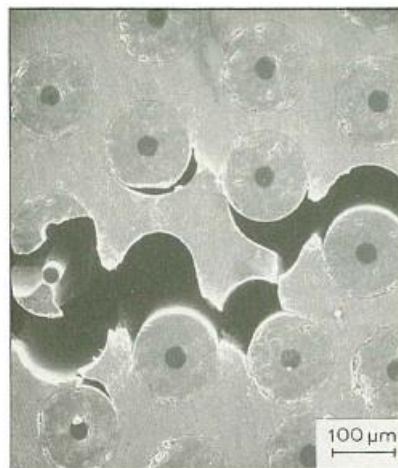


Figura 3 – Trincas em uma matriz metálica de titânio reforçada com fibras de carbetto de silício (créditos: J G Robinson and British Petroleum) [King, 2018].

Os efeitos da variação de temperatura nas propriedades mecânicas de materiais compósitos têm sido estudados por décadas sendo o compósito mais comumente utilizado o de matriz vítrea, dada a facilidade de se observar a região de interface entre a inclusão e a matriz. Este tipo de material, apesar de apresentar certas dificuldades de experimentação, devido à grande velocidade com que se transformam do estado íntegro para o fraturado [Davidge and Green, 1968] é capaz de enaltecer o fenômeno físico de concentração de tensão estudados, reduzindo a influência de outros

fenômenos secundários, sendo chamados de materiais modelo [Stoneham and Harding, 2009]. Estes materiais geram informações importantes sobre a relação entre a escala microscópica e a escala macroscópica de aplicação.

É significativo o número de trabalhos acadêmicos dedicados à modelagem computacional para o estudo do comportamento dos materiais sujeitos a carregamentos térmicos. Entretanto, nota-se que estes trabalhos muitas vezes focam na discussão sobre as condições que devem ser aplicadas aos modelos, sejam de contorno ou periodicidade, ou sobre os resultados obtidos por tais simulações, deixando-se de lado as questões de implementação e de utilização das ferramentas computacionais. Dessa maneira, propôs-se a produção de um algoritmo utilizando o método dos elementos finitos que fosse capaz de reproduzir modelos tridimensionais, a aplicação das condições de contorno e periodicidade e realizasse a solução do sistema para o caso de estudo proposto. Teve-se por finalidade a geração de uma ferramenta que fosse capaz de auxiliar nos estudos sobre o tema, sendo robusta e de fácil utilização, contribuindo, portanto, ao processo de obtenção de informações sobre o surgimento de tensões internas em materiais compósitos sujeitos a carregamentos térmicos.

1.1 Objetivos

O desenvolvimento deste trabalho de conclusão de curso visa o desenvolvimento da ferramenta computacional para a geração das microestruturas artificiais tridimensionais a partir de características físicas do material, como a porosidade, fração volumétrica, disposição e tamanho dos grãos, etc. De maneira mais específica, pode-se dividir os objetivos da seguinte maneira:

- Familiarizar-se e aprender a ferramenta de elementos finitos Abaqus® ;
- Gerar as geometrias regulares de estudo segundo os dados apresentados por Tessier et al. [Tessier-Doyen et al., 2006];
- Estudar e discutir as condições de contorno a serem aplicadas ao modelo para garantir a sua periodicidade;
- Desenvolver o algoritmo automatizado para a geração das microestruturas artificiais tridimensionais;
- Realizar estudo comparativo dos resultados obtidos com dados da literatura.

1.2 Visão geral do documento

De forma a facilitar o acompanhamento e compreensão deste trabalho, o texto foi dividido em seções, apresentadas a seguir.

No Capítulo 1, Introdução, foram apresentados o contexto de inserção deste trabalho, o fenômeno de do comportamento de materiais compósitos submetidos a variações de temperatura, a necessidade de criação de um algoritmo computacional que possa ser utilizado para estudar este fenômeno. Foi-se apresentado também um breve panorama atual acerca do tema, a importância de se estudar estes fenômenos que se relacionam intrinsecamente ao surgimento de trincas em materiais e à sua consequente falha, levando-se à exposição dos objetivos propostos.

No Capítulo 2 será apresentada a fundamentação teórica necessária ao entendimento

do presente texto juntamente com a formulação matemática da modelagem e dos processos realizados. Será dada uma atenção especial às definições de materiais compósitos e volumes elementares representativos e sua relação com a representatividade estatística das propriedades dos materiais; às técnicas de geração de microestruturas artificiais com ênfase aos métodos geométricos, utilizados no presente trabalho; e à aplicação dos métodos dos elementos finitos na identificação das propriedades efetivas dos materiais, com foco no desenvolvimento do equacionamento das condições de contorno e periodicidade e nas técnicas de homogeneização utilizadas.

No Capítulo 3 serão apresentadas os passos utilizados para a geração dos modelos, a aplicação das propriedades físico/mecânicas das fases constituintes, os tipos de arranjo geométricos utilizados e a aplicação das condições de contorno e periodicidade na formulação em elementos finitos. Também será apresentada a metodologia de automatização utilizada para a geração dos modelos e aplicação dos campos de esforços térmicos e mecânicos.

No Capítulo 4 serão apresentados os resultados obtidos pela implementação de todos os algoritmos. Será elaborado uma descrição de exemplo para a geração de uma micro-estrutura, a geração da malha em elementos finitos correspondente, a aplicação das condições de periodicidade e os resultados qualitativos obtidos tanto do campo de tensões devido ao carregamento térmico quanto aos campos de tensão devido aos carregamentos mecânicos. Na segunda parte serão desenvolvidos os resultados numéricos para a computação das propriedades efetivas dos modelos simulados e a sua comparação com os resultados experimentais e teóricos apresentados pela literatura de interesse.

Por fim, no Capítulo 5 são apresentadas as conclusões obtidas com a realização do presente trabalho, bem como perspectivas e orientações para futuros trabalhos relacionados à esta mesma área de desenvolvimento.

CAPÍTULO 2

REVISÃO BIBLIOGRÁFICA

O desenvolvimento e aperfeiçoamento das técnicas de manufatura possibilitam a criação de novos materiais a partir de diferentes combinações das fases constituintes. Tais combinações permitem alcançar um desempenho termomecânico superior quando comparado com as propriedades dos materiais constituintes isoladamente. Essas características, entretanto, dependem diretamente da forma em que é organizado o arranjo microestrutural do novo material, tais como a disposição e tamanho dos grãos, o nível de porosidade, a composição de fases combinadas, etc. Desta forma, se faz necessário o desenvolvimento de técnicas e metodologias para caracterizar as propriedades mecânicas efetivas apresentadas pelos materiais com sua composição microestrutural [Bargmann et al., 2018].

Neste capítulo serão abordados os três temas de maior relevância para a elaboração deste trabalho. Esta apresentação se faz necessária para um bom entendimento dos procedimentos realizados bem como das hipóteses adotadas e suas correspondentes justificativas. Inicialmente se apresentará os conceitos de volumes elementares representativos, seguido dos procedimentos de geração de microestruturas artificiais e, por último, a aplicação dos métodos dos elementos finitos na identificação das propriedades efetivas. Parte desta fundamentação teórica será consolidada, por meio de equacionamento, na Seção de Metodologia.

2.1 Materiais compósitos e volumes elementares representativos

De forma geral, todos os materiais podem ser classificados como heterogêneos, a depender somente da escala de observação. Materiais heterogêneos são materiais formados por fases de diferentes materiais, como um compósito, ou por um mesmo material em diferentes estados, como um material policristalino. Estes materiais podem ser divididos em porosos e não porosos. Exemplos de materiais porosos são as cerâmicas e tecidos, enquanto que para materiais não-porosos têm-se os materiais compósitos do tipo matriz-inclusão (inclusões particulares ou fibrosas). A combinação de diferentes fases pode originar materiais com propriedades mecânicas, térmicas e acústicas de interesse de inúmeras aplicações de engenharia. Na Figura 4 é apresentada a classificação dos materiais heterogêneos em dois grandes grupos (não-porosos e porosos). Além

disso, apresenta uma representação da microestrutura do material heterogêneo, tanto do material real quanto do material virtual associado.

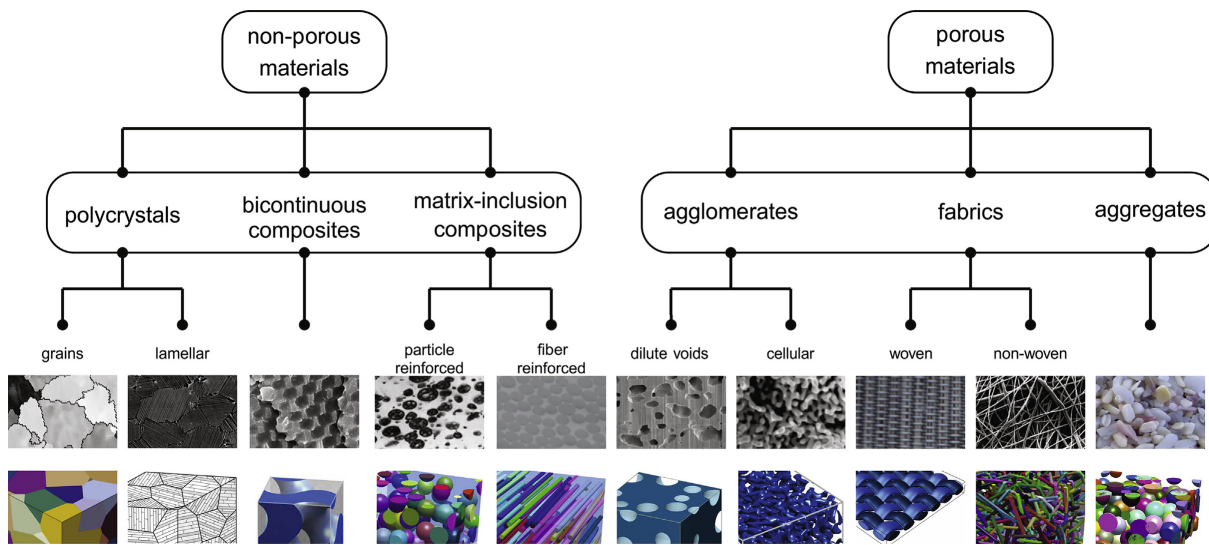


Figura 4 – Classificação dos materiais heterogêneos. Apresenta tanto os materiais reais quanto os modelos virtuais associados. Fonte: [Bargmann et al., 2018]

Materiais compósitos são materiais multifásicos formados artificialmente que possuem fases constituintes com propriedades químicas distintas e interface bem destacada. Muitos destes materiais são compostos por duas fases, a matriz (contínua) e a fase dispersa, onde as características da fase dispersa, como geometria das partículas, tamanho, orientação; são predominantes na determinação das propriedades do material resultante [Callister, 2008]. Estes materiais são os exemplos mais evidentes de materiais heterogêneos e apresentam crescente interesse de aplicação nas diversas áreas da engenharia devido à sua relação resistência/peso superior em relação aos materiais tradicionais. Devido a este interesse crescente de utilização, estudos sobre o comportamento destes materiais têm sido bastante desenvolvidos mesmo apresentando certas dificuldades como a multidisciplinaridade envolvida e a complexidade das incertezas enfrentadas [Bargmann et al., 2018, Torquato, 2013].

Duas formas de abordagem desta problemática estão disponíveis, sendo elas a abordagem experimental e a abordagem computacional. A primeira, apesar de apresentar resultados mais confiáveis (desde que bem executada), sendo até mesmo a metodologia de validação de diversas ferramentas computacionais, apresenta-se como uma solução dispendiosa, tanto financeiramente quanto em relação ao tempo necessário para a sua execução. Já a segunda abordagem traz a vantagem de diminuição de custos financeiros e temporais. Como o problema analisado consiste na investigação de diversas configurações microestruturais, a abordagem experimental acaba por se tornar impraticável, haja visto o grande número de variáveis envolvidas nas combinações.

Para que seja possível a execução de estudo a partir das ferramentas computacionais é necessário, então, a utilização de uma formulação micromecânica, a qual seja capaz de descrever detalhes do arranjo interno estrutural de um material heterogêneo, considerando cada fase constituinte como um meio contínuo. Desta formulação, surge o conceito de volume representativo elementar (VRE) ou célula unitária. Um VRE é um modelo volumétrico cujo comportamento é

representativo do material como um todo, sendo grande o suficiente para capturar as propriedades do material, de um ponto de vista estatístico, a partir da equivalência entre as condições de contorno de tensão e deslocamento [Aboudi et al., 2013, Drago and Pindera, 2007]. Surge junto desta definição, outro conceito igualmente importante e necessário, o de célula unitária repetitiva. Esta é uma aproximação utilizada quando se trata de materiais heterogêneos, onde a sua microestrutura é admitida como periódica, sem perda de generalidade estatística, e a estrutura do material é formada pela repetição destas células de forma ordenada, com a combinação de condições periódicas de tensão e deslocamento. Apesar destes dois conceitos serem formalmente distintos, seus significados têm se confundido quando se tratam de materiais compósitos [Drago and Pindera, 2007].

Como forma de garantir representatividade estatística, o VRE deve satisfazer a condição de separação de escalas $\mathcal{L}^\mu \ll \mathcal{L}^{VRE} \ll \mathcal{L}^M$, onde $\mathcal{L}^\mu$ denota a microescala, ou comprimento característico associado com elementos microestruturais, como o tamanho das inclusões; $\mathcal{L}^{VRE}$, a mesoescala ou tamanho do VRE; e $\mathcal{L}^M$ denota a macroescala ou tamanho característico do corpo macroscópico. Esta condição existe para garantir que o modelo de VRE seja estruturalmente típico do material na sua média e que o VRE contenha número suficiente de inclusões para que o módulo aparente de comportamento seja independente de tensões e deslocamentos superficiais [Ostoja-Starzewski, 2006]. Esta última condição é traduzida justamente na aplicação das condições de contorno, a serem apresentadas posteriormente.

Materiais compósitos do tipo matriz-inclusão são caracterizados pela não-sobreposição de partículas imersas em uma matriz interconectada, onde estas inclusões têm a função de melhorar propriedades mecânicas do material como um todo. No entanto, essa capacidade de melhoria tem estrita relação com a caracterização microestrutural do material [Bargmann et al., 2018]. Tendo em vista o exposto sobre VRE, a sua modelagem numérica depende, então, principalmente do arranjo geométrico adotado para a distribuição da fase dispersa na matriz, da fração volumétrica entre as fases constituintes e das propriedades das fases constituintes. O arranjo geométrico a ser modelado tem sua origem no tipo de material compósito estudado, de forma a se garantir as propriedades de periodicidade e de representatividade estatística. Para compósitos reforçados por partículas, estes arranjos ainda podem ser modelados de duas formas, arranjos regulares (possuem distribuição bem definida das inclusões na matriz) ou aleatórios (possuem distribuição aleatória das inclusões na matriz). A fração volumétrica é definida como a fração do volume ocupado pela fase dispersa pelo volume total do VRE ($\Phi = V_f/V$). Já as propriedades dos materiais constituintes está relacionado com a escolha dos materiais a serem estudados.

2.2 Geração de microestruturas artificiais

A abordagem computacional deve levar em consideração a formulação matemática e geométrica envolvida em sua constituição, para que os modelos simulados sejam representativos da realidade, assegurando a confiança dos resultados obtidos. Assim, pode-se seguir três metodologias de representação [Bargmann et al., 2018], reconstrução da microestrutura a partir de dados experimentais, geração da microestrutura baseada em física e a partir de métodos geométricos.

O primeiro método utiliza-se de técnicas de processamento de imagens por contraste,

seja pela utilização de tomografia de raio-X, espectroscopia por energia dispersiva de raio-X ou por tomografia de prova atômica. Estes métodos, entretanto, possuem dificuldades relacionadas ao comportamento diferente de cada fase constituinte do material, já que as técnicas de corte e de fotografia devem ser adequadas para todos os componentes. Estas técnicas de fotografia devem, muitas vezes, ter sua capacidade de identificação de contraste ampliada, com a utilização de métodos de pré-processamento, como a cobertura do material por finas camadas de material condutor, que podem acabar por obstruir a imagem por contraste [Bargmann et al., 2018].

Para a modelagem 3-D, é utilizada uma combinação de métodos de fotografia e de seccionamento, geralmente montados de forma automatizada para reduzir erros e tempo de execução. As técnicas mais avançadas utilizam de técnicas de microscopia ótica combinadas com técnicas de abrasão, polimento ou ultramicrotomia. Mesmo a utilização de diferentes processos de remoção de materiais, toda a modelagem sofre com as grandes diferenças nas propriedades dos materiais (em especial a dureza), o que acaba afetando os resultados obtidos. Um exemplo é o caso dos compósitos de matriz polimérica reforçados com partículas cerâmicas, onde a remoção do material é altamente irregular [Bargmann et al., 2018]. Posteriormente, ainda é necessário o tratamento dos dados coletados pelas fotografias e sua transposição de espaço dimensional, do 2-D para o 3-D. Assim, apesar deste método ser o mais chamativo em relação à fidedignidade da modelagem com a realidade, ainda estão presentes as incertezas experimentais, além do alto custo de realização de tais experimentos.

O segundo método é caracterizado pela utilização de simulações sobre a locomoção de elementos discretos da fase dispersa, resultando na distribuição final desta fase no material. De forma geral, a modelagem de processos físicos envolvidos na geração de microestruturas do tipo matriz-inclusão é de difícil execução, dado a complexidade das simulações, que demandam muito tempo para serem concluídas. O que se usa, então, são métodos inspirados em processos físicos, como os de dinâmica molecular, simulação de crescimento, movimento de partículas. A utilização destes métodos tem por base a aplicação de força de gravidade sobre o processo de densificação do material, onde as partículas são geradas arbitrariamente no interior da matriz e a gravidade é a responsável pelo posicionamento das partículas, uma sobre as outras, até se atingir uma posição de equilíbrio. Estes métodos podem ser denominados como contração mecânica [Williams and Philipse, 2003] ou modelo de queda-e-rolagem [VISSCHER and BOLSTERLI, 1972]. Neste processo, condições de contato ou ligação entre agregados são tão importantes quanto o tamanho e distribuição de cada partícula. As formas dos particulados são idealizados como poliedros tridimensionais e, durante todo o processo, estas partículas interferem entre si gerando VRE's próximos a um verdadeiro pacote de inclusões. Por isso, ocorre o surgimento de anisotropias na direção vertical.

Por fim, tem-se os métodos geométricos, que serão aqueles utilizados no decorrer deste trabalho. Este tipo de metodologia tem longa história na geração de microestruturas artificiais de materiais do tipo matriz-inclusão. Neste caso, uma das questões a serem endereçadas tem o objetivo de definir a geometria das inclusões a serem utilizadas na modelagem, impactando diretamente nas propriedades do material final obtido e nos resultados das simulações realizadas. De forma geral, pode-se destacar algumas formas usuais utilizadas, a depender do objetivo de estudo. Na Figura 5 são apresentados estes tipos de geometria utilizados. Cabe destacar que as

inclusões do tipo esféricas são isotrópicas e não possuem orientação, sendo descritas somente pelo valor de seu raio e posição do seu centro. Inclusões do tipo elipsoidal, cilíndrico, cápsula e poliédricos são naturalmente anisotrópicos, e sua modelagem necessita da orientação, além de suas dimensões. Para todos estes tipos de geometria deve-se garantir que não exista a sobreposição das inclusões, fato que deve ser levado em conta e ter-se bastante cuidado quando da modelagem de microestruturas aleatórias.

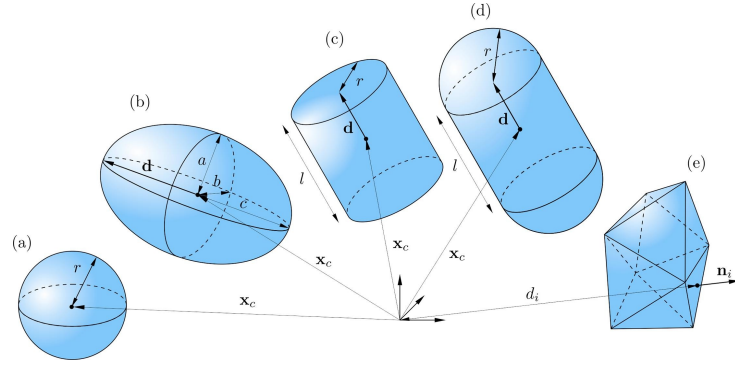


Figura 5 – Diferentes geometrias para as inclusões da fase dispersa. Fonte: [Bargmann et al., 2018]

Em geral, existem dois tipos de classes para sistemas do tipo matriz-inclusão, tanto para microestruturas regulares ou aleatórias. Estes sistemas podem ser do tipo que não possuam contato entre as partículas da fase dispersa (como no caso de concreto, ou núcleos celulares), ou sistemas do tipo empacotado, onde é abundante o número de contatos entre partículas, podendo formar até mesmo uma estrutura mecanicamente estável, existindo inúmeros métodos para a geração de tais sistemas, muitos dos quais são gerados a partir do conceito de cadeia de Markov (processo estocástico onde o próximo estado é gerado a partir de informações presentes somente no estado atual, sem considerar informações de estados passados)[DIETRICH, 2002].

Para tais métodos, o objetivo consiste em representar o mais fielmente possível informações experimentais estatísticas de tais materiais, sendo que, neste caso, a fração volumétrica possui uma das maiores influências, haja visto a condição de não sobreposição de partículas. Para sistemas aleatórios, essa condição se resume por testes de intersecção entre as partículas ou por cálculos de distância e, no caso específico de inclusões esféricas, tudo se resume ao cálculo entre centros das inclusões. Para este último tipo de arranjo, portanto, o máximo valor possível para a fração volumétrica reportado é de aproximadamente 64%, quando se considera uma dispersão de inclusões de mesmo tamanho, enquanto que para arranjos regulares este valor depende do tipo de arranjo escolhido [Jaeger and Nagel, 1992]. Para inclusões do tipo cilíndrico, elipsóides, cápsulas ou poliédricos, outros tipos de determinação para a condição de sobreposição existem, até mesmo com expressões analíticas [Bargmann et al., 2018].

Para os objetivos deste trabalho, se utiliza o tipo de arranjo chamado de empacotamento regular de microestruturas e tem justificativa no fato de que, para a análise de alguns tipos de comportamento, como deformação plástica ou resistência a dano, volumes elementares de maiores dimensões são necessários para garantir representatividade. Assim, a criação de microestruturas aleatórias pode se tornar até mesmo um problema para volumes com altas frações volumétricas.

No caso deste tipo de modelagem, a maior dificuldade se torna a automatização da geração de VRE's, de tal forma que se possa analisar diversas configurações espaciais dos arranjos bem como a realização de diferentes estudos sobre as propriedades estruturais destes materiais.

Todos estes pontos se resumem, portanto, na geração de uma ferramenta computacional robusta, que permita a análise térmica e estrutural de diferentes arranjos regulares de materiais compósitos bifásicos, com inclusões esféricas.

2.3 Métodos dos elementos finitos aplicados na identificação de propriedades efetivas

A descrição e formulação da metodologia dos elementos finitos está fora do escopo de desenvolvimento do presente trabalho, sendo que existem inúmeros trabalhos na literatura que realizam a apresentação de forma exímia e didática além de apresentarem diversos casos de aplicação de tais técnicas.

A aplicação do método dos elementos finitos para a análise deste tipo de comportamento mecânico de VRE's pode ser dividido em três passos: (i) a aplicação das condições de contorno periódicas e a definição da malha de elementos finitos; (ii) as relações constitutivas entre os componentes, e (iii) a determinação das propriedades elásticas efetivas do modelo.

A aplicação das condições de contorno devem, como já mencionado, ser aplicadas de forma a garantir uma distribuição de esforços internos representativa do comportamento do material. Para tal, duas condições devem ser satisfeitas nas regiões de superfície dos modelo. A primeira é que os deslocamentos dos nós constituintes destas regiões devem ser contínuos, devido ao fato de que VRE's vizinhos não podem ser separados ou se sobrepor, uns sobre os outros; enquanto que a segunda diz respeito a que a distribuição de tensões em superfícies paralelas opostas devem ser uniformes [Xu and Xu, 2008]. Estas condições podem ser traduzidas pela seguinte formulação, assim como apresentado em [Xia et al., 2003]:

$$u_i(x_1, x_2, x_3) = \bar{\varepsilon}_{ik} x_k + u_i^*(x_1, x_2, x_3) \quad (2.1)$$

onde $\bar{\varepsilon}_{ik}$ é o tensor de deformação em escala macro da estrutura periódica, x_k é um fator dado pela posição geométrica do nó e o primeiro termo a direita representa uma distribuição linear de deformação. O segundo termo ($u_i^*(x_1, x_2, x_3)$) é a função periódica que representa a modificação do campo de deslocamentos linear causado pela estrutura heterogênea do compósito. Este último termo geralmente é desconhecido e, por este motivo, não podem ser aplicadas diretamente. Quando considera-se um modelo tridimensional com superfícies paralelas, entretanto, os deslocamentos de pares paralelos de superfícies podem ser reescritos como:

$$u_i^{j+} = \bar{\varepsilon}_{ik} x_k^{j+} + u_i^*; \quad u_i^{j-} = \bar{\varepsilon}_{ik} x_k^{j-} + u_i^* \quad (2.2)$$

e, dada a condição de periodicidade, denotada pelos índices que identificam os pares, estas equações podem ser reescritas como:

$$u_i^{j+} - u_i^{j-} = \bar{\varepsilon}_{ik} (x_k^{j+} - x_k^{j-}) = \bar{\varepsilon}_{ik} \Delta x_k^j \quad (2.3)$$

sendo os valores de Δx_k^j definidos para cada par de superfícies de contorno, como uma deformação determinada, os valores a direita se tornam constantes, e a sua aplicação ao modelo VRE é simplificada como uma aplicação de restrição nodal de deslocamento.

Esta aplicação é, então, distribuída entre os vários elementos componentes do modelo e segue uma formulação tensorial tal qual a apresentada por Danielsson [Danielsson et al., 2002] que relaciona o deslocamento dos nós opostos correspondentes do modelo com a deformação total do mesmo. Este equacionamento tensorial pode ser apresentado como:

$$\begin{Bmatrix} u_{ij} \\ v_{ij} \\ w_{ij} \end{Bmatrix} = [W_{ij}] \begin{bmatrix} \epsilon_x & \epsilon_{xy} & \epsilon_{xz} \\ \epsilon_{xy} & \epsilon_y & \epsilon_{yz} \\ \epsilon_{xz} & \epsilon_{yz} & \epsilon_z \end{bmatrix} \quad (2.4)$$

onde: u , v e w são os deslocamentos dos nós i e j nas direções x , y e z , respectivamente; A matriz W contém os pesos correspondentes a cada deslocamento e $\epsilon_{..}$ são as componentes de deformação do modelo. O nó i e o nó j estão localizados em regiões opostas sobre o modelo a partir dos planos xy , xz ou yz . A partir desta equação tensorial é possível realizar um desenvolvimento algébrico e se encontrar equações explícitas envolvendo os nós dos vértices, arestas e faces do modelo. Para o desenvolvimento deste conjunto de equações é utilizado um modelo referência com uma nomenclatura a ser utilizada durante toda a execução do algoritmo. Este modelo é o apresentado na Figura 6.

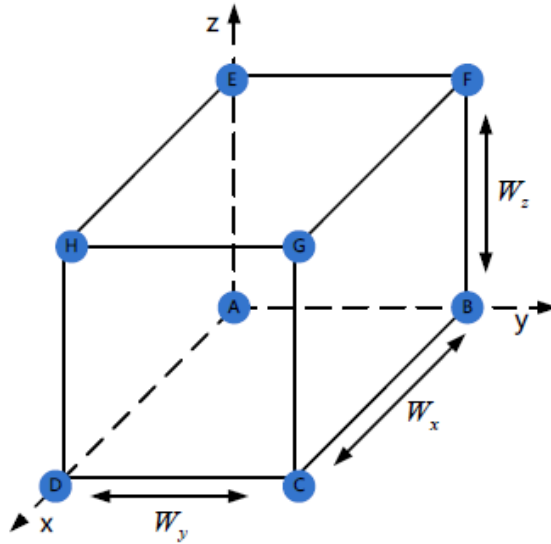


Figura 6 – Modelo com a nomenclatura utilizada para o desenvolvimento das equações de periodicidade. Fonte: [Fan and Wang, 2017]

Resolvida a questão de consistência da nomenclatura encontra-se o conjunto de equações tais quais apresentado em [Fan and Wang, 2017] e descritos a seguir.

Para as arestas:

$$\begin{cases} u_{CG} - u_{AE} = W_x \varepsilon_x + W_y \varepsilon_{xy} \\ v_{CG} - v_{AE} = W_x \varepsilon_{yx} + W_y \varepsilon_y \\ w_{CG} - w_{AE} = W_x \varepsilon_{zx} + W_y \varepsilon_{zy} \end{cases} \quad \begin{cases} u_{HD} - u_{FB} = W_x \varepsilon_x + W_y \varepsilon_{xy} \\ v_{HD} - v_{FB} = W_x \varepsilon_{yx} + W_y \varepsilon_y \\ w_{HD} - w_{FB} = W_x \varepsilon_{zx} + W_y \varepsilon_{zy} \end{cases}$$

$$\begin{cases} u_{HG} - u_{AB} = W_x \varepsilon_x + W_z \varepsilon_{xz} \\ v_{HG} - v_{AB} = W_x \varepsilon_{yx} + W_z \varepsilon_{yz} \\ w_{HG} - w_{AB} = W_x \varepsilon_{zx} + W_z \varepsilon_z \end{cases} \quad \begin{cases} u_{DC} - u_{EF} = W_x \varepsilon_x + W_z \varepsilon_{xz} \\ v_{DC} - v_{EF} = W_x \varepsilon_{yx} + W_z \varepsilon_{yz} \\ w_{DC} - w_{EF} = W_x \varepsilon_{zx} + W_z \varepsilon_z \end{cases}$$

$$\begin{cases} u_{FG} - u_{AD} = W_y \varepsilon_{xy} + W_z \varepsilon_{xz} \\ v_{FG} - v_{AD} = W_y \varepsilon_y + W_z \varepsilon_{yz} \\ w_{FG} - w_{AD} = W_y \varepsilon_{zy} + W_z \varepsilon_z \end{cases} \quad \begin{cases} u_{BC} - u_{EH} = W_y \varepsilon_{xy} + W_z \varepsilon_{xz} \\ v_{BC} - v_{EH} = W_y \varepsilon_y + W_z \varepsilon_{yz} \\ w_{BC} - w_{EH} = W_y \varepsilon_{zy} + W_z \varepsilon_z \end{cases}$$

Para os vértices:

$$\begin{cases} u_G - u_A = W_x \varepsilon_x + W_y \varepsilon_{xy} + W_z \varepsilon_{xz} \\ v_G - v_A = W_x \varepsilon_{yx} + W_y \varepsilon_y + W_z \varepsilon_{yz} \\ w_G - w_A = W_x \varepsilon_{zx} + W_y \varepsilon_{zy} + W_z \varepsilon_z \end{cases} \quad \begin{cases} u_F - u_D = -W_x \varepsilon_x + W_y \varepsilon_{xy} + W_z \varepsilon_{xz} \\ v_F - v_D = -W_x \varepsilon_{yx} + W_y \varepsilon_y + W_z \varepsilon_{yz} \\ w_F - w_D = -W_x \varepsilon_{zx} + W_y \varepsilon_{zy} + W_z \varepsilon_z \end{cases}$$

$$\begin{cases} u_H - u_B = W_x \varepsilon_x - W_y \varepsilon_{xy} + W_z \varepsilon_{xz} \\ v_H - v_B = W_x \varepsilon_{yx} - W_y \varepsilon_y + W_z \varepsilon_{yz} \\ w_H - w_B = W_x \varepsilon_{zx} - W_y \varepsilon_{zy} + W_z \varepsilon_z \end{cases} \quad \begin{cases} u_C - u_E = W_x \varepsilon_x + W_y \varepsilon_{xy} - W_z \varepsilon_{xz} \\ v_C - v_E = W_x \varepsilon_{yx} + W_y \varepsilon_y - W_z \varepsilon_{yz} \\ w_C - w_E = W_x \varepsilon_{zx} + W_y \varepsilon_{zy} - W_z \varepsilon_z \end{cases}$$

Para as faces, foi encontrado um problema de consistência entre a formulação apresentada na literatura com os resultados obtidos. A partir da análise cuidadosa dos equacionamentos das referências para as condições de contorno periódicas, notou-se que as condições das faces não apresentavam relação como os fatores de cisalhamento globais do material (ε_{xy} , ε_{xz} e ε_{yz}). Esta carência provoca uma concentração de tensões nas arestas dos modelos MEF, alterando completamente a interpretação dos resultados obtidos. A seguir, apresenta-se o equacionamento apropriado para as faces, adaptado daquele da literatura:

$$\begin{cases} u_{HGCD} - u_{EFBA} = W_x \varepsilon_x \\ v_{HGCD} - v_{EFBA} = W_y \varepsilon_{xy} \\ w_{HGCD} - w_{EFBA} = W_z \varepsilon_{xz} \end{cases} \quad \begin{cases} u_{BCGF} - u_{ADHE} = W_x \varepsilon_{xy} \\ v_{BCGF} - v_{ADHE} = W_y \varepsilon_y \\ w_{BCGF} - w_{ADHE} = W_z \varepsilon_{yz} \end{cases} \quad \begin{cases} u_{EFGH} - u_{ABCD} = W_x \varepsilon_{xz} \\ v_{EFGH} - v_{ABCD} = W_y \varepsilon_{yz} \\ w_{EFGH} - w_{ABCD} = W_z \varepsilon_z \end{cases}$$

Nota-se que para a aplicação de tais condições, representadas pelas equações acima, deve existir compatibilidade posicional entre os nós da malha de elementos finitos. Esta compatibilidade pode ser obtida pela simetria da malha, obtida ao assumir que a ligação entre as inclusões e a matriz é perfeita e a geração da malha é feita a partir de elementos do tipo tetraédricos tridimensionais. Caso não exista simetria, pode-se proceder ao pareamento dos nós a partir de seus posicionamentos geométricos, realizando-se uma combinação linear entre os nós que se aproximam da posição ideal de pareamento.

As relações constitutivas são um resultado dos métodos de homogeneização utilizados para a formulação do problema. Neste caso, o VRE é modelado como um material homogêneo ortotrópico com propriedades efetivas correspondentes às propriedades médias estatísticas de tal

compósito. A descrição destas propriedades macroscópicas, os esforços e deformações macroscópicos são normalizados pelo volume do VRE, e a sua formulação matemática pode ser visualizada a seguir, tal como é descrita por [Bensoussan et al., 2011]:

$$\bar{\sigma}^{ij} = \frac{1}{V_{VRE}} \int_V \sigma_{ij}(x, y, z) dV \quad (2.5)$$

$$\bar{\varepsilon}^{ij} = \frac{1}{V_{VRE}} \int_V \varepsilon_{ij}(x, y, z) dV \quad (2.6)$$

Então, a energia de deformação contida no VRE heterogêneo pode ser descrita por:

$$U^* = \frac{1}{2} \int_{V_{VRE}} \sigma^{ij} \varepsilon^{ij} dV \quad (2.7)$$

Para um VRE de material homogêneo, esta relação pode ser simplificada para:

$$U = \frac{1}{2} \sigma^{ij} \varepsilon^{ij} V^{VRE} \quad (2.8)$$

Como mostrado em [teh Sun and Vaidya, 1996], as equações 2.7 e 2.8 podem ser combinadas utilizando o teorema de Gauss para uma integral sobre a superfície do modelo:

$$U^* - U = \frac{1}{2} \int_{S_j} \sigma^{ij} (u_i - \bar{u}_i) n_j dS_j \quad (2.9)$$

onde S_j é a j ésima superfície e n é a normal a esta superfície, u_i é o deslocamento e $\bar{u}_i$ é o deslocamento médio. Como na superfície tem-se $u_i = \bar{u}_i$, a diferença de energia resultante entre os modelos heterogêneos e homogêneos é:

$$U^* - U = 0 \quad (2.10)$$

garantindo a equivalência entre as energias de deformação entre os materiais homogêneos e heterogêneos. Em [teh Sun and Vaidya, 1996], é apresentado que se pode seguir a mesma sequência para as deformações médias do VRE, obtendo-se:

$$\bar{\varepsilon}^{ij} = \frac{1}{V_{VRE}} \int_{S_1} (u_i n_j + u_j n_i) n_j dS_1 \quad (2.11)$$

onde S_1 é a superfície externa do VRE. Esta formulação possibilita o cálculo das deformações médias do modelo a partir dos deslocamentos das superfícies externas, sem a necessidade de integração no volume do modelo.

Por fim, o tensor de rigidez do modelo pode ser obtido pela razão entre os esforços médios sofridos pelo VRE pela sua deformação, resultando na equação tensorial apresentada em 2.12:

$$\begin{Bmatrix} \bar{\sigma}_{11} \\ \bar{\sigma}_{22} \\ \bar{\sigma}_{33} \\ \bar{\sigma}_{12} \\ \bar{\sigma}_{13} \\ \bar{\sigma}_{23} \end{Bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & 0 & 0 & 0 \\ C_{12} & C_{22} & C_{23} & 0 & 0 & 0 \\ C_{13} & C_{23} & C_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & C_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & C_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & C_{66} \end{bmatrix} \begin{Bmatrix} \bar{\varepsilon}_{11} \\ \bar{\varepsilon}_{22} \\ \bar{\varepsilon}_{33} \\ \bar{\varepsilon}_{12} \\ \bar{\varepsilon}_{13} \\ \bar{\varepsilon}_{23} \end{Bmatrix} \quad (2.12)$$

Para obter as propriedades efetivas do modelo VRE basta solucionar o sistema apresentado em 2.12 para a matriz de rigidez e, a partir desta, encontrar a matriz de flexibilidade do sistema (inversa da matriz de rigidez). A partir da matriz de flexibilidade pode-se calcular as propriedades efetivas a partir do seguinte conjunto de equações [Xu and Xu, 2008]:

$$\begin{aligned} E_x &= \frac{1}{S_{11}} & E_y &= \frac{1}{S_{22}} & E_z &= \frac{1}{S_{33}} \\ \nu_{xy} &= -\frac{S_{12}}{S_{22}} & \nu_{xz} &= -\frac{S_{13}}{S_{33}} & \nu_{yz} &= -\frac{S_{23}}{S_{33}} \\ G_{xy} &= \frac{1}{S_{44}} & G_{xz} &= \frac{1}{S_{55}} & G_{yz} &= \frac{1}{S_{66}} \end{aligned} \tag{2.13}$$

CAPÍTULO 3

METODOLOGIA

A metodologia deste trabalho pode ser dividida em duas etapas principais: a geração das microestruturas artificiais tridimensionais e a geração das ferramentas automatizadas. Estas ferramentas podem, então, ser utilizadas para a aplicação em outras fases do presente trabalho para a determinação das propriedades efetivas dos materiais a partir de sua microestrutura. Na Figura 7 é apresentada uma visão geral de como o trabalho foi desenvolvido.

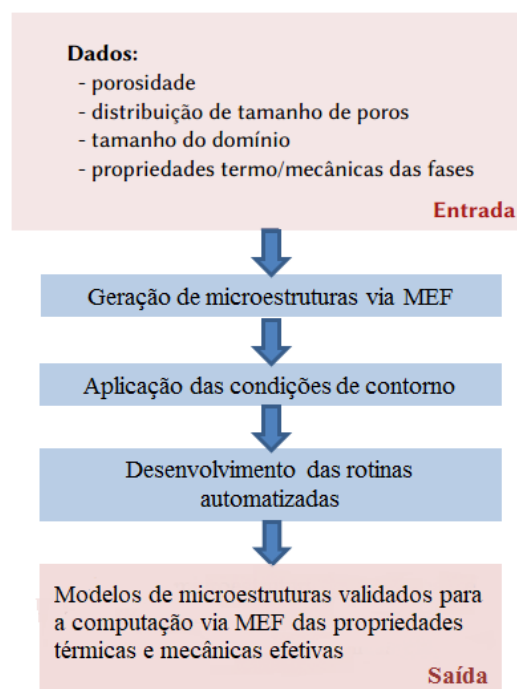


Figura 7 – Metodologia adotada para o desenvolvimento do trabalho de conclusão de curso.

Na primeira etapa foram gerados modelos de microestrutura representativos a partir de parâmetros macroscópicos dos materiais, como a porosidade, tamanho e disposição dos grãos, características geométricas e proporção de fases constituintes. As microestruturas foram geradas utilizando conceitos de células de Kelvin e Wearie-Phelan.

Na segunda etapa, foram desenvolvidas rotinas computacionais automatizadas para a geração artificial e análise das microestruturas regulares e periódicas. As rotinas são capazes de gerar o modelo geométrico, aplicar a malha de elementos finitos com refinamento desejado, aplicar as condições de contorno e periodicidade ao modelo e realizar a resolução do problema via MEF para posterior comparação com resultados experimentais.

3.1 Geração dos modelos de volumes representativos elementares

Inicialmente foi-se necessário a geração dos modelos de microestruturas analisados por este trabalho. Foram gerados modelos bi e tridimensionais, com o auxílio do software de modelagem e análise MEF Abaqus®. Estes modelos são baseados em materiais compósitos de matriz vítrea e inclusões de alumina com arranjo estrutural regular. Foi-se utilizado do trabalho publicado [Tessier-Doyen et al., 2006] para obtenção das informações necessárias à construção dos modelos, tais como: dimensões das microestruturas, frações volumétricas, fases constituintes, propriedades mecânicas (Módulo de Young - E -, coeficiente de Poisson - ν -) e térmicas (coeficiente de expansão - α -). Estas informações são de extrema importância para a construção de um modelo computacional representativo da realidade.

Em [Tessier-Doyen et al., 2006], são apresentadas as dependências entre as propriedades mecânicas e as térmicas para alguns modelos de microestruturais. Estes modelos apresentados são constituídos de duas fases: a matriz vítrea e as inclusões (agregado). As propriedades das matrizes são correspondentes às apresentadas por 3 diferentes composições de vidro de boro-silicato sendo chamadas de G1, G2 e G3. Já as propriedades das inclusões são correspondentes às propriedades de esferas de 99.9% pureza de alumina (Al_2O_3). Para as proporções de fases constituintes, foram utilizados as proporções de 15%, 30% e 45%. As dimensões dos modelos foram calculadas levando-se em conta inclusões esféricas de diâmetro de 500 μm . As características mecânicas e térmicas aplicadas aos modelos podem ser visualizadas na Figura 8 e foram retiradas de [Tessier-Doyen et al., 2006].

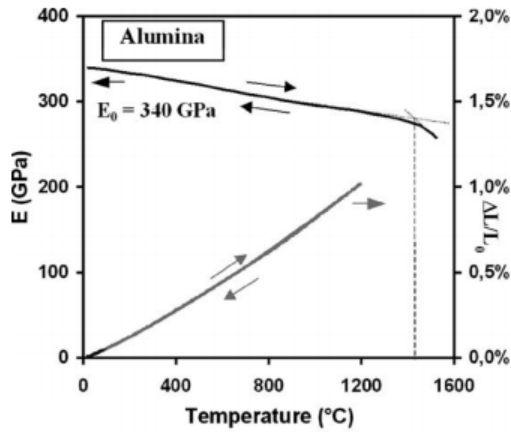
Com o auxílio de uma rotina de tira-pontos em Matlab®, foi possível retirar da Figura 8 as informações necessárias à construção do modelo. Estas informações são basicamente a relação entre o módulo de elasticidade dos materiais (Módulo de Young) com a temperatura bem como a relação entre a expansão térmica dos mesmos com a temperatura. De posse destes dados prosseguiu-se à construção dos modelos.

Para o caso analisado e apresentado a seguir foram utilizados as seguintes informações sobre o comportamento mecânico e térmico dos materiais:

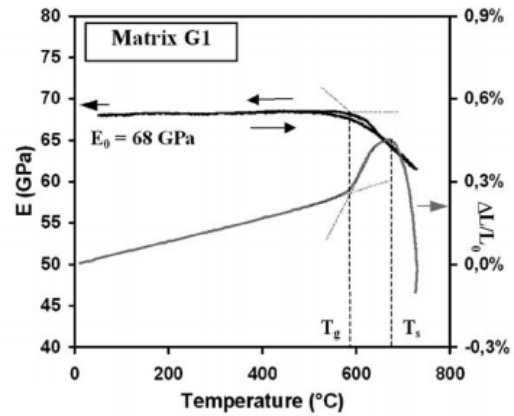
- Alumina: Decaimento linear do Módulo de Young (E) de 340 GPa a 300GPa, para uma temperatura (T) de 25°C a 605°C; coeficiente de Poisson (ν) de 0.24 e coeficiente de expansão térmica (α) de $7.6e^{-6}K^{-1}$;
- Matriz G1: Módulo de Young (E) de 68GPa aproximadamente constante, para uma temperatura (T) de 25°C a 595°C; coeficiente de Poisson (ν) de 0.2 e coeficiente de expansão térmica (α) de $4.6e^{-6}K^{-1}$;
- Matriz G2: Módulo de Young (E) de 76GPa aproximadamente constante, para uma temperatura (T) de 25°C a 605°C; coeficiente de Poisson (ν) de 0.21 e coeficiente de

expansão térmica (α) de $7.4e^{-6}K^{-1}$;

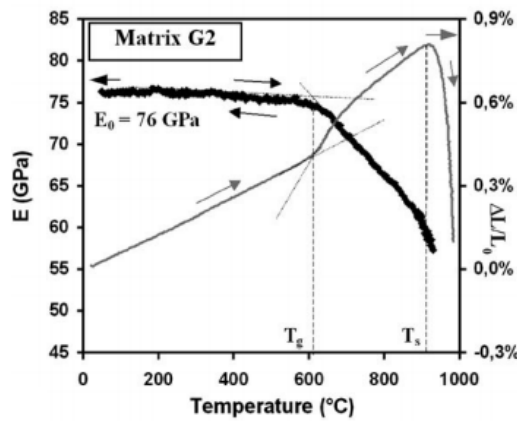
- Matriz G3: Decaimento aproximadamente linear do Módulo de Young (E) de 72GPa para 62GPa, para uma temperatura (T) de 25°C a 470°C; coeficiente de Poisson (ν) de 0.23 e coeficiente de expansão térmica (α) de $11.6e^{-6}K^{-1}$;



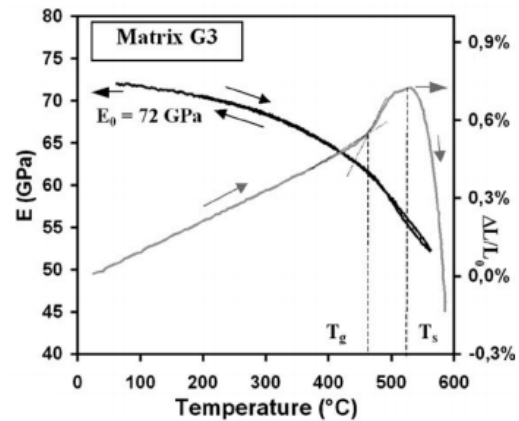
(a) Propriedades - Alumina



(b) Propriedades - Matriz G1



(c) Propriedades - Matriz G2



(d) Propriedades - Matriz G3

Figura 8 – Evolução do Módulo de Young e expansão térmica dos materiais constituintes. Fonte: [Tessier-Doyen et al., 2006]

Para a aplicação das condições de periodicidade nos nós da malha de elementos finitos gerada há a necessidade de compatibilidade da malha localizada em regiões opostas do modelo, por exemplo nós correspondentes de duas faces opostas. Para gerar esta compatibilidade de malha, inicialmente foi gerada uma geometria correspondente a $\frac{1}{8}$ do volume representativo desejado. Em seguida utilizou-se de espelhamento desta geometria nos três planos espaciais de referência. Por fim, as oito geometrias geradas são concatenadas em um único modelo, que possui a compatibilidade de malha necessária entre as superfícies.

Os modelos construídos apresentam arranjos geométricos regulares, baseadas nos arranjos atômicos cúbico (C), cúbico de corpo centrado (CCC) e cúbico de face centrada (CFC). Na Figura 9 são apresentadas as configurações geradas.

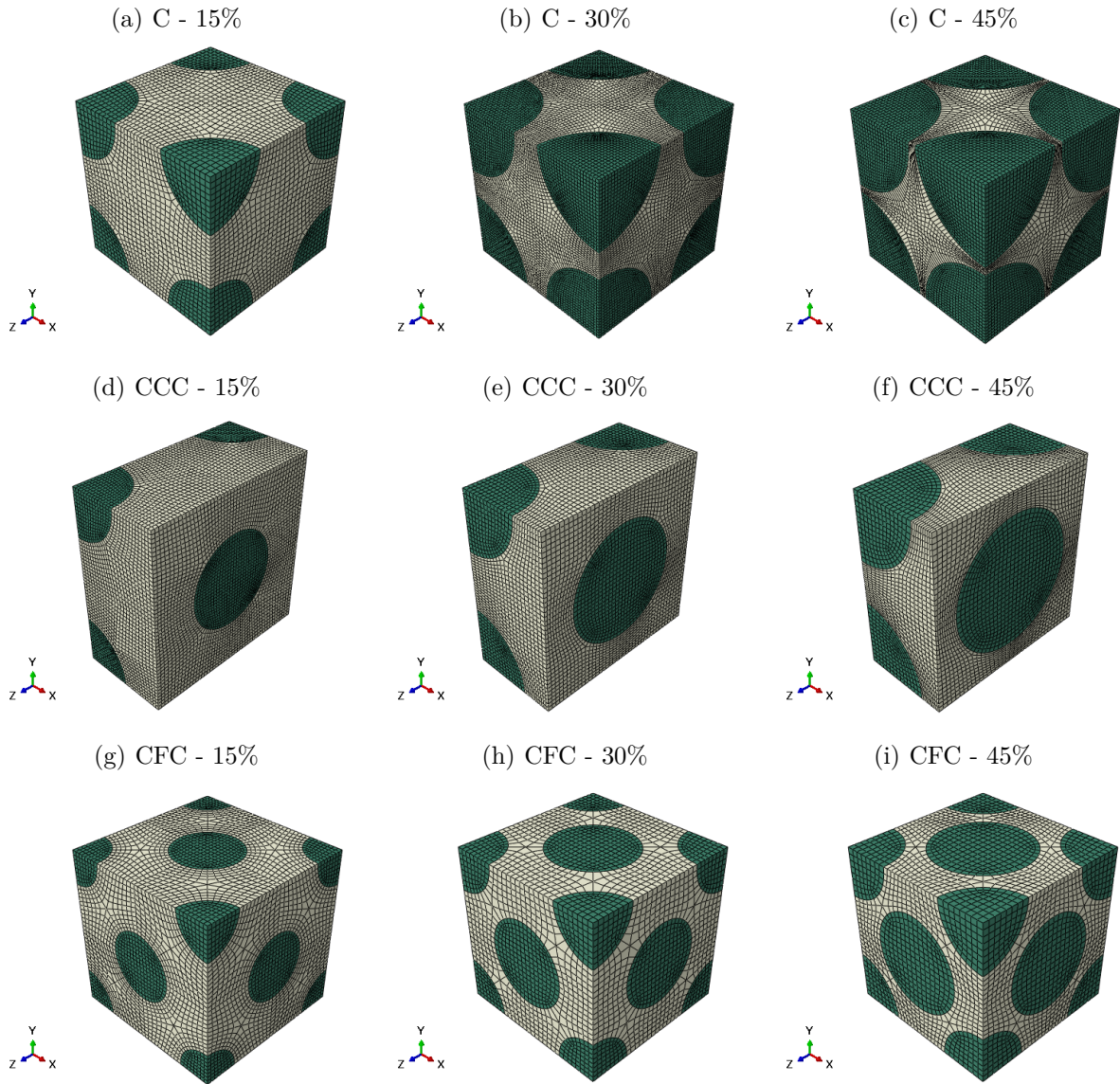


Figura 9 – Modelos computacionais gerados de acordo com o arranjo microestrutural e a fração volumétrica. Inclusões de Alumina e matriz G2.

De posse das equações apresentadas para os vértices, arestas e faces, pôde-se realizar a aplicação das condições de periodicidade aos modelos gerados. Esta aplicação pode ser realizada a partir de duas estratégias, sendo que a primeira consiste em uma simplificação destas condições para o caso em que o modelo não seja submetido a esforços cisalhantes e a segunda se baseia na utilização de *dummy nodes*.

Para a primeira estratégia, são anulados todos os termos que sejam composição de deformação em eixos diferentes, sendo utilizados somente aqueles que sejam esforços puros de deformação uniaxial, podendo existir a combinação destes esforços. A segunda estratégia utiliza os *dummy nodes*, que nada mais são que pontos de referência gerados no modelo, cujos graus de liberdades correspondam às deformações totais do modelo. Assim, devem ser utilizados dois pontos de referência: o primeiro governa as deformações normais ϵ_x , ϵ_y e ϵ_z e o segundo ponto de

referência governa as deformações cisalhantes ϵ_{xy} , ϵ_{xz} e ϵ_{yz} .

Para o estudo dos efeitos de temperatura que são de interesse, a primeira abordagem pode ser utilizada sem grandes restrições, haja visto que os campos de temperatura aplicados ao modelo são uniformes em todas as direções e causam expansão ou encolhimento volumétricos. Como as inclusões são de formato esférico, as interações entre a matriz e as inclusões se dá de forma isotrópica e, apesar de existirem esforços cisalhantes na interface, estas interações não se relacionam às condições de periodicidade aplicadas por meio da malha e sim, surge como resultado da própria formulação em MEF. A abordagem utilizando *dummy nodes* é mais geral e comporta a aplicação de esforços cisalhantes. Como, para a computação das propriedades efetivas dos modelos, é necessária a aplicação destes esforços de deformação e cisalhamento, esta foi a estratégia escolhida e seguida para a aplicação das condições de periodicidade ao modelo, mesmo que tenha apresentado grande dificuldade de implementação, especialmente pela formulação matemática incompleta apresentada pela literatura.

Com as condições de periodicidade aplicadas ao modelo, ainda se faz necessário a discussão e aplicação das condições de contorno ao problema analisado. Como mencionado na introdução deste trabalho, o objetivo final é a investigação dos fenômenos de concentração de tensão que ocorrem na interface entre as inclusões e a matriz ao nível microestrutural quando do aquecimento ou resfriamento do material, devido a diferenças entre os coeficientes de expansão térmica dos materiais da matriz e das inclusões. Desta forma, as condições de contorno espaciais a serem aplicadas ao modelo é o impedimento dos deslocamentos do nó central do modelo. Os graus de liberdade de giro em torno dos eixos cartesianos devem ser ignorados para que não ocorra o fenômeno de concentração de tensão pontual, que ocasionaria a distorção do campo de tensões e deformações locais, comprometendo os resultados numéricos encontrados naquela região. Assim:

$$\begin{Bmatrix} u \\ v \\ w \end{Bmatrix}_{centernode} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix} \quad (3.1)$$

Por fim, prossegue-se à análise em MEF dos modelos para a posterior extração das propriedades efetivas. Inicialmente, aplica-se o campo de temperaturas de análise desejado e, em seguida aplica-se, isoladamente, um dos seis casos de carga puros ao modelo, com a utilização dos *dummy nodes*, que distribuem a carga para o restante dos nós do modelo. Os casos de carga aplicados são representados simbolicamente na Equação 3.2:

$$\begin{Bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ \epsilon_{xy} \\ \epsilon_{xz} \\ \epsilon_{yz} \end{Bmatrix} = \begin{Bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix}; \begin{Bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix}; \begin{Bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{Bmatrix}; \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{Bmatrix}; \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{Bmatrix}; \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{Bmatrix} \quad (3.2)$$

3.2 Organização das rotinas de geração dos modelos de microestruturas

Com as sequências de geração de todos os modelos já implementadas seguiu-se para geração do algoritmo responsável por realizar todas estas tarefas de maneira automatizada. Este algoritmo consegue controlar o arranjo desejado para a modelagem, as dimensões (tanto das inclusões quanto dos modelos), a composição matriz-inclusão a ser utilizada e a aplicação das propriedades mecânicas e térmicas correspondentes, o refino da malha de elementos finitos.

Este algoritmo é escrito em *Python*, utilizando as bibliotecas do *software* Abaqus®, sendo apresentado, na íntegra, no Apêndice. O propósito deste algoritmo é reproduzir os procedimentos descritos acima para a geração automática dos modelos de interesse. O primeiro passo é a geração das geometrias; segue-se para a aplicação das propriedades físicas e mecânicas de cada uma das fases constituintes; aplica-se, então, as condições de periodicidade aos nós da malha de elementos finitos; e, por fim, realiza-se a simulação numérica do modelo e a computação das propriedades efetivas do modelo a partir da técnica de homogeneização apresentada. Na Figura 10 é apresentado um fluxograma ilustrativo de como esta rotina está organizada e os passos seguidos em sequência para a geração e simulação dos modelos.

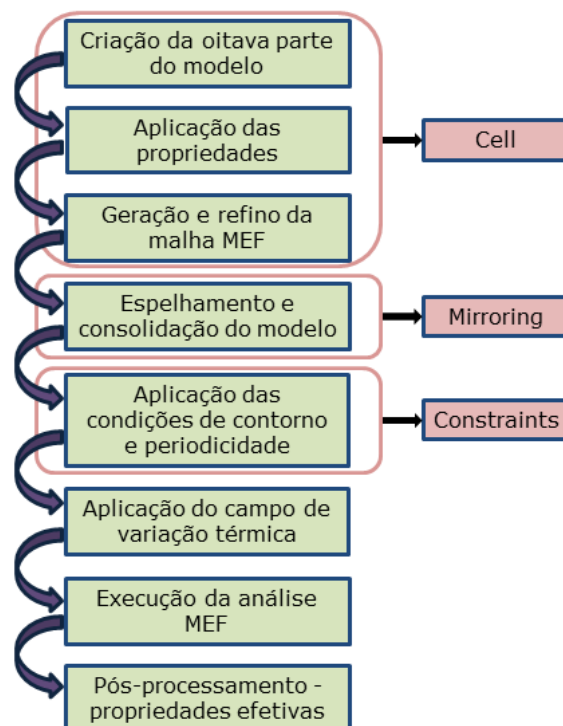


Figura 10 – Fluxograma do algoritmo de automatização da geração dos modelos

Com estes algoritmos finalizados, outra possibilidade implementada foi a criação de *plugins* internos ao ambiente Abaqus®, de forma a se eliminar a necessidade de execução manual do algoritmo, reduzindo o trabalho do operador. Optou-se pela execução separada dos algoritmos (Cell, Mirroring, Constraints, Analysis e Report) para que o usuário possua melhor controle e acompanhamento mais claro de quais etapas estão sendo executadas e caso o mesmo queira efetuar quaisquer mudanças no algoritmo. Na Figura 11 é apresentado um fluxograma de execução

dos algoritmos implementados.

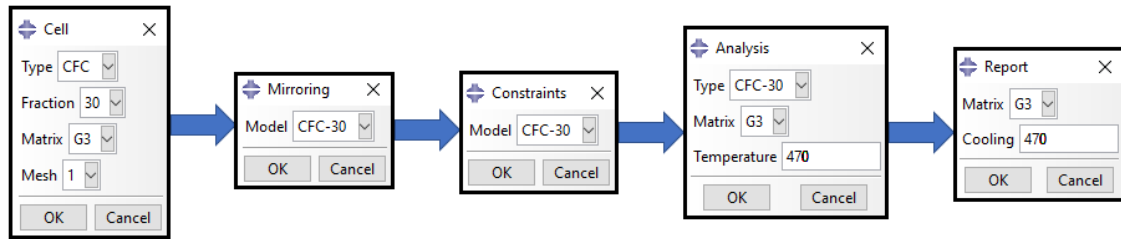


Figura 11 – Fluxograma de execução dos algoritmos em forma de *plug-ins* internos ao ambiente Abaqus®

A análise do comportamento mecânico dos modelos sujeitos a um gradiente de temperatura é feito considerando a fase de resfriamento do sistema. Assim como apresentado em [Tessier-Doyen et al., 2006], o comportamento durante a fase de aquecimento é extremamente não linear, devido às interações entre a matriz e as inclusões, como o alívio das tensões internas próximo às temperaturas de transição vítrea da matriz e a consequente acomodação da massa matricial devido ao seu amolecimento. Caso a matriz possua coeficiente de expansão térmica inferior ao das inclusões, o material, ao ser resfriado, apresenta o fenômeno de descolamento na região de interface matriz/inclusão. No caso contrário, se a matriz apresentar coeficiente de expansão térmica superior ao das inclusões, o fenômeno característico é o surgimento de micro-trincas de direção radial em relação à superfície de interface. Caso os materiais apresentem coeficientes semelhantes, não são observados fenômenos de interferência significativa. Durante a fase de resfriamento, ao atingir determinada temperatura, a plasticidade da matriz não é mais capaz de acomodar os deslocamentos relativos na região da interface, tanto para casos de descolamento quanto para os de trincas radiais. A partir deste ponto, os efeitos induzidos pelas tensões residuais térmicas levam ao decaimento do módulo de elasticidade do material, não sendo possível identificar precisamente qual dos defeitos é o responsável por este decaimento, descolamento ou as trincas radiais. Dadas estas condições, é possível realizar a comparação entre os resultados experimentais e os resultados obtidos pelos modelos em MEF durante a fase de resfriamento entre as temperaturas de transição vítrea e a temperatura na qual o decaimento do módulo de elasticidade é acentuado. Os modelos MEF não levam em conta quaisquer efeitos de decaimento do módulo de elasticidade pela existência de trincas, devendo se fazer esta ressalva quando da comparação dos resultados.

Aplicado o campo de variação de temperaturas, utiliza-se os resultados gerados para a análise da influência do fenômeno de concentração de tensão nas propriedades efetivas do material. Para tal, aplica-se as deformações correspondentes aos graus de liberdade dos *dummy nodes*, de acordo com as condições apresentadas em 3.2, e computa-se as propriedades efetivas de acordo com as relações apresentadas em 2.13.

CAPÍTULO 4

RESULTADOS

Os resultados deste trabalho serão apresentados em duas seções. A primeira é orientada à apresentação dos modelos resultantes desta modelagem, com a exibição dos modelos gerados ao final de cada etapa de geração e a apresentação do comportamento mecânico apresentado pelos modelos dadas as condições de solicitação de tração, cisalhamento e variação de temperatura. A segunda parte é orientada à verificação das propriedades efetivas resultantes e a sua comparação com resultados experimentais e numéricos disponíveis na literatura.

4.1 Apresentação dos modelos resultantes

O algoritmo é iniciado com a geração da oitava parte do modelo desejado. Neste momento são aplicadas as propriedades dos materiais componentes, os parâmetros da malha de elementos finitos bem como são definidas as dimensões do modelo a ser avaliado. Tudo é realizado a partir da utilização do algoritmo instalado no *plug-in Cell*. O resultado da execução para um modelo CFC-30 pode ser observado na Figura 12.

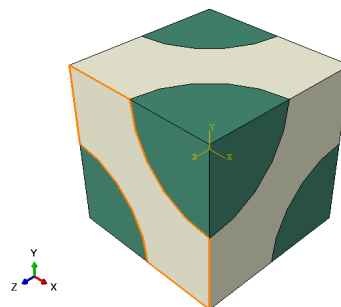


Figura 12 – Geração da oitava parte do modelo CFC-30

Em seguida é utilizado o algoritmo instalado em *plug-in Mirror*. Nesta etapa, a oitava parte do modelo é replicada e transladada no plano cartesiano, de forma a se originar o modelo final, garantindo-se que as malhas de elementos finitos aplicadas em todas as faces do modelo sejam simétricas, permitindo a posterior aplicação das condições de periodicidade. O resultado

obtido pela execução deste algoritmo é apresentada na Figura 13.

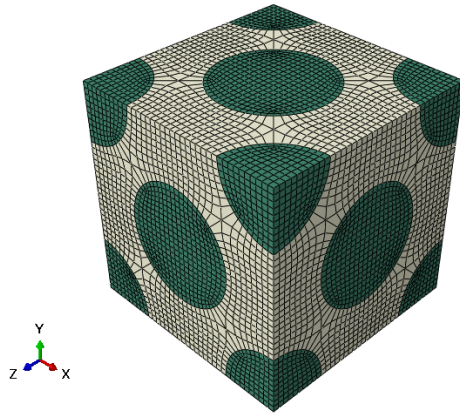


Figura 13 – Modelo CFC-30 resultante, já com a aplicação da malha simétrica de elementos finitos

O próximo passo é a aplicação das condições de periodicidade, tal qual descrito nas equações apresentadas anteriormente, a partir da utilização do algoritmo implementado no *plug-in Constraints*. Como já explicado, esta etapa realiza a vinculação entre os graus de liberdade dos nós, tomados dois a dois, diametralmente opostos junto aos graus de liberdade correspondentes dos *dummy nodes*. Lembra-se que o *dummy node* RP1 é utilizado para governar os esforços de tração aplicados ao modelo ($\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{zz}$) enquanto que o *dummy node* RP2 governa os esforços de cisalhamento ($\varepsilon_{xy}, \varepsilon_{xz}, \varepsilon_{yz}$). O resultado obtido por esta etapa pode ser visualizado na Figura 14.

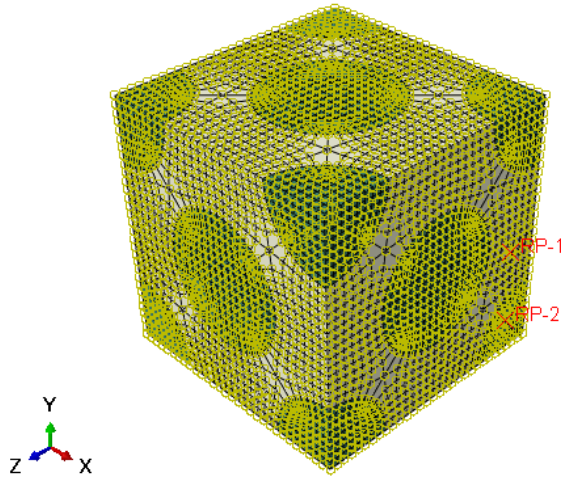


Figura 14 – Aplicação das condições de periodicidade ao modelo CFC-30, de matriz G3

A aplicação do campo de temperaturas é realizado em conjunto com a aplicação dos casos de carga puros durante a execução do algoritmo implementado no *plug-in Analysis*. A temperatura definida pelo usuário gera um campo de tensões residuais que é preservado durante a aplicação e solução para os casos dos esforços mecânicos aplicados. Durante o desenvolvimento dos algoritmos de automatização foram analisadas algumas estratégias para a aplicação da variação de temperaturas. A primeira delas consiste no aquecimento do modelo, de uma temperatura de

25°C à T_g seguida do resfriamento do modelo à temperatura desejada; a segunda consiste no aquecimento do modelo de 25°C à temperatura desejada; e a terceira consiste no resfriamento do modelo da temperatura T_g à temperatura desejada. Todas as estratégias utilizadas apresentaram os mesmos resultados. Tal fato vai de encontro com a formulação em elementos finitos, que realiza a sobreposição linear dos efeitos de temperatura e de esforços mecânicos. Na Figura 15 é apresentado este campo de tensões residuais para a simulação do modelo CFC-30 composto pela matriz G3, submetido a um campo de temperatura da ordem de 470°C.

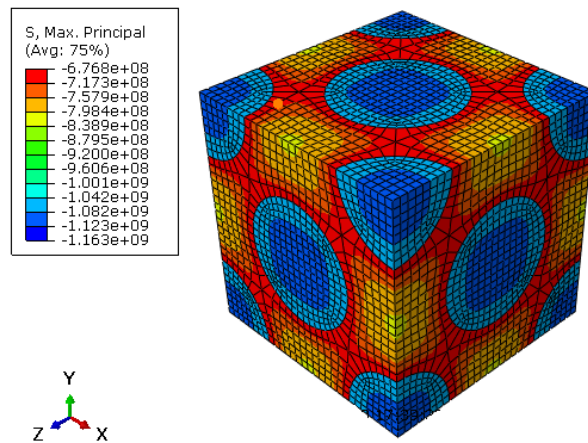


Figura 15 – Campo residual de tensões devido à variação de temperatura de 470°C a 25°C aplicada ao modelo CFC-30, matriz G3

Neste ponto, antes da aplicação dos casos de carga, cabe a discussão de qual o valor de deslocamento deve ser aplicado aos *dummy nodes* do modelo para que os resultados das propriedades efetivas obtidos sejam, de fato, representativos do comportamento dos modelos. A aplicação de valores muito pequenos de deslocamento ocasiona uma dominância dos efeitos de temperatura sobre os efeitos mecânicos e, conseqüentemente, os valores obtidos para as propriedades mecânicas serão distorcidos por este efeito. Em geral, caso os materiais possuísem coeficientes de deformação térmica idênticos, os valores das propriedades efetivas encontrados seriam inferiores aos esperados, haja visto que a deformação térmica não ocasionaria qualquer concentração de tensão correspondente. Já a aplicação de valores de deslocamento muito grandes ocasiona o efeito oposto, o de dominância dos efeitos mecânicos sobre os de temperatura, gerando resultados não significativos para o caso de estudo deste trabalho. Uma série de testes foi realizada com o intuito de se descobrir quais os valores ideais de deslocamento que devem ser aplicados ao modelo de forma a se obter valores de deformação e tensão com, no máximo, uma ordem de grandeza de diferença. Os valores encontrados encontram-se na ordem de $\varepsilon = 10^{-4}$.

Na Figura 16 são apresentados os resultados obtidos para a aplicação dos seis casos correspondentes dos esforços mecânicos, apresentados na Equação 3.2, sobre o modelo CFC-30, de matriz G3, a uma temperatura de 470°C.

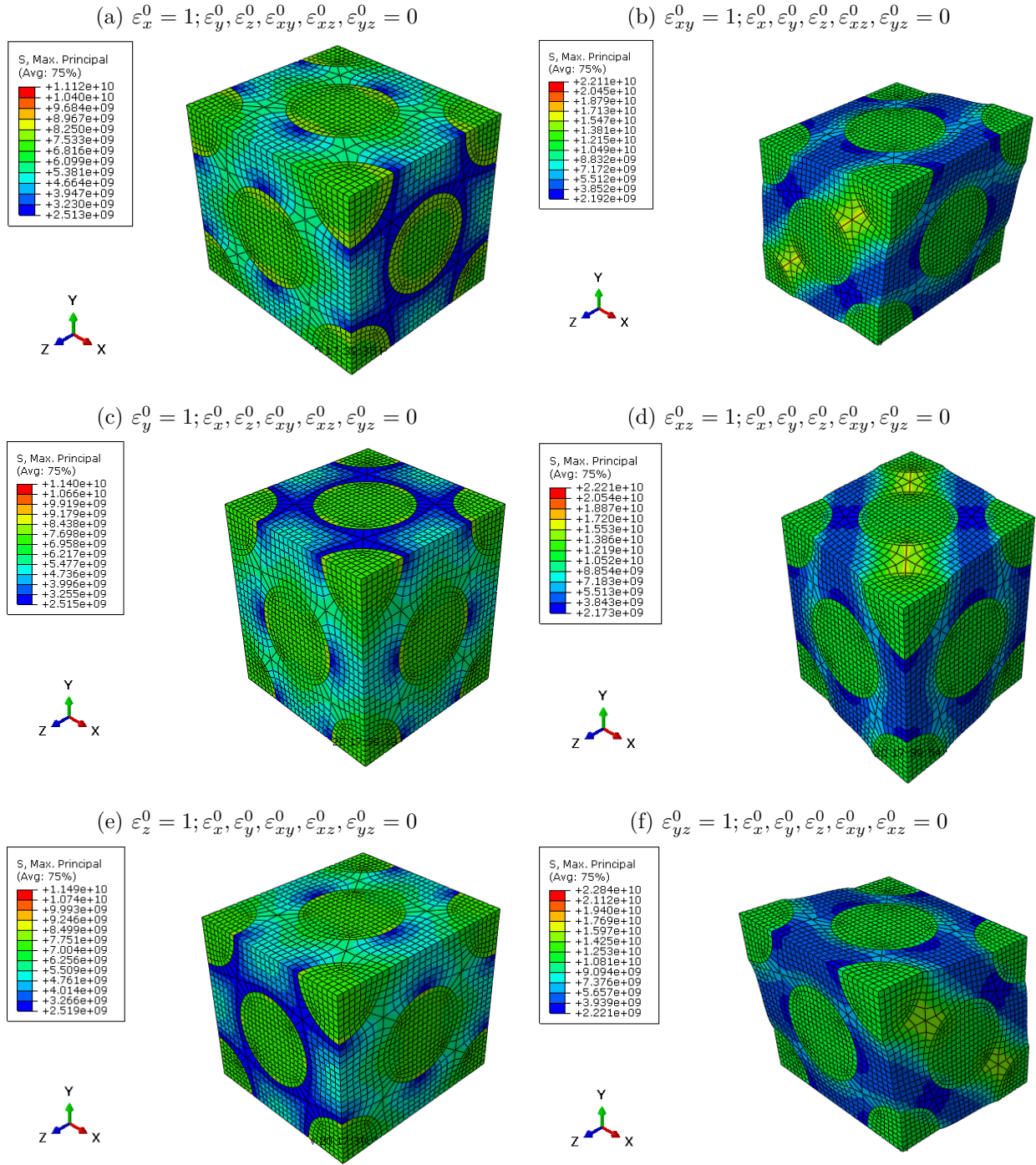


Figura 16 – Tensão máxima principal resultante para casos de carga de deformação global aplicados nos *dummy nodes*

Os modelos apresentados nesta seção e suas correspondentes etapas de geração podem ser replicados facilmente com as funções apresentadas no Apêndice deste trabalho. Com os resultados e modelos apresentados, finaliza-se a apresentação dos modelos resultantes e dos campos de deformação aplicados sobre eles, tanto térmicos como mecânicos.

4.2 Comparação com dados experimentais e numéricos

Neste momento, o único passo restante para a avaliação e validação dos algoritmos é a geração dos resultados numéricos e comparação com dados da literatura, principalmente os apresentados em [Tessier-Doyen et al., 2006]. Os resultados apresentados a seguir foram obtidos a partir da utilização do algoritmo implementado no *plug-in Report*, responsável pela criação de arquivos de texto que contêm todos os valores de deformação e de tensão para cada um dos elementos finitos componentes da malha dos modelos. A aplicação das técnicas de homogeneização apresentadas em 2.5 e 2.6 pressupõem a integração das deformações e das tensões sobre o VRE. Para um modelo discreto, tal qual o caso de uma simulação MEF, estas equações podem ser substituídas pelo somatório em todos os elementos componentes, desde que haja uma consideração quanto ao tamanho do elemento finito resultante.

Nas Figuras 17, 18 e 19 são apresentados os resultados obtidos para os modelos gerados considerando todos os três arranjos espaciais (C, CCC e CFC) bem como para todas as frações volumétricas (15%, 30% e 45%). Apresentam, ainda, os resultados experimentais e o limite inferior teórico descrito em [Tessier-Doyen et al., 2006] para as propriedades efetivas. O caso de modelagem matriz G3 com inclusões de alumina é o mais interessante, do ponto de vista experimental, haja visto que, como a alumina possui coeficiente de deformação térmica superior ao da matriz vítrea, o resfriamento dos corpos de prova ensaiados ocasiona o surgimento de trincas radiais com relação à interface matriz/inclusão. Este caso de surgimento de trincas é o mais estudado historicamente, existindo modelos matemáticos analíticos de determinação deste fenômeno. Além disto, como exposto em [Tessier-Doyen et al., 2006] o módulo de Young resultante para um determinado VRE é mais sensível a trincas na matriz do que a vazios de descolamento interfaciais. O resfriamento submete as inclusões a um estado de compressão radial e tensão circunferencial, provocando o surgimento de trincas na matriz, já que a deformação plástica da mesma não é capaz de suportar o deslocamento relativo dos materiais na interface.

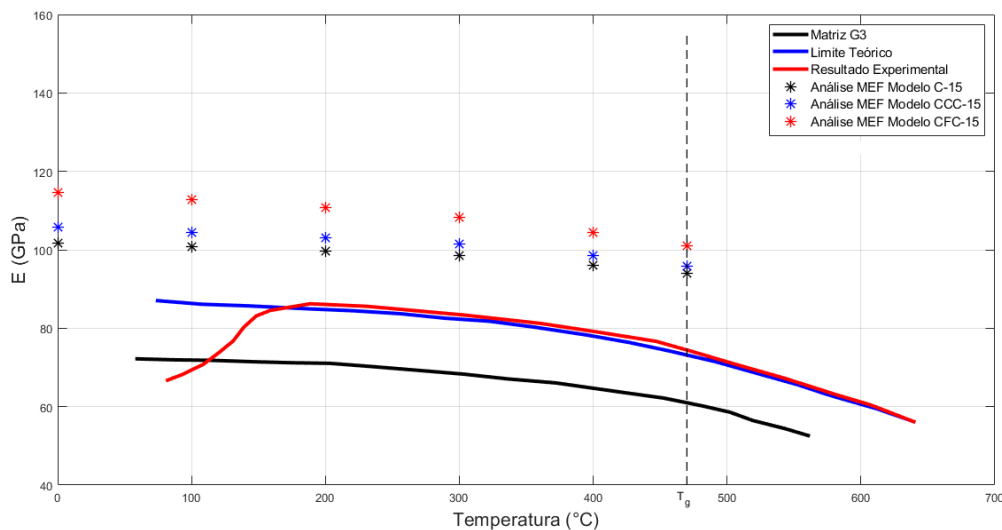


Figura 17 – Comparativo entre os resultados da modelagem MEF com os teóricos e experimentais de [Tessier-Doyen et al., 2006] para os modelos de matriz G3 e $\phi = 15\%$

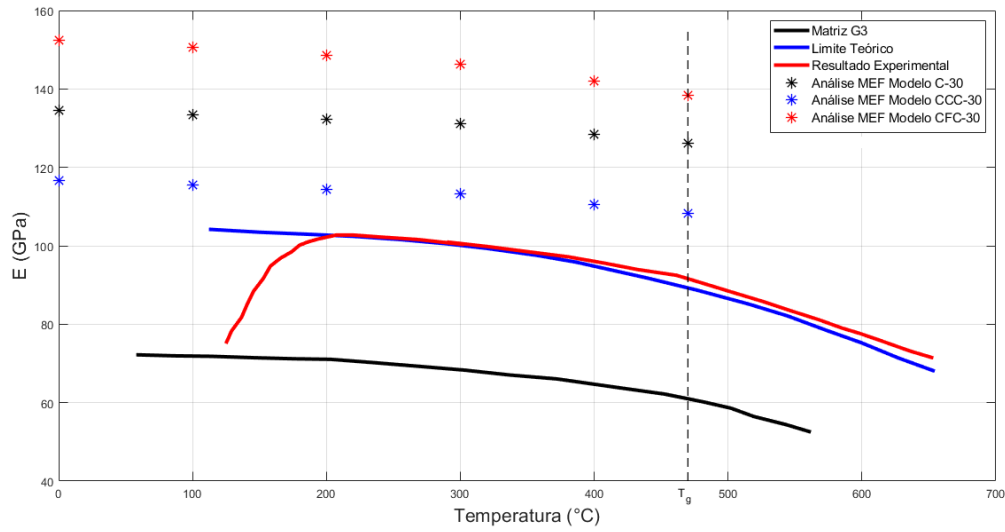


Figura 18 – Comparativo entre os resultados da modelagem MEF com os teóricos e experimentais de [Tessier-Doyen et al., 2006] para os modelos de matriz G3 e $\phi = 30\%$

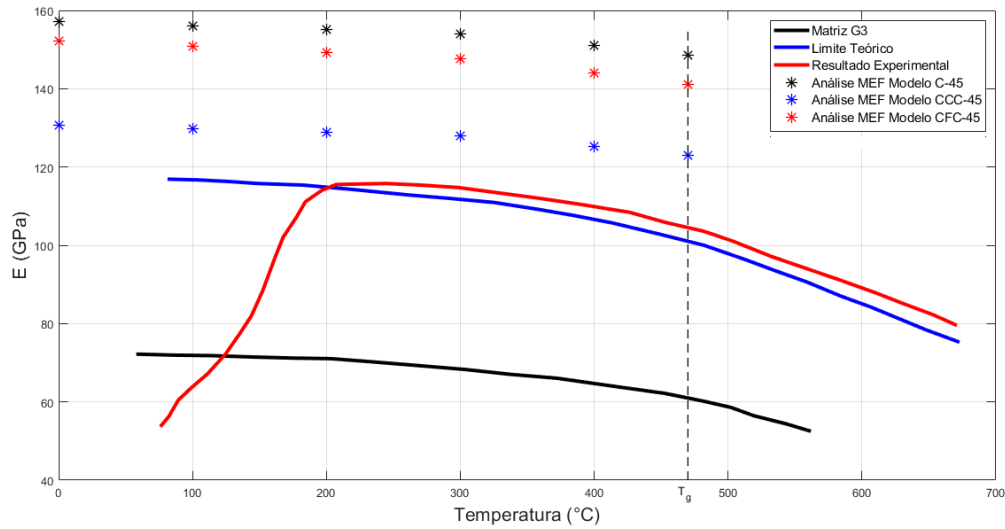


Figura 19 – Comparativo entre os resultados da modelagem MEF com os teóricos e experimentais de [Tessier-Doyen et al., 2006] para os modelos de matriz G3 e $\phi = 45\%$

Para o caso da matriz G1, o fenômeno de descolamento da interface ainda se apresenta com grande dificuldade de descrição matemática. Já o caso G2 não apresenta efeitos significativos de concentração de tensão devido ao gradiente de temperatura e, por isto, não se apresenta com grande interesse para a execução deste trabalho. Mesmo assim, também foram gerados resultados para os dois últimos casos, de forma a se apresentar certa generalidade na aplicação dos algoritmos desenvolvidos. Entretanto, o número de simulações realizadas se restringiu somente aos modelos de fração volumétrica de 30%. Estes resultados são apresentados nas Figuras 20 e 21.

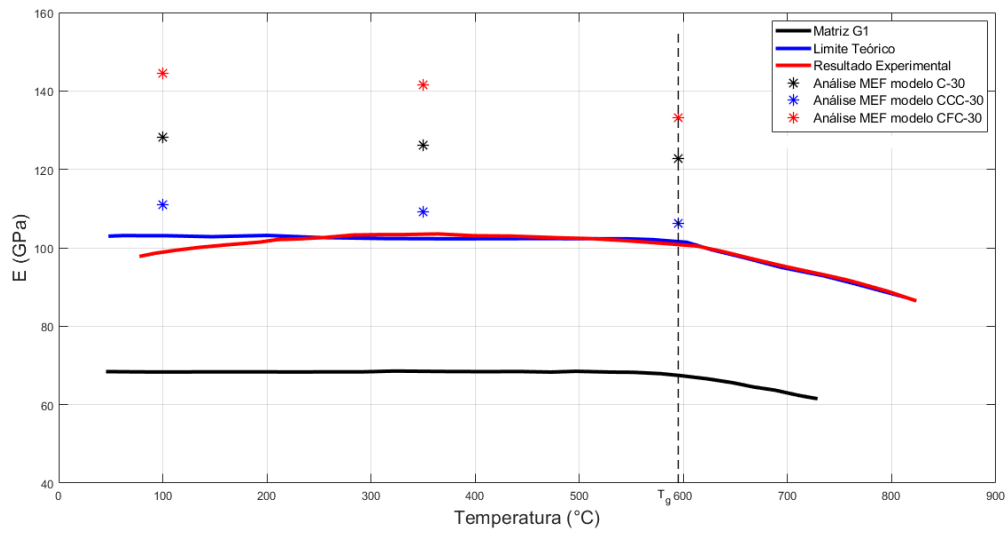


Figura 20 – Comparativo entre os resultados da modelagem MEF com os teóricos e experimentais [Tessier-Doyen et al., 2006] para os modelos de matriz G1 e $\phi=30\%$.

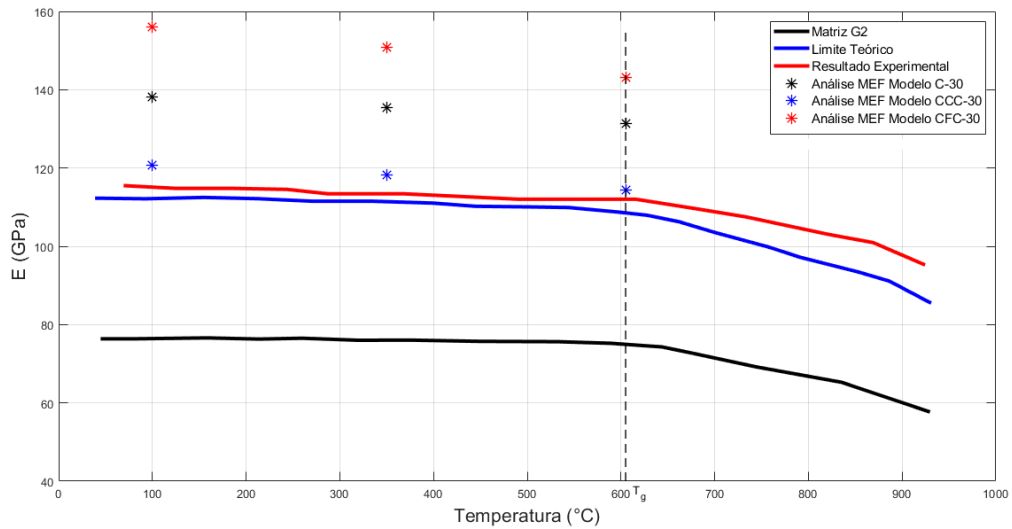


Figura 21 – Comparativo entre os resultados da modelagem MEF com os teóricos e experimentais [Tessier-Doyen et al., 2006] para os modelos de matriz G2 e $\phi=30\%$.

Nota-se que os resultados obtidos pela modelagem MEF se apresentam superiores a ambos os resultados experimentais e teóricos apresentados na literatura. Cabe aqui, portanto, a discussão de dois pontos bastante importantes. O primeiro é o de que os resultados obtidos pela modelagem em elementos finitos não apresentam qualquer consideração sobre os efeitos de descolamento ou trincamento que ocorre nas interfaces matriz/inclusão. Estes trincamentos/descolamentos ocasionam significativa perda de resistência e rigidez dos materiais ensaiados, bem como promovem um efeito não linear nas curvas de evolução dos módulos de elasticidade pela temperatura. Um outro ponto que deve ser observado é a dependência dos resultados das propriedades efetivas obtidos com a aplicação das condições de periodicidade das malha de elementos finitos. Como

estas condições de periodicidade são aplicadas na superfície dos modelos, a área superficial ocupada pelas inclusões ou pela matriz ocasiona um efeito de elevação do módulos de elasticidade efetivos obtidos. Este efeito, apesar de não muito significativo, poderia ser evitado se os modelos construídos possuísem todas as suas inclusões imersas na matriz, de forma a que a superfície dos modelos serem compostas somente pelo material da matriz. Esta abordagem, entretanto, tem impacto na determinação estatística das propriedades e, portanto, deve ser evitada sua aplicação de forma individual. A alternativa que se apresentou mais promissora foi a utilização de diversos arranjos geométricos para a disposição das inclusões no interior da matriz, de forma que se possua modelos que possuam tanto inclusões na superfície como modelos que possuam somente superfície de material da matriz e a posterior utilização destes valores para a obtenção de propriedades médias. Neste trabalho, entretanto, devido à escassez de tempo hábil, somente se realizou a simulação dos modelos já apresentados anteriormente.

A Tabela 1 apresenta os erros percentuais entre as simulações MEF e os resultados teóricos da literatura.

Tabela 1 – Erros percentuais entre os valores obtidos pela simulação MEF e os valores teóricos para a matriz G3.

	15%	30%	45%
C	13	22	25
CCC	17	6	4
CFC	27	38	22

Mesmo com todas as ressalvas feitas anteriormente, nota-se que os erros percentuais encontrados são relativamente baixos quando comparados com os valores teóricos/experimentais de limite inferior descritos na literatura, demonstrando que a metodologia aplicada e os algoritmos gerados possuem grande precisão na descrição do fenômeno proposto. Nota-se que os modelos do tipo CCC são aqueles que apresentam maior concordância de seus resultados, com consequente erro relativo menor. Este fato pode ser explicado pela menor área superficial do modelo que é composta pelo material da inclusão. Faz-se necessário lembrar que as condições de periodicidade são aplicados aos nós da superfície do modelo e, portanto, quanto maior a razão de área superficial ocupada pelas inclusões em comparação à matriz, maior será o efeito de elevação do módulo de elasticidade efetivo, haja visto que as inclusões possuem módulo de elasticidade mais alto que a matriz.

CAPÍTULO 5

CONCLUSÕES E PERSPECTIVAS

O presente trabalho dedicou-se ao desenvolvimento de rotinas automatizadas para a geração de experimentos virtuais para a determinação de propriedades efetivas de materiais compósitos que apresentam propriedades térmicas e mecânicas distintas entre as fases constituintes. O foco principal foi no estudo do campo de tensões residual que existe em materiais compósitos submetidos a uma variação de temperatura.

Discutiu-se a respeito dos métodos de geração de microestruturas artificiais bem como a necessidade de representatividade estatística do comportamento mecânico obtido pelos volumes elementares representativos. Este trabalho também tenta apresentar a dificuldade e complexidade da aplicação das condições de periodicidade aos modelos MEF para a realização das simulações e desenvolve, baseado nas formulações da literatura, o equacionamento geral a ser aplicado aos nós da malha de elementos finitos. Apresentou-se também a aplicação de técnicas de homogeneização para a determinação das propriedades efetivas dos modelos, partindo das relações constitutivas do comportamento mecânico dos materiais, relacionando o comportamento microscópico ao macroscópico dos modelos.

Nota-se um melhor desempenho da microestrutura CCC para as configurações estudadas sendo que ela apresenta a menor diferença percentual quando os seus resultados com os experimentais da literatura são comparados, sendo de 4% de diferença para o módulo de Young previsto pelo modelo em elementos finitos com condições de contorno periódicas, para o caso da matriz G3 com 45% de fração volumétrica. Já a configuração microestrutural CFC foi a que apresentou desempenho mais destoante, com diferença de 38% para a mesma matriz com fração volumétrica de 30%.

Durante a execução do presente trabalho foram encontrados diversos pontos de dificuldade e melhorias que podem ser mitigados com a implementação de melhorias nos algoritmos aqui apresentados. Dentre estes pontos pode-se citar: a dificuldade na aplicação das condições de periodicidade, dado que a formulação encontrada na literatura muitas vezes é apresentada de forma incompleta ou mesmo dada como conhecida; a dependência dos resultados obtidos com as características da superfícies dos modelos MEF utilizados, que determinam, até certo ponto, a

influência dos módulos de elasticidade das fases de maior área superficial no módulo resultante final do modelo; e a dificuldade de se encontrar resultados experimentais confiáveis e abrangentes que sirvam para a validação dos modelos numéricos desenvolvidos.

Para trabalhos futuros se propõem sugestões de implementação que possam melhorar os resultados obtidos, tanto em precisão quando comparado à teoria, quanto em representatividade estatística das propriedades efetivas obtidas dos modelos MEF. A principal sugestão é a implementação de um algoritmo de construção de modelos onde as inclusões estejam completamente imersas na fase da matriz, reduzindo ou eliminando as áreas superficiais que influenciam os valores das propriedades efetivas computadas. Além disto é de grande interesse a implementação de estruturas não regulares, pela disposição aleatória das inclusões no interior dos modelos de VRE.

REFERÊNCIAS

- [Aboudi et al., 2013] Aboudi, J., Arnold, S., and Bednarczyk, B. (2013). *Micromechanics of Composite Materials: A Generalized Multiscale Analysis Approach*.
- [Ashby, 2005] Ashby, M. F. (2005). *Materials Selection in Mechanical Design*.
- [Bargmann et al., 2018] Bargmann, S., Klusemann, B., Markmann, J., Schnabel, J. E., Schneider, K., Soyarslan, C., and Wilmers, J. (2018). Generation of 3d representative volume elements for heterogeneous materials: A review. *Progress in Materials Science*, 96:322–384.
- [Bensoussan et al., 2011] Bensoussan, A., Lions, J.-L., and Papanicolaou, G. (2011). *Asymptotic analysis for periodic structures*, volume 374. American Mathematical Soc.
- [Callister, 2008] Callister, W. (2008). *Ciência e engenharia de materiais: uma introdução*. Livros Técnicos e Científicos.
- [Danielsson et al., 2002] Danielsson, M., M. Parks, D., and C. Boyce, M. (2002). Three-dimensional micromechanical modeling of voided polymeric materials. *Journal of The Mechanics and Physics of Solids - J MECH PHYS SOLIDS*, 50:351–379.
- [Davidge and Green, 1968] Davidge, R. W. and Green, T. D. (1968). The Strength of Two-Phase Ceramic/Glass Materials. *Journal of Materials Science*, 3:629–634.
- [DIETRICH, 2002] DIETRICH, S. (2002). Simulation and characterization of random systems of hard particles. *Image Analysis and Stereology*, 21.
- [Drago and Pindera, 2007] Drago, A. and Pindera, M.-J. (2007). Micro-macromechanical analysis of heterogeneous materials: Macroscopically homogeneous vs periodic microstructures. *Composites Science and Technology - COMPOSITES SCI TECHNOL*, 67:1243–1263.
- [Fan and Wang, 2017] Fan, Y. and Wang, H. (2017). A simple python code for computing effective properties of 2d and 3d representative volume element under periodic boundary conditions.

- [Hollaway, 2000] Hollaway, L.C., H. P. (2000). Composite materials and structures in civil engineering. *Compr Compos Mater*, 6:489–527.
- [Jaeger and Nagel, 1992] Jaeger, H. M. and Nagel, S. R. (1992). Physics of the granular state. *Science*, 255(5051):1523–1531.
- [King, 2018] King, J. E. (2018). Failure in composite materials.
- [Luchini et al., 2016] Luchini, B., Sciuti, V. F., Angélico, R. A., Canto, R. B., and Pandolfelli, V. C. (2016). Thermal expansion mismatch inter-inclusion cracking in ceramic systems. *Ceramics International*, 42(10):12512–12515.
- [Luchini et al., 2017] Luchini, B., Sciuti, V. F., Angélico, R. A., Canto, R. B., and Pandolfelli, V. C. (2017). Critical inclusion size prediction in refractory ceramics via finite element simulations. *Journal of the European Ceramic Society*, 37(1):315–321.
- [Ostoja-Starzewski, 2006] Ostoja-Starzewski, M. (2006). Ostoja-starzewski, m.: Material spatial randomness: from statistical to representative volume element. *probab. eng. mech.* 21(2), 112–132. *Probabilistic Engineering Mechanics*, 21:112–132.
- [Sparavigna, 2011] Sparavigna, A. C. (2011). Ancient concrete works.
- [Stoneham and Harding, 2009] Stoneham, A. M. and Harding, J. H. (2009). Invited review: Mesoscopic modelling: materials at the appropriate scale. *Materials Science and Technology*, 25(4):460–465.
- [teh Sun and Vaidya, 1996] teh Sun, C. and Vaidya, R. S. (1996). Prediction of composite properties from a representative volume element.
- [Tessier-Doyen et al., 2006] Tessier-Doyen, N., Glandus, J., and Huger, M. (2006). Untypical young’s modulus evolution of model refractories at high temperature. *Journal of the European Ceramic Society*, 26:289–295.
- [Torquato, 2013] Torquato, S. (2013). *Random heterogeneous materials: microstructure and macroscopic properties*, volume 16. Springer Science & Business Media.
- [VISSCHER and BOLSTERLI, 1972] VISSCHER, W. M. and BOLSTERLI, M. (1972). Random packing of equal and unequal spheres in two and three dimensions. *Nature*, 239:504–507.
- [Williams and Philipse, 2003] Williams, S. R. and Philipse, A. P. (2003). Random packings of spheres and spherocylinders simulated by mechanical contraction. *Phys. Rev. E*, 67:051301.
- [Xia et al., 2003] Xia, Z., Zhang, Y., and Ellyin, F. (2003). A unified periodical boundary conditions for representative volume elements of composites and applications. *International Journal of Solids and Structures*, 40:1907–1921.
- [Xu and Xu, 2008] Xu, K. and Xu, X. (2008). Finite element analysis of mechanical properties of 3d five-directional braided composites. *Materials Science and Engineering: A*, 487:499–509.

- [Zago, 2019] Zago, I. P. (2019). Estudo do comportamento mecânico de compósitos modelo submetidos à variação de temperatura via correlação de imagens digitais.
- [Zhang et al., 2018] Zhang, X., Chen, Y., and Hu, J. (2018). Recent advances in the development of aerospace materials. *Progress in Aerospace Sciences*, 97:22–34.

APÊNDICE A

CÓDIGO EM PYTHON PARA A GERAÇÃO DAS GEOMETRIAS MODELO E APLICAÇÃO DAS CONDIÇÕES DE PERIODICIDADE

A seguir são apresentados os três arquivos em *Python* utilizados para a geração e automatização dos procedimentos de modelagem e de aplicação das condições de contorno periódicas aos modelos. Como já citado no decorrer deste texto, optou-se pela separação do código em três arquivos diferentes, de forma a facilitar os procedimentos de correção e alteração dos códigos. Os códigos gerados são apresentados na mesma ordem em que devem ser executados: *Cell*, *Mirror* e *Constraints*.

A.1 Cell

```

from abaqus import *
from abaqusConstants import *
from caeModules import *
from driverUtils import executeOnCaeStartup
def cell(tipo, fraction, matrix, meshsize):
    if tipo == 'CCC':
        sime = float(meshsize)
        sizemesh = 9.6e-05/(2*sime)
        if fraction == '15':
            a = 955.8075e-6
        if fraction == '30':
            a = 758.47e-6
        if fraction == '45':
            a = 662.585e-6
        executeOnCaeStartup()
        s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
            sheetSize=200.0)
        g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
        s.setPrimaryObject(option=STANDALONE)
        #Parts Creation
        s.Spot(point=(0.0, 0.0))
        s.FixedConstraint(entity=v[0])
        s.rectangle(point1=(0.0, 0.0), point2=(13.75, 13.75))
        s.EqualLengthConstraint(entity1=g[2], entity2=g[3])
        s.ObliqueDimension(vertex1=v[4], vertex2=v[5], textPoint=(a, 0.0),
            value=a)
        p = mdb.models['Model-1'].Part(name='Part-1', dimensionality=THREE_D,
            type=DEFORMABLE_BODY)
        p = mdb.models['Model-1'].parts['Part-1']
        p.BaseSolidExtrude(sketch=s, depth=a)
        s.unsetPrimaryObject()

```

```

p = mdb.models['Model-1'].parts['Part-1']
del mdb.models['Model-1'].sketches['__profile__']
s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
sheetSize=200.0)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=STANDALONE)
s1.ConstructionLine(point1=(0.0, -100.0), point2=(0.0, 100.0))
s1.FixedConstraint(entity=g[2])
s1.Spot(point=(0.0, 0.0))
s1.CoincidentConstraint(entity1=v[0], entity2=g[2], addUndoState=False)
s1.FixedConstraint(entity=v[0])
s1.Line(point1=(16.25, 0.0), point2=(0.0, 0.0))
s1.HorizontalConstraint(entity=g[3], addUndoState=False)
s1.Line(point1=(0.0, 0.0), point2=(0.0, 19.2429962158203))
s1.VerticalConstraint(entity=g[4], addUndoState=False)
s1.PerpendicularConstraint(entity1=g[3], entity2=g[4], addUndoState=False)
s1.CoincidentConstraint(entity1=v[4], entity2=g[2], addUndoState=False)
s1.EqualLengthConstraint(entity1=g[4], entity2=g[3])
s1.ArcByCenterEnds(center=(0.0, 0.0), point1=(0.0, 19.2429962158203), point2=(
19.2429962158203, 0.0), direction=CLOCKWISE)
s1.ObliqueDimension(vertex1=v[1], vertex2=v[2], textPoint=(10.7691106796265,
-8.25650024414063), value=0.0005)
p = mdb.models['Model-1'].Part(name='Part-2', dimensionality=THREE_D,
type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-2']
p.BaseSolidRevolve(sketch=s1, angle=90.0, flipRevolveDirection=OFF)
s1.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-2']
del mdb.models['Model-1'].sketches['__profile__']
a = mdb.models['Model-1'].rootAssembly
a = mdb.models['Model-1'].rootAssembly
a.DatumCsysByDefault(CARTESIAN)
p = mdb.models['Model-1'].parts['Part-1']
a.Instance(name='Part-1-1', part=p, dependent=ON)
p = mdb.models['Model-1'].parts['Part-2']
a.Instance(name='Part-2-1', part=p, dependent=ON)
p1 = a.instances['Part-2-1']
p1.translate(vector=(0.00100580750000034, 0.0, 0.0))
a = mdb.models['Model-1'].rootAssembly
p = mdb.models['Model-1'].parts['Part-2']
a.Instance(name='Part-2-2', part=p, dependent=ON)
p1 = a.instances['Part-2-2']
p1.translate(vector=(0.00155580749999906, 0.0, 0.0))
a = mdb.models['Model-1'].rootAssembly
d11 = a.instances['Part-2-2'].datums
e1 = a.instances['Part-1-1'].edges
a.ParallelEdge(movableAxis=d11[1], fixedAxis=e1[7], flip=ON)
a = mdb.models['Model-1'].rootAssembly
v11 = a.instances['Part-2-2'].vertices
v12 = a.instances['Part-1-1'].vertices
a.CoincidentPoint(movablePoint=v11[1], fixedPoint=v12[4])
a = mdb.models['Model-1'].rootAssembly
v11 = a.instances['Part-2-2'].vertices
v12 = a.instances['Part-1-1'].vertices
a.CoincidentPoint(movablePoint=v11[1], fixedPoint=v12[3])
a = mdb.models['Model-1'].rootAssembly
a.InstanceFromBooleanMerge(name='Part-3', instances=(a.instances['Part-1-1'],
a.instances['Part-2-1'], a.instances['Part-2-2'], ), keepIntersections=ON,
originalInstances=SUPPRESS, domain=GEOMETRY)
## Material Definition - Temperature Dependent
p = mdb.models['Model-1'].parts['Part-2']
mdb.models['Model-1'].Material(name='Alumina')
mdb.models['Model-1'].materials['Alumina'].Elastic(temperatureDependency=ON,
table=((326.0, 0.24, 298.0), (317.0, 0.24, 508.0), (300.0, 0.24, 868.0)))
mdb.models['Model-1'].materials['Alumina'].Expansion(table=((7.6e-06, ), ))
if str(matrix) == 'G3':
    mdb.models['Model-1'].Material(name='Glass')
    mdb.models['Model-1'].materials['Glass'].Elastic(temperatureDependency=ON,
table=((72.0e9, 0.23, 337.0), (69.2e9, 0.23, 549.0), (61.7e9, 0.23, 736.0)))
    mdb.models['Model-1'].materials['Glass'].Expansion(table=((1.16e-05, ), ))
if str(matrix) == 'G2':
    mdb.models['Model-1'].Material(name='Glass')
    mdb.models['Model-1'].materials['Glass'].Elastic(temperatureDependency=ON,

```

```

        table=((76.0e9, 0.21, 298.0), (76.0e9, 0.21, 549.0), (76.0e9, 0.21, 868.0)))
        mdb.models['Model-1'].materials['Glass'].Expansion(table=((7.4e-06, ), ))
if str(matrix) == 'G1':
    mdb.models['Model-1'].Material(name='Glass')
    mdb.models['Model-1'].materials['Glass'].Elastic(temperatureDependency=ON,
        table=((68.0e9, 0.2, 298.0), (68.0e9, 0.2, 549.0), (68.0e9, 0.2, 868.0)))
    mdb.models['Model-1'].materials['Glass'].Expansion(table=((4.6e-06, ), ))
mdb.models['Model-1'].HomogeneousSolidSection(name='Section-1',
    material='Alumina', thickness=None)
mdb.models['Model-1'].HomogeneousSolidSection(name='Section-2',
    material='Glass', thickness=None)
p = mdb.models['Model-1'].parts['Part-3']
a = mdb.models['Model-1'].rootAssembly
p = mdb.models['Model-1'].parts['Part-3']
p = mdb.models['Model-1'].parts['Part-3']
c = p.cells
cells = c.getSequenceFromMask(mask=('[#4 ]', ), )
region = p.Set(cells=cells, name='Set-1')
p = mdb.models['Model-1'].parts['Part-3']
p.SectionAssignment(region=region, sectionName='Section-2', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
p = mdb.models['Model-1'].parts['Part-3']
c = p.cells
cells = c.getSequenceFromMask(mask=('[#3 ]', ), )
region = p.Set(cells=cells, name='Set-2')
p = mdb.models['Model-1'].parts['Part-3']
p.SectionAssignment(region=region, sectionName='Section-1', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
## Mesh Definition
p = mdb.models['Model-1'].parts['Part-3']
p.seedPart(size=sizemesh, deviationFactor=0.1, minSizeFactor=0.1)
p = mdb.models['Model-1'].parts['Part-3']
p.generateMesh()
a = mdb.models['Model-1'].rootAssembly
a.regenerate()
## Input File Generation
mdb.Job(name='Cell', model='Model-1', description='', type=ANALYSIS,
    atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90,
    memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
    explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echoPrint=OFF,
    modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='',
    scratch='', resultsFormat=ODB, parallelizationMethodExplicit=DOMAIN,
    numDomains=1, activateLoadBalancing=False, multiprocessingMode=DEFAULT,
    numCpus=1)
mdb.jobs['Cell'].writeInput(consistencyChecking=OFF)
#: The job input file has been written to "Job-1.inp".
#####
elif tipo == 'CFC':
    # Part Creation
    sime = float(meshsize)
    sizemesh = 9.6e-05/(2*sime)

    if fraction == '15':
        a = 1203.998e-6
    if fraction == '30':
        a = 955.614e-6
    if fraction == '45':
        a = 834.8056e-6
    executeOnCaeStartup()
    s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
        sheetSize=200.0)
    g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
    s.setPrimaryObject(option=STANDALONE)
    s.Spot(point=(0.0, 0.0))
    s.FixedConstraint(entity=v[0])
    s.rectangle(point1=(0.0, 0.0), point2=(15.0, 13.75))
    s.EqualLengthConstraint(entity1=g[5], entity2=g[4])
    s.ObliqueDimension(vertex1=v[4], vertex2=v[5], textPoint=(4.39339447021484,
        -11.8999996185303), value=a)
    p = mdb.models['Model-1'].Part(name='Part-1', dimensionality=THREE_D,
        type=DEFORMABLE_BODY)

```

```

p = mdb.models['Model-1'].parts['Part-1']
p.BaseSolidExtrude(sketch=s, depth=a)
s.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-1']
del mdb.models['Model-1'].sketches['__profile__']
s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=STANDALONE)
s1.ConstructionLine(point1=(0.0, -100.0), point2=(0.0, 100.0))
s1.FixedConstraint(entity=g[2])
s1.Spot(point=(0.0, 0.0))
s1.CoincidentConstraint(entity1=v[0], entity2=g[2], addUndoState=False)
s1.Line(point1=(17.5, 0.0), point2=(0.0, 0.0))
s1.HorizontalConstraint(entity=g[3], addUndoState=False)
s1.Line(point1=(0.0, 0.0), point2=(0.0, 15.7499961853027))
s1.VerticalConstraint(entity=g[4], addUndoState=False)
s1.PerpendicularConstraint(entity1=g[3], entity2=g[4], addUndoState=False)
s1.CoincidentConstraint(entity1=v[4], entity2=g[2], addUndoState=False)
s1.EqualLengthConstraint(entity1=g[4], entity2=g[3])
s1.ArcByCenterEnds(center=(0.0, 0.0), point1=(0.0, 15.7499961853027), point2=(
    15.7499961853027, 0.0), direction=CLOCKWISE)
s1.ObliqueDimension(vertex1=v[1], vertex2=v[2], textPoint=(4.92061614990234,
    -13.6499996185303), value=0.0005)
p = mdb.models['Model-1'].Part(name='Part-2', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-2']
p.BaseSolidRevolve(sketch=s1, angle=90.0, flipRevolveDirection=OFF)
s1.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-2']
del mdb.models['Model-1'].sketches['__profile__']
## Material Definition
mdb.models['Model-1'].Material(name='Alumina')
mdb.models['Model-1'].materials['Alumina'].Elastic(temperatureDependency=ON,
    table=((3260000000000.0, 0.24, 298.0), (3090000000000.0, 0.24, 688.0), (
    3020000000000.0, 0.24, 838.0)))
mdb.models['Model-1'].materials['Alumina'].Expansion(table=((7.6e-06, ), ))
if str(matrix) == 'G3':
    mdb.models['Model-1'].Material(name='Glass')
    mdb.models['Model-1'].materials['Glass'].Elastic(temperatureDependency=ON,
        table=((72.0e9, 0.23, 337.0), (69.2e9, 0.23, 549.0), (61.7e9, 0.23, 736.0)))
    mdb.models['Model-1'].materials['Glass'].Expansion(table=((1.16e-05, ), ))
if str(matrix) == 'G2':
    mdb.models['Model-1'].Material(name='Glass')
    mdb.models['Model-1'].materials['Glass'].Elastic(temperatureDependency=ON,
        table=((76.0e9, 0.21, 298.0), (76.0e9, 0.21, 549.0), (76.0e9, 0.21, 868.0)))
    mdb.models['Model-1'].materials['Glass'].Expansion(table=((7.4e-06, ), ))
if str(matrix) == 'G1':
    mdb.models['Model-1'].Material(name='Glass')
    mdb.models['Model-1'].materials['Glass'].Elastic(temperatureDependency=ON,
        table=((68.0e9, 0.2, 298.0), (68.0e9, 0.2, 549.0), (68.0e9, 0.2, 868.0)))
    mdb.models['Model-1'].materials['Glass'].Expansion(table=((4.6e-06, ), ))
mdb.models['Model-1'].HomogeneousSolidSection(name='Section-1',
    material='Alumina', thickness=None)
mdb.models['Model-1'].HomogeneousSolidSection(name='Section-2',
    material='Glass', thickness=None)
a = mdb.models['Model-1'].rootAssembly
a = mdb.models['Model-1'].rootAssembly
a.DatumCsysByDefault(CARTESIAN)
p = mdb.models['Model-1'].parts['Part-1']
a.Instance(name='Part-1-1', part=p, dependent=ON)
p = mdb.models['Model-1'].parts['Part-2']
a.Instance(name='Part-2-1', part=p, dependent=ON)
p1 = a.instances['Part-2-1']
p1.translate(vector=(0.00125399800000121, 0.0, 0.0))
a = mdb.models['Model-1'].rootAssembly
p = mdb.models['Model-1'].parts['Part-2']
a.Instance(name='Part-2-2', part=p, dependent=ON)
p1 = a.instances['Part-2-2']
p1.translate(vector=(0.00180399800000188, 0.0, 0.0))
a = mdb.models['Model-1'].rootAssembly
p = mdb.models['Model-1'].parts['Part-2']
a.Instance(name='Part-2-3', part=p, dependent=ON)

```

```

p1 = a.instances['Part-2-3']
p1.translate(vector=(0.002353998000000255, 0.0, 0.0))
a = mdb.models['Model-1'].rootAssembly
d11 = a.instances['Part-2-1'].datums
e1 = a.instances['Part-1-1'].edges
a.ParallelEdge(movableAxis=d11[1], fixedAxis=e1[7], flip=ON)
session.viewports['Viewport: 1'].view.setValues(nearPlane=0.00524128,
    farPlane=0.0101367, width=0.00399707, height=0.00138687,
    viewOffsetX=4.94979e-005, viewOffsetY=-6.91218e-005)
a = mdb.models['Model-1'].rootAssembly
v11 = a.instances['Part-2-1'].vertices
v12 = a.instances['Part-1-1'].vertices
a.CoincidentPoint(movablePoint=v11[1], fixedPoint=v12[4])
a = mdb.models['Model-1'].rootAssembly
e1 = a.instances['Part-2-2'].edges
e2 = a.instances['Part-1-1'].edges
a.ParallelEdge(movableAxis=e1[0], fixedAxis=e2[10], flip=OFF)
a = mdb.models['Model-1'].rootAssembly
v11 = a.instances['Part-2-2'].vertices
v12 = a.instances['Part-1-1'].vertices
a.CoincidentPoint(movablePoint=v11[1], fixedPoint=v12[0])
a = mdb.models['Model-1'].rootAssembly
e1 = a.instances['Part-2-3'].edges
e2 = a.instances['Part-1-1'].edges
a.ParallelEdge(movableAxis=e1[1], fixedAxis=e2[8], flip=ON)
a = mdb.models['Model-1'].rootAssembly
v11 = a.instances['Part-2-3'].vertices
v12 = a.instances['Part-1-1'].vertices
a.CoincidentPoint(movablePoint=v11[1], fixedPoint=v12[7])
a = mdb.models['Model-1'].rootAssembly
a = mdb.models['Model-1'].rootAssembly
p = mdb.models['Model-1'].parts['Part-2']
a.Instance(name='Part-2-4', part=p, dependent=ON)
p = a.instances['Part-2-4']
p.translate(vector=(0.001253998000000121, 0.0, 0.0))
a = mdb.models['Model-1'].rootAssembly
d1 = a.instances['Part-2-4'].datums
e1 = a.instances['Part-1-1'].edges
a.ParallelEdge(movableAxis=d1[1], fixedAxis=e1[2], flip=OFF)
a = mdb.models['Model-1'].rootAssembly
e1 = a.instances['Part-2-4'].edges
e2 = a.instances['Part-1-1'].edges
a.ParallelEdge(movableAxis=e1[1], fixedAxis=e2[1], flip=ON)
a = mdb.models['Model-1'].rootAssembly
v1 = a.instances['Part-2-4'].vertices
v2 = a.instances['Part-1-1'].vertices
a.CoincidentPoint(movablePoint=v1[1], fixedPoint=v2[2])
#: The instance "Part-2-4" is fully constrained
a = mdb.models['Model-1'].rootAssembly
a.InstanceFromBooleanMerge(name='Part-3', instances=(a.instances['Part-1-1'],
    a.instances['Part-2-1'], a.instances['Part-2-2'], a.instances['Part-2-3'],
    a.instances['Part-2-4'], ), keepIntersections=ON,
    originalInstances=SUPPRESS, domain=GEOMETRY)
p = mdb.models['Model-1'].parts['Part-1']
p = mdb.models['Model-1'].parts['Part-3']
p = mdb.models['Model-1'].parts['Part-3']
c = p.cells
cells = c.getSequenceFromMask(mask=('[#10 ]', ), )
region = p.Set(cells=cells, name='Set-1')
p = mdb.models['Model-1'].parts['Part-3']
p.SectionAssignment(region=region, sectionName='Section-2', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
p = mdb.models['Model-1'].parts['Part-3']
c = p.cells
cells = c.getSequenceFromMask(mask=('[#f ]', ), )
region = p.Set(cells=cells, name='Set-2')
p = mdb.models['Model-1'].parts['Part-3']
p.SectionAssignment(region=region, sectionName='Section-1', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
a = mdb.models['Model-1'].rootAssembly
a.regenerate()

```

```

a = mdb.models['Model-1'].rootAssembly
mdb.Job(name='Cell', model='Model-1', description='', type=ANALYSIS,
        atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90,
        memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
        explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echoPrint=OFF,
        modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='',
        scratch='', resultsFormat=ODB, parallelizationMethodExplicit=DOMAIN,
        numDomains=1, activateLoadBalancing=False, multiprocessingMode=DEFAULT,
        numCpus=1)
p = mdb.models['Model-1'].parts['Part-3']
p = mdb.models['Model-1'].parts['Part-3']
## Mesh Definition
p.seedPart(size=sizemesh, deviationFactor=0.1, minSizeFactor=0.1)
p = mdb.models['Model-1'].parts['Part-3']
p.generateMesh()
a = mdb.models['Model-1'].rootAssembly
a.regenerate()
a = mdb.models['Model-1'].rootAssembly
a = mdb.models['Model-1'].rootAssembly
mdb.jobs['Cell'].writeInput(consistencyChecking=OFF)
#: The job input file has been written to "Cell.inp".
if tipo == 'C':
    sime = float(meshsize)
    sizemesh = 9.6e-05/(2*sime)

    if fraction == '15':
        a = 758.47e-6
    if fraction == '30':
        a = 602e-6
    if fraction == '45':
        a = 525.895e-6
    executeOnCaeStartup()
    s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
        sheetSize=200.0)
    g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
    s.setPrimaryObject(option=STANDALONE)
    s.Spot(point=(0.0, 0.0))
    s.FixedConstraint(entity=v[0])
    s.rectangle(point1=(0.0, 0.0), point2=(13.75, 12.5))
    s.EqualLengthConstraint(entity1=g[5], entity2=g[4])
    s.ObliqueDimension(vertex1=v[4], vertex2=v[5], textPoint=(5.62355804443359,
        -7.70000076293945), value=a)
    p = mdb.models['Model-1'].Part(name='Part-1', dimensionality=THREE_D,
        type=DEFORMABLE_BODY)
    p = mdb.models['Model-1'].parts['Part-1']
    p.BaseSolidExtrude(sketch=s, depth=a)
    s.unsetPrimaryObject()
    p = mdb.models['Model-1'].parts['Part-1']
    del mdb.models['Model-1'].sketches['__profile__']
    s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
        sheetSize=200.0)
    g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
    s1.setPrimaryObject(option=STANDALONE)
    s1.ConstructionLine(point1=(0.0, -100.0), point2=(0.0, 100.0))
    s1.FixedConstraint(entity=g[2])
    s1.Spot(point=(0.0, 0.0))
    s1.CoincidentConstraint(entity1=v[0], entity2=g[2], addUndoState=False)
    s1.FixedConstraint(entity=v[0])
    s1.Line(point1=(17.5, 0.0), point2=(0.0, 0.0))
    s1.HorizontalConstraint(entity=g[3], addUndoState=False)
    s1.Line(point1=(0.0, 0.0), point2=(0.0, 22.5749969482422))
    s1.VerticalConstraint(entity=g[4], addUndoState=False)
    s1.PerpendicularConstraint(entity1=g[3], entity2=g[4], addUndoState=False)
    s1.CoincidentConstraint(entity1=v[4], entity2=g[2], addUndoState=False)
    s1.EqualLengthConstraint(entity1=g[4], entity2=g[3])
    s1.ArcByCenterEnds(center=(0.0, 0.0), point1=(0.0, 22.5749969482422), point2=(
        22.5749969482422, 0.0), direction=CLOCKWISE)
    s1.ObliqueDimension(vertex1=v[1], vertex2=v[2], textPoint=(12.4772567749023,
        -13.649997711816), value=0.0005)
    p = mdb.models['Model-1'].Part(name='Part-2', dimensionality=THREE_D,
        type=DEFORMABLE_BODY)
    p = mdb.models['Model-1'].parts['Part-2']
    p.BaseSolidRevolve(sketch=s1, angle=90.0, flipRevolveDirection=OFF)

```

```

s1.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-2']
del mdb.models['Model-1'].sketches['__profile__']
if str(matrix) == 'G3':
    mdb.models['Model-1'].Material(name='Glass')
    mdb.models['Model-1'].materials['Glass'].Elastic(temperatureDependency=ON,
        table=((72.0e9, 0.23, 337.0), (69.2e9, 0.23, 549.0), (61.7e9, 0.23, 736.0)))
    mdb.models['Model-1'].materials['Glass'].Expansion(table=((1.16e-05, ), ))
if str(matrix) == 'G2':
    mdb.models['Model-1'].Material(name='Glass')
    mdb.models['Model-1'].materials['Glass'].Elastic(temperatureDependency=ON,
        table=((76.0e9, 0.21, 298.0), (76.0e9, 0.21, 549.0), (76.0e9, 0.21, 868.0)))
    mdb.models['Model-1'].materials['Glass'].Expansion(table=((7.4e-06, ), ))
if str(matrix) == 'G1':
    mdb.models['Model-1'].Material(name='Glass')
    mdb.models['Model-1'].materials['Glass'].Elastic(temperatureDependency=ON,
        table=((68.0e9, 0.2, 298.0), (68.0e9, 0.2, 549.0), (68.0e9, 0.2, 868.0)))
    mdb.models['Model-1'].materials['Glass'].Expansion(table=((4.6e-06, ), ))
mdb.models['Model-1'].Material(name='Alumina')
mdb.models['Model-1'].materials['Alumina'].Elastic(temperatureDependency=ON,
    table=((326000000000.0, 0.24, 298.0), (309000000000.0, 0.24, 688.0), (
        302000000000.0, 0.24, 838.0)))
mdb.models['Model-1'].materials['Alumina'].Expansion(table=((7.6e-06, ), ))
mdb.models['Model-1'].HomogeneousSolidSection(name='Section-1',
    material='Alumina', thickness=None)
mdb.models['Model-1'].HomogeneousSolidSection(name='Section-2',
    material='Glass', thickness=None)
a = mdb.models['Model-1'].rootAssembly
a = mdb.models['Model-1'].rootAssembly
a.DatumCsysByDefault(CARTESIAN)
p = mdb.models['Model-1'].parts['Part-1']
a.Instance(name='Part-1-1', part=p, dependent=ON)
p = mdb.models['Model-1'].parts['Part-2']
a.Instance(name='Part-2-1', part=p, dependent=ON)
p1 = a.instances['Part-2-1']
p1.translate(vector=(0.0008084700000000838, 0.0, 0.0))
a = mdb.models['Model-1'].rootAssembly
d11 = a.instances['Part-2-1'].datums
e1 = a.instances['Part-1-1'].edges
a.ParallelEdge(movableAxis=d11[1], fixedAxis=e1[7], flip=ON)
a = mdb.models['Model-1'].rootAssembly
v11 = a.instances['Part-2-1'].vertices
v12 = a.instances['Part-1-1'].vertices
a.CoincidentPoint(movablePoint=v11[1], fixedPoint=v12[4])
a = mdb.models['Model-1'].rootAssembly
a.InstanceFromBooleanMerge(name='Part-3', instances=(a.instances['Part-1-1'],
    a.instances['Part-2-1'], ), keepIntersections=ON,
    originalInstances=SUPPRESS, domain=GEOMETRY)
p = mdb.models['Model-1'].parts['Part-2']
p = mdb.models['Model-1'].parts['Part-3']
p = mdb.models['Model-1'].parts['Part-3']
c = p.cells
cells = c.getSequenceFromMask(mask=('[#2 ]', ), )
region = p.Set(cells=cells, name='Set-1')
p = mdb.models['Model-1'].parts['Part-3']
p.SectionAssignment(region=region, sectionName='Section-2', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
p = mdb.models['Model-1'].parts['Part-3']
c = p.cells
cells = c.getSequenceFromMask(mask=('[#1 ]', ), )
region = p.Set(cells=cells, name='Set-2')
p = mdb.models['Model-1'].parts['Part-3']
p.SectionAssignment(region=region, sectionName='Section-1', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
p = mdb.models['Model-1'].parts['Part-3']
## Mesh
p.seedPart(size=sizemesh, deviationFactor=0.1, minSizeFactor=0.1)
p = mdb.models['Model-1'].parts['Part-3']
p.generateMesh()
a = mdb.models['Model-1'].rootAssembly
a.regenerate()

```

```

mdb.Job(name='Cell', model='Model-1', description='', type=ANALYSIS,
        atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90,
        memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
        explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echoPrint=OFF,
        modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='',
        scratch='', resultsFormat=ODB, parallelizationMethodExplicit=DOMAIN,
        numDomains=1, activateLoadBalancing=False, multiprocessingMode=DEFAULT,
        numCpus=1)
mdb.jobs['Cell'].writeInput(consistencyChecking=OFF)
#: The job input file has been written to "Cell.inp".

```

A.2 Mirror

```

from abaqus import *
from abaqusConstants import *
from caeModules import *
from driverUtils import executeOnCaeStartup
def cellmirror(model):
    modelpath = 'C:/Users/duhju/Desktop/abaqus/modelos/' + model + '.cae'
    executeOnCaeStartup()
    a = mdb.models['Model-1'].rootAssembly
    mdb.ModelFromInputFile(name='Cell', inputFileName='C:/Temp/Cell.inp')
    a = mdb.models['Cell'].rootAssembly
    a = mdb.models['Model-1'].rootAssembly
    del mdb.models['Model-1']
    a = mdb.models['Cell'].rootAssembly
    p1 = mdb.models['Cell'].parts['PART-3']
    p = mdb.models['Cell'].Part(name='PART-1',
                                objectToCopy=mdb.models['Cell'].parts['PART-3'], compressFeatureList=ON,
                                mirrorPlane=XYPLANE)
    p1 = mdb.models['Cell'].parts['PART-3']
    p = mdb.models['Cell'].Part(name='PART-2',
                                objectToCopy=mdb.models['Cell'].parts['PART-3'], compressFeatureList=ON,
                                mirrorPlane=YZPLANE)
    p1 = mdb.models['Cell'].parts['PART-2']
    p = mdb.models['Cell'].Part(name='PART-4',
                                objectToCopy=mdb.models['Cell'].parts['PART-2'], compressFeatureList=ON,
                                mirrorPlane=XYPLANE)
    p1 = mdb.models['Cell'].parts['PART-3']
    p = mdb.models['Cell'].Part(name='PART-5',
                                objectToCopy=mdb.models['Cell'].parts['PART-3'], compressFeatureList=ON,
                                mirrorPlane=XZPLANE)
    p1 = mdb.models['Cell'].parts['PART-5']
    p = mdb.models['Cell'].Part(name='PART-6',
                                objectToCopy=mdb.models['Cell'].parts['PART-5'], compressFeatureList=ON,
                                mirrorPlane=XYPLANE)
    p1 = mdb.models['Cell'].parts['PART-5']
    p = mdb.models['Cell'].Part(name='PART-7',
                                objectToCopy=mdb.models['Cell'].parts['PART-5'], compressFeatureList=ON,
                                mirrorPlane=YZPLANE)
    p1 = mdb.models['Cell'].parts['PART-7']
    p = mdb.models['Cell'].Part(name='PART-8',
                                objectToCopy=mdb.models['Cell'].parts['PART-7'], compressFeatureList=ON,
                                mirrorPlane=XYPLANE)
    a = mdb.models['Cell'].rootAssembly
    a = mdb.models['Cell'].rootAssembly
    p = mdb.models['Cell'].parts['PART-2']
    a.Instance(name='PART-2-1', part=p, dependent=ON)
    p = mdb.models['Cell'].parts['PART-4']
    a.Instance(name='PART-4-1', part=p, dependent=ON)
    p = mdb.models['Cell'].parts['PART-5']
    a.Instance(name='PART-5-1', part=p, dependent=ON)
    p = mdb.models['Cell'].parts['PART-6']
    a.Instance(name='PART-6-1', part=p, dependent=ON)
    p = mdb.models['Cell'].parts['PART-7']
    a.Instance(name='PART-7-1', part=p, dependent=ON)
    p = mdb.models['Cell'].parts['PART-8']
    a.Instance(name='PART-8-1', part=p, dependent=ON)
    a = mdb.models['Cell'].rootAssembly
    a1 = mdb.models['Cell'].rootAssembly
    p = mdb.models['Cell'].parts['PART-1']

```

```

a1.Instance(name='PART-1-1', part=p, dependent=ON)
a1 = mdb.models['Cell'].rootAssembly
a1.InstanceFromBooleanMerge(name='Cell', instances=(a1.instances['PART-3-1'],
    a1.instances['PART-2-1'], a1.instances['PART-4-1'],
    a1.instances['PART-5-1'], a1.instances['PART-6-1'],
    a1.instances['PART-7-1'], a1.instances['PART-8-1'],
    a1.instances['PART-1-1'], ), mergeNodes=BOUNDARY_ONLY,
    nodeMergingTolerance=1e-06, domain=MESH, originalInstances=SUPPRESS)
mdb.models['Cell'].StaticStep(name='Step-1', previous='Initial')
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Step-1')
mdb.saveAs(pathName=modelpath)
#: The model database has been saved.

```

A.3 Constraints

```

from abaqus import *
from abaqusConstants import *
from caeModules import *
from driverUtils import executeOnCaeStartup
def constr(arq):
    executeOnCaeStartup()
    arquivo = "C:/Users/duhju/Desktop/abaqus/modelos/" + arq + ".cae"
    mdb.openAuxMdb(pathName=arquivo)
    mdb.copyAuxMdbModel(fromName='Cell', toName='Cell')
    mdb.closeAuxMdb()
    p1 = mdb.models['Cell'].parts['Cell']
    a = mdb.models['Cell'].rootAssembly
    p = mdb.models['Cell'].parts['Cell']
    a = mdb.models['Cell'].rootAssembly
    #a = mdb.models['Model-1'].rootAssembly
    #del mdb.models['Model-1']
    a = mdb.models['Cell'].rootAssembly
    ##### Reference Points / Sets #####
    a.ReferencePoint(point=(0.0015, 0.0005, 0.0))
    a.ReferencePoint(point=(0.0015, 0.0, 0.0))
    #a.ReferencePoint(point=(0.0015, -0.0005, 0.0))
    r1 = a.referencePoints
    refPoints1=(r1[20], )
    a.Set(referencePoints=refPoints1, name='Set-rp1')
    #: The set 'Set-rp1' has been created (1 reference point).
    #refPoints1=(r1[22], )
    #a.Set(referencePoints=refPoints1, name='Set-rp3')
    #: The set 'Set-rp3' has been created (1 reference point).
    r1 = a.referencePoints
    refPoints1=(r1[21], )
    a.Set(referencePoints=refPoints1, name='Set-rp2')
    #: The set 'Set-rp2' has been created (1 reference point).
    ##### Geometry Limits #####
    p = mdb.models['Cell'].parts['Cell']
    nnodes = len(p.nodes)
    xmin, ymin, zmin, xmax, ymax, zmax = 0, 0, 0, 0, 0, 0
    for k in range(nnodes):
        x,y,z = p.nodes[k].coordinates[0],p.nodes[k].coordinates[1],p.nodes[k].coordinates[2]
        if (x < xmin):
            xmin = x
        if (y < ymin):
            ymin = y
        if (z < zmin):
            zmin = z
        if(x > xmax):
            xmax = x
        if(y > ymax):
            ymax = y
        if(z > zmax):
            zmax = z
    #####
    p = mdb.models['Cell'].parts['Cell']
    qsi = 0.00000001
    nnodes = len(p.nodes)

```

```

n=1
##### Sets Creation #####
##### B Middle Point
for k in range(nodes):
    x,y,z = p.nodes[k].coordinates[0],p.nodes[k].coordinates[1],p.nodes[k].coordinates[2]
    if (x == 0.0 and y == 0.0 and z == 0.0):
        center_node = p.nodes[k:k+1]
        p.Set(nodes=center_node, name='center_node')
##### Boundary Conditions #####
a = mdb.models['Cell'].rootAssembly
a.regenerate()
region = a.instances['Cell-1'].sets['center_node']
# Setar quais os deslocamentos e rotacoes serao restringidos no center point!!!!
mdb.models['Cell'].DisplacementBC(name='BC-1', createStepName='Initial',
    region=region, u1=SET, u2=SET, u3=SET, ur1=UNSET, ur2=UNSET, ur3=UNSET,
    amplitude=UNSET, distributionType=UNIFORM, fieldName='', localCsys=None)
# Setar quais os deslocamentos e rotacoes serao aplicados nos Reference Points!!!!
##### RP 1
region = a.sets['Set-rp1']
mdb.models['Cell'].DisplacementBC(name='BC-rp1', createStepName='Step-1',
    region=region, u1=SET, u2=SET, u3=SET, ur1=UNSET, ur2=UNSET, ur3=UNSET,
    amplitude=UNSET, fixed=OFF, distributionType=UNIFORM, fieldName='', localCsys=None)
##### RP 2
region = a.sets['Set-rp2']
mdb.models['Cell'].DisplacementBC(name='BC-rp2', createStepName='Step-1',
    region=region, u1=SET, u2=SET, u3=SET, ur1=UNSET, ur2=UNSET, ur3=UNSET,
    amplitude=UNSET, fixed=OFF, distributionType=UNIFORM, fieldName='', localCsys=None)
##### Cantos
for k in range(nodes):
    x,y,z = p.nodes[k].coordinates[0],p.nodes[k].coordinates[1],p.nodes[k].coordinates[2]
    if (((x>xmin-qsi)and(x<xmin+qsi))or((x>xmax-qsi)and(x<xmax+qsi)))and\
        (((y>ymin-qsi)and(y<ymin+qsi))or((y>ymin-qsi)and(y<ymin+qsi)))and\
        (((z>zmin-qsi)and(z<zmin+qsi))or((z>zmax-qsi)and(z<zmax+qsi)))):
        if (((x>xmin-qsi)and(x<xmin+qsi))and\
            ((y>ymin-qsi)and(y<ymin+qsi))and\
            ((z>zmin-qsi)and(z<zmin+qsi))):
            corner_a = p.nodes[k:k+1]
            p.Set(nodes=corner_a, name='a_corner')
        if (((x>xmin-qsi)and(x<xmin+qsi))and\
            ((y>ymin-qsi)and(y<ymin+qsi))and\
            ((z>zmax-qsi)and(z<zmax+qsi))):
            corner_e = p.nodes[k:k+1]
            p.Set(nodes=corner_e, name='e_corner')
        if (((x>xmin-qsi)and(x<xmin+qsi))and\
            ((y>ymin-qsi)and(y<ymin+qsi))and\
            ((z>zmin-qsi)and(z<zmin+qsi))):
            corner_b = p.nodes[k:k+1]
            p.Set(nodes=corner_b, name='b_corner')
        if (((x>xmin-qsi)and(x<xmin+qsi))and\
            ((y>ymin-qsi)and(y<ymin+qsi))and\
            ((z>zmax-qsi)and(z<zmax+qsi))):
            corner_f = p.nodes[k:k+1]
            p.Set(nodes=corner_f, name='f_corner')
        if (((x>xmax-qsi)and(x<xmax+qsi))and\
            ((y>ymin-qsi)and(y<ymin+qsi))and\
            ((z>zmin-qsi)and(z<zmin+qsi))):
            corner_d = p.nodes[k:k+1]
            p.Set(nodes=corner_d, name='d_corner')
        if (((x>xmax-qsi)and(x<xmax+qsi))and\
            ((y>ymin-qsi)and(y<ymin+qsi))and\
            ((z>zmax-qsi)and(z<zmax+qsi))):
            corner_h = p.nodes[k:k+1]
            p.Set(nodes=corner_h, name='h_corner')
        if (((x>xmax-qsi)and(x<xmax+qsi))and\
            ((y>ymin-qsi)and(y<ymin+qsi))and\
            ((z>zmin-qsi)and(z<zmin+qsi))):
            corner_c = p.nodes[k:k+1]
            p.Set(nodes=corner_c, name='c_corner')
        if (((x>xmax-qsi)and(x<xmax+qsi))and\
            ((y>ymin-qsi)and(y<ymin+qsi))and\
            ((z>zmax-qsi)and(z<zmax+qsi))):
            corner_g = p.nodes[k:k+1]
            p.Set(nodes=corner_g, name='g_corner')

```

```

mdb.models['Cell'].Equation(name='A_G_eq_1', terms=((-1.0, 'Cell-1.a_corner', 1),
(1.0, 'Cell-1.g_corner', 1), (-1.0, 'Set-rp1', 1), (-1.0, 'Set-rp2', 1),
(-1.0, 'Set-rp2', 2)))
mdb.models['Cell'].Equation(name='A_G_eq_2', terms=((-1.0, 'Cell-1.a_corner', 2),
(1.0, 'Cell-1.g_corner', 2), (-1.0, 'Set-rp1', 2), (-1.0, 'Set-rp2', 1),
(-1.0, 'Set-rp2', 3)))
mdb.models['Cell'].Equation(name='A_G_eq_3', terms=((-1.0, 'Cell-1.a_corner', 3),
(1.0, 'Cell-1.g_corner', 3), (-1.0, 'Set-rp1', 3), (-1.0, 'Set-rp2', 2),
(-1.0, 'Set-rp2', 3)))
mdb.models['Cell'].Equation(name='B_H_eq_1', terms=((-1.0, 'Cell-1.b_corner', 1),
(1.0, 'Cell-1.h_corner', 1), (-1.0, 'Set-rp1', 1), (1.0, 'Set-rp2', 1),
(-1.0, 'Set-rp2', 2)))
mdb.models['Cell'].Equation(name='B_H_eq_2', terms=((-1.0, 'Cell-1.b_corner', 2),
(1.0, 'Cell-1.h_corner', 2), (1.0, 'Set-rp1', 2), (-1.0, 'Set-rp2', 1),
(-1.0, 'Set-rp2', 3)))
mdb.models['Cell'].Equation(name='B_H_eq_3', terms=((-1.0, 'Cell-1.b_corner', 3),
(1.0, 'Cell-1.h_corner', 3), (-1.0, 'Set-rp1', 3), (-1.0, 'Set-rp2', 2),
(1.0, 'Set-rp2', 3)))
mdb.models['Cell'].Equation(name='F_D_eq_1', terms=((1.0, 'Cell-1.f_corner', 1),
(-1.0, 'Cell-1.d_corner', 1), (1.0, 'Set-rp1', 1), (-1.0, 'Set-rp2', 1),
(-1.0, 'Set-rp2', 2)))
mdb.models['Cell'].Equation(name='F_D_eq_2', terms=((1.0, 'Cell-1.f_corner', 2),
(-1.0, 'Cell-1.d_corner', 2), (-1.0, 'Set-rp1', 2), (1.0, 'Set-rp2', 1),
(-1.0, 'Set-rp2', 3)))
mdb.models['Cell'].Equation(name='F_D_eq_3', terms=((1.0, 'Cell-1.f_corner', 3),
(-1.0, 'Cell-1.d_corner', 3), (-1.0, 'Set-rp1', 3), (1.0, 'Set-rp2', 2),
(-1.0, 'Set-rp2', 3)))
mdb.models['Cell'].Equation(name='E_C_eq_1', terms=((-1.0, 'Cell-1.e_corner', 1),
(1.0, 'Cell-1.c_corner', 1), (-1.0, 'Set-rp1', 1), (-1.0, 'Set-rp2', 1),
(1.0, 'Set-rp2', 2)))
mdb.models['Cell'].Equation(name='E_C_eq_2', terms=((-1.0, 'Cell-1.e_corner', 2),
(1.0, 'Cell-1.c_corner', 2), (-1.0, 'Set-rp1', 2), (-1.0, 'Set-rp2', 1),
(1.0, 'Set-rp2', 3)))
mdb.models['Cell'].Equation(name='E_C_eq_3', terms=((-1.0, 'Cell-1.e_corner', 3),
(1.0, 'Cell-1.c_corner', 3), (1.0, 'Set-rp1', 3), (-1.0, 'Set-rp2', 2),
(-1.0, 'Set-rp2', 3)))
mdb.saveAs(pathName=arquivo)
##### Arestas
ab = []
hg = []
for k in range(nnodes):
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    if ((x > xmin-qs) and (x < xmin+qs)) and\
        ((z > zmin-qs) and (z < zmin+qs)) and\
        ((y > ymin+qs) and (y < ymax-qs)):
        ab.append(k)
    elif ((x > xmax-qs) and (x < xmax+qs)) and\
        ((z < zmax+qs) and (z > zmax-qs)) and\
        ((y > ymin+qs) and (y < ymax-qs)):
        hg.append(k)
for k in ab:
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    for j in hg:
        x1, y1, z1 = p.nodes[j].coordinates[0], p.nodes[j].coordinates[1],\
            p.nodes[j].coordinates[2]
        if ((y1 > y-qs) and (y1 < y+qs)):
            AB = p.nodes[k:k+1]
            HG = p.nodes[j:j+1]
            name_set1 = 'set_node_' + str(k)
            name_set2 = 'set_node_' + str(j)
            constraint1 = 'Cell-1.' + name_set1
            constraint2 = 'Cell-1.' + name_set2
            eq_name1 = 'AB_HG_eq_' + str(k) + '_' + str(j) + '_1'
            eq_name2 = 'AB_HG_eq_' + str(k) + '_' + str(j) + '_2'
            eq_name3 = 'AB_HG_eq_' + str(k) + '_' + str(j) + '_3'
            p.Set(nodes=AB, name=name_set1)
            p.Set(nodes=HG, name=name_set2)
            mdb.models['Cell'].Equation(name=eq_name1, terms=((-1.0, constraint1, 1),
(1.0, constraint2, 1), (-1.0, 'Set-rp1', 1), (-1.0, 'Set-rp2', 2)))
            mdb.models['Cell'].Equation(name=eq_name2, terms=((-1.0, constraint1, 2),
(1.0, constraint2, 2), (-1.0, 'Set-rp2', 1), (-1.0, 'Set-rp2', 3)))
            mdb.models['Cell'].Equation(name=eq_name3, terms=((-1.0, constraint1, 3),
(1.0, constraint2, 3), (-1.0, 'Set-rp1', 3), (-1.0, 'Set-rp2', 2)))

```

```

        hg.remove(j)
        break

dc = []
ef = []
for k in range(nnodes):
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    if ((x > xmin-qs) and (x < xmin+qs)) and\
        ((z > zmax-qs) and (z < zmax+qs)) and\
        ((y > ymin+qs) and (y < ymax-qs)):
        ef.append(k)
    elif ((x > xmax-qs) and (x < xmax+qs)) and\
        ((z < zmin+qs) and (z > zmin-qs)) and\
        ((y > ymin+qs) and (y < ymax-qs)):
        dc.append(k)
for k in dc:
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    for j in ef:
        x1, y1, z1 = p.nodes[j].coordinates[0], p.nodes[j].coordinates[1],\
            p.nodes[j].coordinates[2]
        if ((y1 > y-qs) and (y1 < y+qs)):
            DC = p.nodes[k:k+1]
            EF = p.nodes[j:j+1]
            name_set1 = 'set_node_' + str(k)
            name_set2 = 'set_node_' + str(j)
            constraint1 = 'Cell-1.' + name_set1
            constraint2 = 'Cell-1.' + name_set2
            eq_name1 = 'DC_EF_eq_' + str(k) + '_' + str(j) + '_1'
            eq_name2 = 'DC_EF_eq_' + str(k) + '_' + str(j) + '_2'
            eq_name3 = 'DC_EF_eq_' + str(k) + '_' + str(j) + '_3'
            p.Set(nodes=DC, name=name_set1)
            p.Set(nodes=EF, name=name_set2)
            mdb.models['Cell'].Equation(name=eq_name1, terms=((1.0, constraint1, 1),
                (-1.0, constraint2, 1), (-1.0, 'Set-rp1', 1), (1.0, 'Set-rp2', 2)))
            mdb.models['Cell'].Equation(name=eq_name2, terms=((1.0, constraint1, 2),
                (-1.0, constraint2, 2), (-1.0, 'Set-rp2', 1), (1.0, 'Set-rp2', 3)))
            mdb.models['Cell'].Equation(name=eq_name3, terms=((1.0, constraint1, 3),
                (-1.0, constraint2, 3), (1.0, 'Set-rp1', 3), (-1.0, 'Set-rp2', 2)))
            ef.remove(j)
            break

eh = []
bc = []
for k in range(nnodes):
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    if ((y > ymin-qs) and (y < ymin+qs)) and\
        ((z > zmax-qs) and (z < zmax+qs)) and\
        ((x > xmin+qs) and (x < xmax-qs)):
        eh.append(k)
    elif ((y > ymax-qs) and (y < ymax+qs)) and\
        ((z < zmin+qs) and (z > zmin-qs)) and\
        ((x > xmin+qs) and (x < xmax-qs)):
        bc.append(k)
for k in bc:
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    for j in eh:
        x1, y1, z1 = p.nodes[j].coordinates[0], p.nodes[j].coordinates[1],\
            p.nodes[j].coordinates[2]
        if ((x1 > x-qs) and (x1 < x+qs)):
            BC = p.nodes[k:k+1]
            EH = p.nodes[j:j+1]
            name_set1 = 'set_node_' + str(k)
            name_set2 = 'set_node_' + str(j)
            constraint1 = 'Cell-1.' + name_set1
            constraint2 = 'Cell-1.' + name_set2
            eq_name1 = 'BC_EH_eq_' + str(k) + '_' + str(j) + '_1'
            eq_name2 = 'BC_EH_eq_' + str(k) + '_' + str(j) + '_2'
            eq_name3 = 'BC_EH_eq_' + str(k) + '_' + str(j) + '_3'
            p.Set(nodes=BC, name=name_set1)
            p.Set(nodes=EH, name=name_set2)
            mdb.models['Cell'].Equation(name=eq_name1, terms=((1.0, constraint1, 1),
                (-1.0, constraint2, 1), (-1.0, 'Set-rp2', 1), (1.0, 'Set-rp2', 2)))
            mdb.models['Cell'].Equation(name=eq_name2, terms=((1.0, constraint1, 2),

```

```

        (-1.0, constraint2, 2), (-1.0, 'Set-rp1', 2), (1.0, 'Set-rp2', 3)))
mdb.models['Cell'].Equation(name=eq_name3, terms=((1.0, constraint1, 3),
        (-1.0, constraint2, 3), (1.0, 'Set-rp1', 3), (-1.0, 'Set-rp2', 3)))
eh.remove(j)
break

fg = []
ad = []
for k in range(nnodes):
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    if ((y > ymax-qs) and (y < ymax+qs)) and\
        ((z > zmax-qs) and (z < zmax+qs)) and\
        ((x > xmin+qs) and (x < xmax-qs)):
        fg.append(k)
    elif ((y > ymin-qs) and (y < ymin+qs)) and\
        ((z < zmin+qs) and (z > zmin-qs)) and\
        ((x > xmin+qs) and (x < xmax-qs)):
        ad.append(k)
for k in fg:
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    for j in ad:
        x1, y1, z1 = p.nodes[j].coordinates[0], p.nodes[j].coordinates[1],\
            p.nodes[j].coordinates[2]
        if ((x1 > x-qs) and (x1 < x+qs)):
            FG = p.nodes[k:k+1]
            AD = p.nodes[j:j+1]
            name_set1 = 'set_node_' + str(k)
            name_set2 = 'set_node_' + str(j)
            constraint1 = 'Cell-1.' + name_set1
            constraint2 = 'Cell-1.' + name_set2
            eq_name1 = 'FG_AD_eq_' + str(k) + '_' + str(j) + '_1'
            eq_name2 = 'FG_AD_eq_' + str(k) + '_' + str(j) + '_2'
            eq_name3 = 'FG_AD_eq_' + str(k) + '_' + str(j) + '_3'
            p.Set(nodes=FG, name=name_set1)
            p.Set(nodes=AD, name=name_set2)
            mdb.models['Cell'].Equation(name=eq_name1, terms=((1.0, constraint1, 1),
                (-1.0, constraint2, 1), (-1.0, 'Set-rp2', 1), (-1.0, 'Set-rp2', 2)))
            mdb.models['Cell'].Equation(name=eq_name2, terms=((1.0, constraint1, 2),
                (-1.0, constraint2, 2), (-1.0, 'Set-rp1', 2), (-1.0, 'Set-rp2', 3)))
            mdb.models['Cell'].Equation(name=eq_name3, terms=((1.0, constraint1, 3),
                (-1.0, constraint2, 3), (-1.0, 'Set-rp1', 3), (-1.0, 'Set-rp2', 3)))
            ad.remove(j)
            break

ae = []
cg = []
for k in range(nnodes):
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    if ((x > xmin-qs) and (x < xmin+qs)) and\
        ((y > ymin-qs) and (y < ymin+qs)) and\
        ((z > zmin+qs) and (z < zmax-qs)):
        ae.append(k)
    elif ((x > xmax-qs) and (x < xmax+qs)) and\
        ((y < ymax+qs) and (y > ymax-qs)) and\
        ((z > zmin+qs) and (z < zmax-qs)):
        cg.append(k)
for k in cg:
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    for j in ae:
        x1, y1, z1 = p.nodes[j].coordinates[0], p.nodes[j].coordinates[1],\
            p.nodes[j].coordinates[2]
        if ((z1 > z-qs) and (z1 < z+qs)):
            CG = p.nodes[k:k+1]
            AE = p.nodes[j:j+1]
            name_set1 = 'set_node_' + str(k)
            name_set2 = 'set_node_' + str(j)
            constraint1 = 'Cell-1.' + name_set1
            constraint2 = 'Cell-1.' + name_set2
            eq_name1 = 'CG_AE_eq_' + str(k) + '_' + str(j) + '_1'
            eq_name2 = 'CG_AE_eq_' + str(k) + '_' + str(j) + '_2'
            eq_name3 = 'CG_AE_eq_' + str(k) + '_' + str(j) + '_3'
            p.Set(nodes=CG, name=name_set1)
            p.Set(nodes=AE, name=name_set2)

```

```

        mdb.models['Cell'].Equation(name=eq_name1, terms=((1.0, constraint1, 1),
            (-1.0, constraint2, 1), (-1.0, 'Set-rp1', 1), (-1.0, 'Set-rp2', 1)))
        mdb.models['Cell'].Equation(name=eq_name2, terms=((1.0, constraint1, 2),
            (-1.0, constraint2, 2), (-1.0, 'Set-rp1', 2), (-1.0, 'Set-rp2', 1)))
        mdb.models['Cell'].Equation(name=eq_name3, terms=((1.0, constraint1, 3),
            (-1.0, constraint2, 3), (-1.0, 'Set-rp2', 2), (-1.0, 'Set-rp2', 3)))
        ae.remove(j)
        break

bf = []
dh = []
for k in range(nnodes):
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    if ((x > xmin-qs) and (x < xmin+qs)) and\
        ((y > ymax-qs) and (y < ymax+qs)) and\
        ((z > zmin+qs) and (z < zmax-qs)):
        bf.append(k)
    elif ((x > xmax-qs) and (x < xmax+qs)) and\
        ((y < ymin+qs) and (y > ymin-qs)) and\
        ((z > zmin+qs) and (z < zmax-qs)):
        dh.append(k)
for k in dh:
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    for j in bf:
        x1, y1, z1 = p.nodes[j].coordinates[0], p.nodes[j].coordinates[1],\
            p.nodes[j].coordinates[2]
        if ((z1 > z-qs) and (z1 < z+qs)):
            DH = p.nodes[k:k+1]
            BF = p.nodes[j:j+1]
            name_set1 = 'set_node_' + str(k)
            name_set2 = 'set_node_' + str(j)
            constraint1 = 'Cell-1.' + name_set1
            constraint2 = 'Cell-1.' + name_set2
            eq_name1 = 'DH_BF_eq_' + str(k) + '_' + str(j) + '_1'
            eq_name2 = 'DH_BF_eq_' + str(k) + '_' + str(j) + '_2'
            eq_name3 = 'DH_BF_eq_' + str(k) + '_' + str(j) + '_3'
            p.Set(nodes=DH, name=name_set1)
            p.Set(nodes=BF, name=name_set2)
            mdb.models['Cell'].Equation(name=eq_name1, terms=((1.0, constraint1, 1),
                (-1.0, constraint2, 1), (-1.0, 'Set-rp1', 1), (1.0, 'Set-rp2', 1)))
            mdb.models['Cell'].Equation(name=eq_name2, terms=((1.0, constraint1, 2),
                (-1.0, constraint2, 2), (1.0, 'Set-rp1', 2), (-1.0, 'Set-rp2', 1)))
            mdb.models['Cell'].Equation(name=eq_name3, terms=((1.0, constraint1, 3),
                (-1.0, constraint2, 3), (-1.0, 'Set-rp2', 2), (1.0, 'Set-rp2', 3)))
            bf.remove(j)
            break

mdb.saveAs(pathName=arquivo)
#####B Faces
abfe = []
dcgh = []
for k in range(nnodes):
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    if ((x > xmin-qs) and (x < xmin+qs)) and\
        ((y > ymin+qs) and (y < ymax-qs)) and\
        ((z > zmin+qs) and (z < zmax-qs)):
        abfe.append(k)
    elif ((x > xmax-qs) and (x < xmax+qs)) and\
        ((y > ymin+qs) and (y < ymax-qs)) and\
        ((z > zmin+qs) and (z < zmax-qs)):
        dcgh.append(k)
for k in abfe:
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    for j in dcgh:
        x1, y1, z1 = p.nodes[j].coordinates[0], p.nodes[j].coordinates[1],\
            p.nodes[j].coordinates[2]
        if ((y1 > y-qs) and (y1 < y+qs)) and ((z1 > z-qs) and (z1 < z+qs)):
            ABFE = p.nodes[k:k+1]
            DCGH = p.nodes[j:j+1]
            name_set1 = 'set_node_' + str(k)
            name_set2 = 'set_node_' + str(j)
            constraint1 = 'Cell-1.' + name_set1
            constraint2 = 'Cell-1.' + name_set2
            eq_name1 = 'ABFE_DCGH_eq_' + str(k) + '_' + str(j) + '_1'

```

```

eq_name2 = 'ABFE_DCGH_eq_' + str(k) + '_' + str(j) + '_2'
eq_name3 = 'ABFE_DCGH_eq_' + str(k) + '_' + str(j) + '_3'
p.Set(nodes=ABFE, name=name_set1)
p.Set(nodes=DCGH, name=name_set2)
mdb.models['Cell'].Equation(name=eq_name1, terms=((-1.0, constraint1, 1),
(1.0, constraint2, 1), (-1.0, 'Set-rp1', 1)))
mdb.models['Cell'].Equation(name=eq_name2, terms=((-1.0, constraint1, 2),
(1.0, constraint2, 2), (-1.0, 'Set-rp2', 1)))
mdb.models['Cell'].Equation(name=eq_name3, terms=((-1.0, constraint1, 3),
(1.0, constraint2, 3), (-1.0, 'Set-rp2', 2)))
dcgh.remove(j)
break

aehd = []
bfgc = []
for k in range(nnodes):
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    if ((y > ymin-qs) and (y < ymin+qs)) and\
        ((x > xmin+qs) and (x < xmax-qs)) and\
        ((z > zmin+qs) and (z < zmax-qs)):
        aehd.append(k)
    elif ((y > ymax-qs) and (y < ymax+qs)) and\
        ((x > xmin+qs) and (x < xmax-qs)) and\
        ((z > zmin+qs) and (z < zmax-qs)):
        bfgc.append(k)
for k in aehd:
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    for j in bfgc:
        x1, y1, z1 = p.nodes[j].coordinates[0], p.nodes[j].coordinates[1],\
            p.nodes[j].coordinates[2]
        if ((x1 > x-qs) and (x1 < x+qs)):
            if ((z1 > z-qs) and (z1 < z+qs)):
                AEHD = p.nodes[k:k+1]
                BFGC = p.nodes[j:j+1]
                name_set1 = 'set_node_' + str(k)
                name_set2 = 'set_node_' + str(j)
                constraint1 = 'Cell-1.' + name_set1
                constraint2 = 'Cell-1.' + name_set2
                eq_name1 = 'AEHD-BFGC_eq_' + str(k) + '_' + str(j) + '_1'
                eq_name2 = 'AEHD-BFGC_eq_' + str(k) + '_' + str(j) + '_2'
                eq_name3 = 'AEHD-BFGC_eq_' + str(k) + '_' + str(j) + '_3'
                p.Set(nodes=AEHD, name=name_set1)
                p.Set(nodes=BFGC, name=name_set2)
                mdb.models['Cell'].Equation(name=eq_name1, terms=((-1.0, constraint1, 1),
(1.0, constraint2, 1), (-1.0, 'Set-rp2', 1)))
                mdb.models['Cell'].Equation(name=eq_name2, terms=((-1.0, constraint1, 2),
(1.0, constraint2, 2), (-1.0, 'Set-rp1', 2)))
                mdb.models['Cell'].Equation(name=eq_name3, terms=((-1.0, constraint1, 3),
(1.0, constraint2, 3), (-1.0, 'Set-rp2', 3)))
                bfgc.remove(j)
            break

abcd = []
efgh = []
for k in range(nnodes):
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    if ((z > zmin-qs) and (z < zmin+qs)) and\
        ((x > xmin+qs) and (x < xmax-qs)) and\
        ((y > ymin+qs) and (y < ymax-qs)):
        abcd.append(k)
    elif ((z > zmax-qs) and (z < zmax+qs)) and\
        ((x > xmin+qs) and (x < xmax-qs)) and\
        ((y > ymin+qs) and (y < ymax-qs)):
        efgh.append(k)
for k in abcd:
    x, y, z = p.nodes[k].coordinates[0], p.nodes[k].coordinates[1], p.nodes[k].coordinates[2]
    for j in efgh:
        x1, y1, z1 = p.nodes[j].coordinates[0], p.nodes[j].coordinates[1],\
            p.nodes[j].coordinates[2]
        if ((x1 > x-qs) and (x1 < x+qs)):
            if ((y1 > y-qs) and (y1 < y+qs)):
                ABCD = p.nodes[k:k+1]

```

```

EFGH = p.nodes[j:j+1]
name_set1 = 'set_node_' + str(k)
name_set2 = 'set_node_' + str(j)
constraint1 = 'Cell-1.' + name_set1
constraint2 = 'Cell-1.' + name_set2
eq_name1 = 'ABCD_EFGH_eq_' + str(k) + '_' + str(j) + '_1'
eq_name2 = 'ABCD_EFGH_eq_' + str(k) + '_' + str(j) + '_2'
eq_name3 = 'ABCD_EFGH_eq_' + str(k) + '_' + str(j) + '_3'
p.Set(nodes=ABCD, name=name_set1)
p.Set(nodes=EFGH, name=name_set2)
mdb.models['Cell'].Equation(name=eq_name1, terms=((-1.0, constraint1, 1),
(1.0, constraint2, 1), (-1.0, 'Set-rp2', 2)))
mdb.models['Cell'].Equation(name=eq_name2, terms=((-1.0, constraint1, 2),
(1.0, constraint2, 2), (-1.0, 'Set-rp2', 3)))
mdb.models['Cell'].Equation(name=eq_name3, terms=((-1.0, constraint1, 3),
(1.0, constraint2, 3), (-1.0, 'Set-rp1', 3)))
efgh.remove(j)
break

mdb.saveAs(pathName=arquivo)

```

A.4 Analysis

```

from abaqus import *
from abaqusConstants import *
from caeModules import *
from driverUtils import executeOnCaeStartup
#####
#####In case of using this code snippet directly, pay attention to the temperature field,
#####as it might be applied to a set of nodes of different size than the actual model
def analise(arq, matrix, resf):
    executeOnCaeStartup()
    if matrix == 'G1':
        aque = 595
    elif matrix == 'G2':
        aque = 605
    elif matrix == 'G3':
        aque = 470

    path = 'C:/Users/duhju/Desktop/abaqus/modelos/' + arq + '.cae'
    openMdb(pathName=path)
    ##: The model database "C:\Users\duhju\Desktop\abaqus\modelos\CFG-30.cae" has been opened.
    p = mdb.models['Cell'].parts['Cell']
    a = mdb.models['Cell'].rootAssembly
    a.regenerate()
    a = mdb.models['Cell'].rootAssembly

    mdb.models['Cell'].StaticStep(name='Step-2', previous='Step-1')
    mdb.models['Cell'].StaticStep(name='Step-3', previous='Step-2')

    a = mdb.models['Cell'].rootAssembly
    n1 = a.instances['Cell-1'].nodes
    region = a.Set(nodes=n1, name='Set-3')

    ###Campo de aquecimento do modelo
    mdb.models['Cell'].Temperature(name='Predefined Field-1',
        createStepName='Initial', region=region, distributionType=UNIFORM,
        crossSectionDistribution=CONSTANT_THROUGH_THICKNESS, magnitudes=(0.0, ))
    mdb.models['Cell'].predefinedFields['Predefined Field-1'].setValuesInStep(
        stepName='Step-1', magnitudes=(float(aque), ))
    ###Campo de resfriamento do modelo
    a = mdb.models['Cell'].rootAssembly
    region = a.sets['Set-3']
    mdb.models['Cell'].Temperature(name='Predefined Field-2',
        createStepName='Step-1', region=region, distributionType=UNIFORM,
        crossSectionDistribution=CONSTANT_THROUGH_THICKNESS, magnitudes=(float(aque), ))
    mdb.models['Cell'].predefinedFields['Predefined Field-2'].setValuesInStep(
        stepName='Step-2', magnitudes=(float(resf), ))
    ###Condicao de deformacao do modelo

```

```

if resf == aque:
    mdb.models['Cell'].predefinedFields['Predefined Field-2'].suppress()

epso = 1.0e-4
cargas = ['xx', 'yy', 'zz', 'xy', 'xz', 'yz']
for carga in cargas:

    if carga=='xx':
        mdb.models['Cell'].boundaryConditions['BC-rp1'].setValuesInStep(
            stepName='Step-3', u1=epso, u2=0.000, u3=0.000)
    elif carga=='yy':
        mdb.models['Cell'].boundaryConditions['BC-rp1'].setValuesInStep(
            stepName='Step-3', u1=0.000, u2=epso, u3=0.000)
    elif carga=='zz':
        mdb.models['Cell'].boundaryConditions['BC-rp1'].setValuesInStep(
            stepName='Step-3', u1=0.000, u2=0.000, u3=epso)
    elif carga=='xy':
        mdb.models['Cell'].boundaryConditions['BC-rp1'].setValuesInStep(
            stepName='Step-3', u1=0.000, u2=0.000, u3=0.000)
        mdb.models['Cell'].boundaryConditions['BC-rp2'].setValuesInStep(
            stepName='Step-3', u1=2*epso, u2=0.000, u3=0.000)
    elif carga=='xz':
        mdb.models['Cell'].boundaryConditions['BC-rp2'].setValuesInStep(
            stepName='Step-3', u1=0.000, u2=2*epso, u3=0.000)
    elif carga=='yz':
        mdb.models['Cell'].boundaryConditions['BC-rp2'].setValuesInStep(
            stepName='Step-3', u1=0.000, u2=0.000, u3=2*epso)

job = 'Job-'+carga
### Analise MEF
mdb.Job(name=job, model='Cell', description='', type=ANALYSIS, atTime=None,
        waitMinutes=0, waitHours=0, queue=None, memory=90, memoryUnits=PERCENTAGE,
        getMemoryFromAnalysis=True, explicitPrecision=SINGLE,
        nodalOutputPrecision=SINGLE, echoPrint=OFF, modelPrint=OFF,
        contactPrint=OFF, historyPrint=OFF, userSubroutine='', scratch='',
        resultsFormat=ODB, multiprocessingMode=DEFAULT, numCpus=1, numGPUs=0)
mdb.jobs[job].submit(consistencyChecking=OFF)

```

1.5 Report

```

from abaqus import *
from abaqusConstants import *
from caeModules import *
from driverUtils import executeOnCaeStartup

def relat(matrix, resf):

    cargas = ['xx', 'yy', 'zz', 'xy', 'xz', 'yz']
    for carga in cargas:
        job = 'Job-'+carga
        job_path = 'C:/Temp/'+job+'.odb'
        reporto = matrix+'_'+carga+'_'+str(resf)+'.rpt'
        session.Viewport(name='Viewport: 1', origin=(0.0, 0.0), width=195.443634033203,
            height=90.3294296264648)
        session.viewports['Viewport: 1'].makeCurrent()
        session.viewports['Viewport: 1'].maximize()
        executeOnCaeStartup()
        session.viewports['Viewport: 1'].partDisplay.geometryOptions.setValues(
            referenceRepresentation=ON)
        o1 = session.openOdb(name=job_path)
        session.viewports['Viewport: 1'].setValues(displayedObject=o1)
        odb = session.odbs[job_path]
        nf = NumberFormat(numDigits=8, precision=0, format=ENGINEERING)
        session.fieldReportOptions.setValues(numberFormat=nf)
        session.writeFieldReport(fileName=reporto, append=ON,
            sortItem='Element Label', odb=odb, step=2, frame=1,
            outputPosition=INTEGRATION_POINT, variable=((('E', INTEGRATION_POINT, ((
                COMPONENT, 'E11'), (COMPONENT, 'E22'), (COMPONENT, 'E33'), (COMPONENT,
                'E12'), (COMPONENT, 'E13'), (COMPONENT, 'E23'), )), ('S',
                INTEGRATION_POINT, ((COMPONENT, 'S11'), (COMPONENT, 'S22'), (COMPONENT,
                'S33'), (COMPONENT, 'S12'), (COMPONENT, 'S13'), (COMPONENT, 'S23'), )), ))

```

