INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA MECÂNICA




Dimensionamento Mecânico de Reforço "Favo de Abelha" de Componentes pelo Método de Elementos Finitos, para a Indústria Automóvel

GABRIEL FILIPE LUCAS RIBEIRO RAMOS
julho de 2022

POLITÉCNICO
DO PORTO

INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA MECÂNICA




Dimensionamento Mecânico de Reforço "Favo de Abelha" de Componentes pelo Método de Elementos Finitos para a Indústria Automóvel

GABRIEL FILIPE LUCAS RIBEIRO RAMOS
Maio de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

DIMENSIONAMENTO MECÂNICO DE REFORÇO "FAVO DE ABELHA" DE COMPONENTES PELO MÉTODO DE ELEMENTOS FINITOS PARA A INDÚSTRIA AUTOMÓVEL

Gabriel Filipe Lucas Ribeiro Ramos

1170763

2021/2022

Instituto Superior de Engenharia do Porto

Departamento de Engenharia Mecânica

POLITÉCNICO DO PORTO

Instituto Superior de Engenharia do Porto

DIMENSIONAMENTO MECÂNICO DE REFORÇO "FAVO DE ABELHA" DE COMPONENTES PELO MÉTODO DE ELEMENTOS FINITOS PARA A INDÚSTRIA AUTOMÓVEL

Gabriel Filipe Lucas Ribeiro Ramos

1170763

Dissertação apresentada ao Instituto Superior de Engenharia do Porto para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob a orientação do Professor Jorge Fonseca Justo e o Engenheiro Fernando Vilaça, na Simoldes Plásticos®.

2021/2022

Instituto Superior de Engenharia do Porto

Departamento de Engenharia Mecânica

POLITÉCNICO DO PORTO

JÚRI

Presidente

Professor Doutor Arnaldo Guedes Pinto

Professor Adjunto, Instituto Superior de Engenharia do Porto

Orientador

Professor Doutor Jorge Fonseca Justo

Professor Adjunto, Instituto Superior de Engenharia do Porto

Arguente

Professor Doutor Marcelo Francisco de Sousa Ferreira de Moura

Professor Associado, Faculdade de Engenharia da Universidade do Porto

AGRADECIMENTOS

Quero agradecer ao meu orientador, Doutor Jorge Fonseca Justo, pela ajuda prestada durante a realização da dissertação. Ao meu orientador na Simoldes Plásticos®, o Engenheiro Fernando Vilaça por me ter ensinado a trabalhar com o software e a programar em Python. E por último à minha família.

PALAVRAS-CHAVE

Materiais de Cariz polimérico, Simoldes Plásticos®, Scripts, Python, Simulia Abaqus®, Simulações pelo Método de Elementos Finitos (MEF), Flexão, Cátia V5®, Simplificação de geometria, Superfícies Médias, Pré-processamento, *Graphical User Interface (GUI)*.

RESUMO

Na fabricação de peças de materiais de cariz polimérico, uma das metodologias empregues para melhorar as propriedades mecânicas de uma determinada zona, consiste em criar secções retas e finas de material, denominadas de frisos, que conferem uma maior rigidez na área aplicada. Isto é muito útil quando apenas uma porção pequena da peça se encontra a condições de trabalho mais exigentes, pois funciona como uma medida de reforço geométrico, a um custo relativamente reduzido. Ora, de maneira a poder-se reduzir ainda mais os custos de produção, é do interesse das empresas, otimizar o design destas geometrias, de forma a ser gasto a menor quantidade de matéria-prima possível, enquanto se mantêm as propriedades desejadas pelo cliente.

A presente dissertação tem como objetivo a otimização do design destes mesmos frisos, em certos componentes de interior de automóveis da empresa Simoldes Plásticos®. Para se alcançar este objetivo desenvolveram-se scripts em Python que permitem fazer análises sequenciais e automáticas no software Simulia Abaqus®, através de simulações pelo Método de Elementos Finitos (MEF), onde as respetivas peças, com diferentes geometrias de reforço, foram submetidas a um ensaio de flexão, e os dados das respostas dos sistemas foram processados automaticamente, de maneira a determinar-se a combinação de características que melhor se adequa ao componente.

Para o desenvolvimento do programa, as peças foram primeiro modeladas no software de CAD Cátia V5®, tendo-se depois recorrido à simplificação das mesmas, através da supressão de atributos irrelevantes para a simulação, e a implementação de superfícies médias. As peças simplificadas foram depois transferidas para o Simulia Abaqus® onde se fez uma simulação, sem que tenham sido aplicados frisos, de maneira a utilizar o código resultante, como base para a construção do modelo do script final. Com a base do programa feita, procedeu-se com a formulação do algoritmo responsável pelo desenho, e parametrização automática dos frisos, bem como as funções relativas ao pré-processamento, pós-processamento, desenvolvimento de um *Graphical User Interface* (GUI), envio de relatórios por correio eletrónico e alteração das prioridades dos programas presentes no sistema operativo.

Com o software desenvolvido, correram-se as simulações, tendo-se obtidos os dados sobre a geometria dos frisos que permite satisfazer os requisitos do cliente, utilizando-se a menor quantidade de material possível.

KEYWORDS

Polymeric parts, Simoldes Plásticos®, Scripts, Python, Simulia Abaqus®, Finite Element Method (FEM), Bending Test, Catia V5, Suppression of irrelevant attributes, Mid-plane, Post-processing, *Graphical User Interface (GUI).*

ABSTRACT

In the manufacturing of polymeric parts, one of the methodologies used to improve the mechanical properties of a given area is to create straight and thin sections of material, called ribs, which provide greater rigidity to the applied area. This is very useful when only a small portion of the part is under more demanding working conditions, as it works as a geometrical reinforcement measure, at a relatively low cost. However, for companies whose production focus is on this type of components, it makes sense that they want to spend as little material as possible, while giving them the properties desired by the customer.

This dissertation aims to optimize the design of these same ribs, in certain car interior components from Simoldes Plásticos®. To achieve this goal, Python scripts were developed that allow automatic sequential analysis in the Abaqus® software, using the Finite Element Method (FEM). Where different combinations of material and geometric properties of the ribs were tested, and the response of the components to a bending test was observed. With the data obtained, the best possible geometry was then selected.

For the development of the software, the parts were first modelled in the CAD software Catia V5, having then resorted to their geometrical simplification, through the suppression of irrelevant attributes for the simulation, and the implementation of mid-planes. The simplified parts were then taken to Abaqus® where a simulation was carried out, without ribs having been applied, in order to use the resulting code as a constructor of the part in the final script. With the constructer completed, next came the coding of the function responsible for the parameterization of the friezes geometry, as well as the functions related to post-processing, development of a Graphical User Interface (GUI), sending reports via email and changing program priorities in the operating system.

With the developed software, the simulations were run over a weekend, having obtained data on the best geometry for the friezes, and implemented these in the final piece.

LISTA DE SÍMBOLOS E ABREVIATURAS

Lista de Abreviaturas

ACEA	European Automobile Manifacturers Association
AFIA	Associação de Fabricantes para a Indústria Automóvel
AI	Artificial Intelligence
API	Aplication Programing Interface
CAE	Complete Abaqus Environment
CFD	Computational Fluid Dynamics
GL	Graus de Liberdade
GUI	Graphical User Interface
IC	Integrated Circuit
MD	Métodos Diretos
MDB	Model Data Base
MEF	Método de Elementos Finitos
ML	Machine Learning
MRP	Método dos Resíduos Ponderados
MV	Métodos Variacionais
ODB	Output Data Base
OICA	International Organization of Motor Vehicle Manufacturers
OS	Operating System
PA	Poliamidas

PC	Policarbonatos
PCB	Printed Circuit Board
PDV	Princípio dos Deslocamentos Virtuais
PIB	Produto Interno Bruto
PNG	Portable Graphics Format
PP	Polipropileno
PS	Poliestireno
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
TEPM	Teorema da Energia Potencial Mínima

Lista de Unidades

%	Percentagem
^{o}	Graus (Ângulo)
Flops	Flops
m	Metro
N	Newton
^{o}C	Graus Celsius
P	Pascal
s	Segundos

Lista de Símbolos

f	Forças nodais
$\emptyset_i$	Variáveis de campo
D_m	Matriz de deformação
E_y	Modulo de Elasticidade
H_p	Energia potencial de trabalho
N_i	Função de interpolação
Q_{cr}^{1D}	Carga critica num modelo unidimensional
Q_{cr}^{FEM}	Carga critica num modelo de membrana
V_i	Volume de um subdomínio de um corpo
$a_e\ e\ A_e$	Vetor de variáveis nodais
$f_e\ e\ F_e$	Vetor de forças nodais
k_e	Matriz de rigidez de um elemento
u_e	Campo de deslocamentos do interior de um elemento
u_i	Deslocamento ortogonal segundo o eixo dos x
v_0	Velocidade da bola no instante $t = 0$ segundos
v_i	Deslocamento ortogonal segundo o eixo dos y
x_0	Posição da bola no instante $t = 0$ segundos
z_i	Deslocamento ortogonal segundo o eixo dos z
ε_1	Deformação no ponto 1
ε_2	Deformação no ponto 2
$\varepsilon_{Plástica\ efetiva}$	Deformação plástica a colocar no Abaqus®

$\varepsilon_{cedência}$	Deformação no ponto de cedência do material
$\varepsilon_{engenharia}$	Deformação de engenharia
ε_{real}	Deformação real
ε_{total}	Deformação total
σ_1	Valor de tensão obtido para a deformação ε_1
σ_2	Valor de tensão obtido para a deformação ε_2
$\sigma_{engenharia}$	Tensão de engenharia
σ_{real}	Tensão real
$\prod$	Energia potencial total
Bottom_Bounding_Box_P1	Ponto 1 da bounding box do *Sketch_Bottom_Plane*
Bottom_Bounding_Box_P2	Ponto 2 da bounding box do *Sketch_Bottom_Plane*
displacement	Dicionário do deslocamento do impactor
Drawing_CSYS	Sistema de coordenadas do desenho
ε	Deformação
height	Altura do hexágono da geometria
$hex\_horizontal\_leg\_1$	Distância entre o ponto da extremidade direita, da linha horizontal, e o hexágono de base
$hex\_diagonal\_number$	Número de hexágonos a desenhar na diagonal
hex_size	Tamanho do hexágono
$hex\_thickness$	Espessura dos frisos
$hex\_vertical\_leg\_1$	Distância vertical entre o hexágono de base e a primeira linha limite
$hex\_vertical\_leg\_4$	Distância vertical entre o hexágono de base e a quarta linha limite

hex_x_lenght_2	Coordenada vertical do hexágono superior de um respetivo par
hex_x_lenght_base	Coordenada horizontal do hexágono vertical que dá origem à sequência diagonal
hex_y_lenght_1	Coordenada vertical do hexágono inferior de um respetivo par
hex_y_lenght_base	Coordenada vertical do hexágono vertical que dá origem à sequência diagonal
horizontal_hex_number_B	Número de hexágonos segundo a linha B
horizontal_limit	Distância entre o ponto da extremidade direita, da linha horizontal, e o eixo dos y do referencial a ser usado
horizontal_limit_3	Distância entre a linha horizontal limite 3, e o eixo dos x do referencial a ser usado
L	Largura
path	Guarda a diretoria onde se deve executar o ensaio seguinte
plastic_strain	Dicionário da deformação plástica
size	Largura do hexágono da geometria
Sketch_Center_Point	Centro do eixo de coordenadas do desenho
Sketch_Plane	Plano de extrusão superior
Sketch_Plane_Bottom	Plano de extrusão inferior
status	Indica se uma geometria passou ou não num dado ensaio
stress	Dicionário dos valores de tensão detetados durante o ensaio
test	Teste que foi realizado

thickness	Espessura do hexágono da geometria
Top_Bounding_Box_P1	Ponto 1 da bounding box do *Sketch_Plane*
Top_Bounding_Box_P2	Ponto 2 da bounding box do *Sketch_Plane*
weight	Peso da peça
A	Largura da linha A
$Altura\ da\ geometria$	Altura da geometria de referência do friso
$Altura\ disponível$	Altura disponível para a colocação dos frisos
B	Largura da linha B
C	Largura da linha C
D	Largura da linha D
E	Largura da linha E
F	Largura da linha F
G	Largura da linha G
H	Largura da linha H
Hex_height	Altura do hexágono
Hex_lenght	Largura das arestas do hexágono
$Hex\_thickness\_height$	Altura da espessura do hexágono
K	Matriz de rigidez global
$Largura\ da\ geometria$	Altura da geometria de referência do friso
$Largura\ disponível$	Largura disponível para desenhar os hexágonos diagonais
P	Força
R	Reações na Estrutura

T	Transposta
U	Energia de deformação
V	Volume de um corpo
W	Função ponderada
X	Número de vezes que é necessário aplicar uma carga unitária
Z	Resíduo
$base\_thickness$	Espessura da base
$first\_int$	Etapa do desenho
g	Aceleração da gravidade na Terra
i	Índice do friso
k	Constante elástica
$offset\_distance$	Distância do offset
p	Força distribuída
t	Instante de tempo
$v(t)$	Velocidade da bola após t segundos
$vertical\_hex\_number\_A$	Número de hexágonos máximo segundo a linha A
$vertical\_limit$	Distância entre a linha horizontal que serve de limite, e o eixo dos x do referencial a ser usado
$vertical\_limit\_1$	Distância entre a linha horizontal limite 1, e o eixo dos x do referencial a ser usado
$x(t)$	Posição da bola após t segundos
Δ	Deslocamento isostático
Δx	Variação da coordenada horizontal em cada ciclo

Δy	Variação da coordenada vertical em cada ciclo
δ	Deslocamento
ν	Coeficiente de Poisson
σ	Tensão normal
τ	Tensão tangencial
∂U^e	Trabalho virtual causado pelas tensões internas
∂W^e	Trabalho virtual causado pelas forças externas

GLOSSÁRIO DE TERMOS

2D	Duas Dimensões/Bidimensional
1D	Uma Dimensão/Unidimensional
3D	Três Dimensões/Tridimensional
Bottom-up	Algoritmo de malha Bottom-up
Double-sided surfaces	Formulação de contacto, no qual os dois lados de um elemento, e todas as suas arestas livres são tidas em consideração para a análise
Edge-based surfaces	Formulação de contacto, no qual se considera que apenas existe contacto nas arestas ao longo do perímetro do elemento
field output	Dados de campo, de uma determinada variável
Free Mesh	Algoritmo de malha Free Mesh
Geometry	Repositório de geometrias
Hard Contact	Formulação do *Tangential Contact* que declara que, quando a distância entre duas superfícies é nula, elas entram em contacto, e qualquer pressão existente é transferida entre elas
Hash Table ou *Hash Map*	Estrutura de dados
history output	Dados de história, de uma determinada variável
Job	Ambiente da simulação que é submetida para análise
Key	Chave que identifica um objeto num repositório\ dicionário
Node-based surfaces	Formulação de contacto, no qual os nós do elemento é que são considerados para a análise do contacto
Normal Contact	Corresponde a todas as componentes das forças normais às superfícies

Quicktime	Formato de um ficheiro de vídeo
Screenshot	Foto da tela do computador
Set	Conjunto de elementos
Single-sided surfaces	Formulação de contacto, no qual o utilizador especifica qual é o lado do elemento que entra em contacto
Sketch	Desenho
Soft Contact	Modelo de contacto, no qual o aumento de pressão não é instantâneo
Step	Passos/Etapas da simulação
Strain Rate	Velocidade de deformação
Structured	Algoritmo de malha Structured
Sweep	Algoritmo de malha Sweep
Tangential Contact	Trata das componentes relativas ao deslizamento entre as superfícies
Vertices	Repositório de vértices
Viewport	Janela de visualização no Abaqus®

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ÍNDICE

INTRODUÇÃO

1 INTRODUÇÃO

1.1 Contextualização

Quando uma empresa pretende produzir qualquer tipo de peças para um cliente, é do seu interesse gastar a menor quantidade de dinheiro possível na sua conceção, enquanto cumpre com as especificações pretendidas, de forma a maximizar os lucros. Para isto, existem diversas maneiras possíveis de o fazer, podendo estas ser de natureza logística, de gestão de recursos, contactos com fornecedores, ou como é no caso do foco desta tese, com o melhoramento do design da peça em questão.

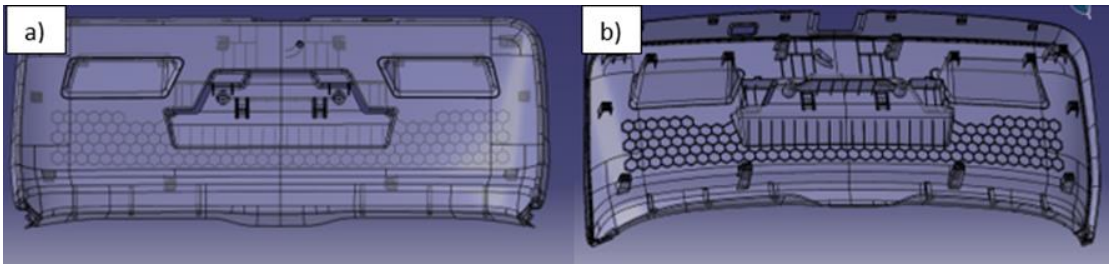


Figura 1 - Peça Original a) Vista Frontal b) Vista Traseira

A Simoldes Plásticos® tem então algumas peças de interior de uma bagageira de um automóvel, como a apresentada na Figura 1,em que gostaria de fazer essa mesma análise. Estas peças encontram-se normalmente sujeitas à flexão, sendo um requisito do cliente que nestas não seja ultrapassado um determinado limite de deformação. Para que isto não aconteça, estas peças possuem uns frisos, Figura 2, que servem de reforço, limitando assim a sua deformação no centro.

O trabalho proposto consiste em fazer uma análise recursiva com o software Simulia Abaqus®, de maneira a simular diversos ensaios nas quais as dimensões dos frisos vão variando em cada uma das simulações, e assim chegar aos frisos, cujas dimensões permitem obter as propriedades que o cliente pretende, gastando a menor quantidade de material possível, e assim obtendo um menor custo de produção. A tese tem então um foco no Método de Elementos Finitos, onde se vai estudar a melhor geometria de frisos, para diversas peças, através de um processo iterativo, com o auxílio das capacidades de *scripting* do software Simulia Abaqus®.

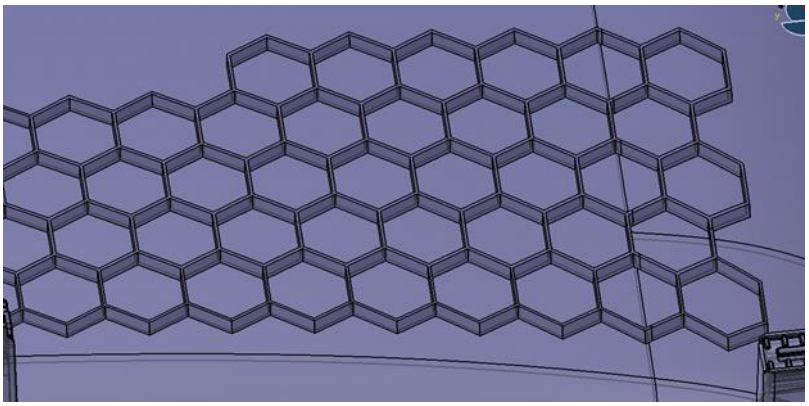


Figura 2 - Frisos do Reforço

1.2 Objetivos

O objetivo deste projeto é o de otimizar o design dos reforços em favo de abelha de componentes plásticos dos interiores de bagageiras de veículos, através de uma análise recursiva do seu comportamento à flexão e impacto. Essa análise foi então feita através do Método de Elementos Finitos (MEF), dentro do software de simulação Simulia Abaqus®, onde foram simulados diversos ensaios das peças á flexão, tendo em cada um deles se alterado a geometria e/ou a densidade do reforço. Apesar de ser possível fazer diversas simulações de forma não automática, foi também proposto que este processo fosse automatizado, através das funcionalidades de *scripting* do software, com a linguagem Python.

Com este trabalho espera-se então conseguir otimizar e parametrizar os reforços das peças, para que no futuro, seja gasto a menor quantidade de material possível nas encomendas dos clientes, resultando num menor custo de produção, por parte da empresa. É expectável que a parte deste projeto, correspondente á automação das simulações, venha a servir de base para a automatização de outros processos dentro da empresa.

1.3 Metodologia

1. **Desenvolvimento dos modelos base** – Consistiu na modelação das peças a estudar, através do Catia V5®, tendo sido feitas algumas alterações, através da remoção, ou supressão de determinadas características que não são importantes para o seu desempenho mecânico. Também foram produzidos os superficies médios da respetiva peça, de modo que, combinando com a

supressão de características, diminuísse a complexidade do modelo, resultando assim em simulações mais rápidas, e numa menor quantidade de recursos computacionais utilizados. De seguida exportaram-se os ficheiros do software Catia®, para o Abaqus®, onde se procedeu com o tratamento da Malha e uma simulação de um ensaio à flexão, numa peça que não possuía qualquer reforço, de maneira que os resultados obtidos servissem de base de comparação, para os que se realizaram no final do projeto.

2. **Análise de materiais e procedimentos** – Esta parte focou-se na análise dos cadernos de encargos das peças e das simulações, bem como as fichas técnicas dos materiais a serem usados, de modo a poder ser gerada uma matriz de possíveis parâmetros dos frisos do reforço, e dos parâmetros de malha a serem utilizados nas simulações.
3. **Parametrização das propriedades dos frisos** – Com os dados obtidos na fase anterior, foi então desenvolvida uma matriz de parâmetros, dos frisos do reforço, em que constam diferentes combinações de espessura, altura e densidade.
4. **Desenvolvimento dos programas** – Tendo a peça base e a matriz de parâmetros, produziu-se um programa em Python capaz de realizar diversas simulações no Abaqus®, de forma sequencial, mas em que os parâmetros dos frisos vão sendo alterados para as parametrizações definidas na matriz. Este software gera também um gráfico Deformação – Densidade, através dos resultados obtidos.
5. **Simulação final** - Nesta etapa final, executou-se o *script* produzido, e obteve-se a geometria dos frisos, que permitia obter uma melhor relação peso-deformação.

1.4 Estrutura do relatório

A análise bibliográfica deste trabalho divide-se em quatro áreas, de modo a poder estudar-se e compreender-se os conceitos básicos e as boas práticas que se devem ter, em cada uma das partes da componente prática do projeto. As áreas são então:

1. **Indústria automóvel** – Neste capítulo procurou-se estudar a importância que este setor tem, não só no ambiente socioeconómico nacional, mas também no mundial, tendo-se, portanto, dividido este em dois subcapítulos, no qual se analisaram os níveis de vendas, oportunidades de trabalho geradas, e contribuições monetárias, para cada uma das vertentes.
2. **Indústria de moldes** – Aqui analisou-se a indústria de moldes a um nível mais técnico, do que no automóvel. Estudou-se os moldes de injeção e como funcionam, mas também as características dos materiais utilizados na indústria,

e os ensaios mecânicos principais, utilizados na avaliação da qualidade dos componentes. Isto de forma a ficar-se a conhecer as matérias primas que são utilizados na componente prática, mas também, como deverão ser realizadas as simulações.

3. **Método de Elementos Finitos** – No capítulo "Método de Elementos finitos" são abordados conceitos relativos a como é formulado o cálculo matemático desta metodologia. Trata-se dos tipos de malha existentes, e quando se deve empregar cada uma. Abordam-se os diferentes campos onde o MEF pode ser aplicado. Os tipos de simulação existentes, podendo estas ser estáticas ou dinâmicas, lineares ou não lineares. Os elementos existentes nos modelos utilizados. Metodologias de simplificação de geometria. E por fim, as etapas de formulação do método.
4. **Softwares de Simulação** – Neste capítulo abordaram-se os diferentes tipos de software existentes, podendo estes ser gráficos, gráficos com capacidade de scripting ou algorítmicos. Para além disto abordam-se metodologias de automatização de processos, e como é feita a interligação do Abaqus® com a linguagem de programação Python.

1.5 Local/Empresa de acolhimento

Este projeto foi realizado na Simoldes Plásticos® (Figura 3), que é uma empresa do Grupo Simoldes®. O grupo foi fundado em 1959, em Oliveira de Azeméis, tendo apenas começado com a divisão responsável pela fabricação de moldes para a indústria de plásticos, a Simoldes Aços® [1].

Em 1890 deu-se então a formação da divisão de plásticos do Grupo Simoldes®, através da constituição da Simoldes Plásticos®. Esta parte da empresa foca-se exclusivamente na injeção de peças em materiais termoplásticos, aproveitando assim a sinergia entre as divisões de termoplásticos e a de fabrico de moldes [2].

Com esta sinergia entre os dois departamentos, a Simoldes Plásticos® trabalha principalmente com o setor automóvel, produzindo peças para os interiores de veículos. Contudo também possui projetos fora deste mercado, como a produção de bilhas de gás, a partir de materiais termoplástico. Os principais clientes incluem algumas das maiores empresas da indústria automóvel, como Renault, Ford, o Grupo PSA e o Grupo Volkswagen. Chegou mesmo a ser considerada "Fornecedor Q1", "Fornecedor Preferencial", "Fornecedor A" e "Fornecedor Majeur" pela Ford, Renault e o Grupo PSA, respetivamente [1].

Hoje em dia, o Grupo Simoldes® é um dos maiores fabricantes Europeus de moldes em aço, e um dos maiores produtores de peças termoplásticas da indústria automóvel. E,

com isso, conseguiu expandir e criar fábricas no Brasil, na Polónia, na Espanha, na França, na Argentina e na República Checa.

Figura 3 - Simoldes Plásticos®, Oliveira de Azeméis, Portugal [2]

REVISÃO BIBLIOGRÁFICA

2.1 Indústria automóvel

2.2 Indústria de Moldes

2.3 Método de Elementos Finitos

2.4 Software de simulação

2 REVISÃO BIBLIOGRÁFICA

2.1 Indústria automóvel

Após tantos anos, a indústria automóvel continua a ser uma das mais prevalentes no mundo, devido ao facto de o automóvel comum ser um bem intrínseco no deslocamento do dia a dia de uma pessoa, e em muitos casos uma necessidade para o individuo ou empresa desempenharem as suas funções. Apesar da sua prevalência, esta procurou sempre a inovação, e, portanto, ao longo dos tempos, tem vindo a adaptar--se ao respetivo ambiente socioeconómico. Hoje em dia essa adaptação passa pela integração de sistemas que permitem o uso de formas de energia renováveis.

2.1.1 Importância da indústria automóvel a nível mundial

Nos últimos anos, quando analisados os valores de vendas totais da *International Organization Of Motor Vehicle Manufacturers* (OICA) das referências [3-5] e representados na Tabela 1 pode-se evidenciar que a indústria tem-se mantido constante, tendo os níveis de vendas rondados os 95 milhões de unidades anuais. Contudo em 2020 a indústria sofreu um decréscimo de aproximadamente 16%. Isto pode estar relacionado com a influência que a pandemia de Covid 19 teve nas vidas das pessoas, visto que muitas ficaram desempregadas, ou estiveram em regime de teletrabalho, reduzindo a necessidade e possibilidade de compra.

Tabela 1 - Venda mundiais de todos os tipos de automóveis [3-5]

	2016	2017	2018	2019	2020
Vendas Mundiais de Veículos (Unidades)	94 771 814	96 706 293	95 706 293	92 175 805	77 621 582
Variação do Ano Anterior (%)	+1.05	-1.03	-1.10	-3.69	-15.80

Fazendo uma análise mais precisa, e utilizando os dados da *European Automobile Manifacturers Association* (ACEA) presentes em [6], e apresentados na Figura 4, sobre os níveis de produção de veículos nos anos de 2019 e 2020, em diferentes locais no mundo. Pode-se constatar que todos os continentes sofreram quedas drásticas dos seus níveis de vendas de automóveis, e a única razão pelo qual a queda não foi mais acentuada, foi por que a China é detentora da maior parte das vendas mundiais, e esta apenas sofreu uma queda de 2,2%.

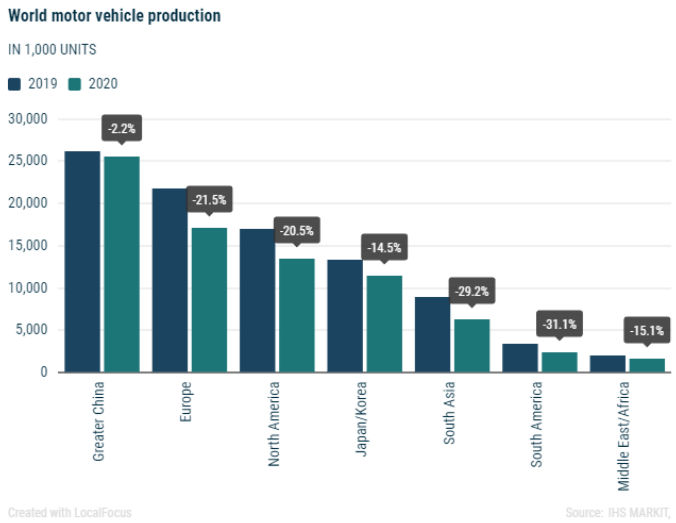


Figura 4- Comparação das vendas de automóveis de 2019 e 2020, em diversas regiões do mundo [6]

Contudo utilizando um estudo com um período de tempo maior, também da ACEA [7], pode-se comprovar que apesar desta queda, os valores não são muito diferentes dos que se tinha há uns anos atrás, pois as vendas têm vindo a subir, pelo menos desde 2005, como mostra na Figura 5.

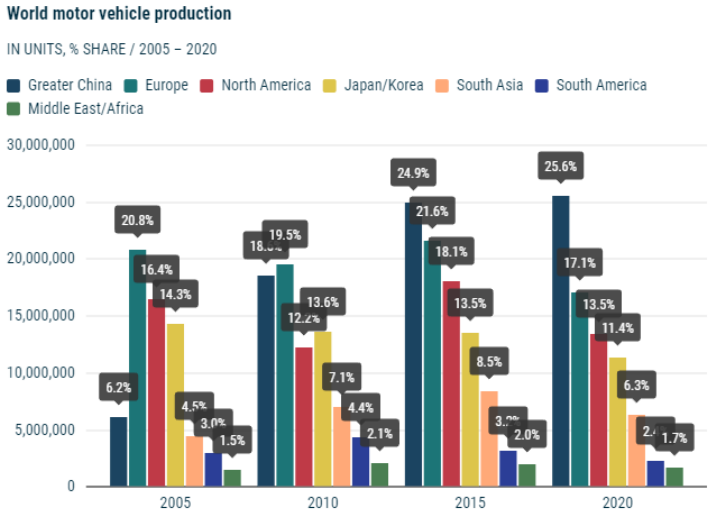


Figura 5 - Produção mundial de automóveis desde 2005 [7]

Como o setor automóvel tem uma enorme prevalência, como demonstrado pelos níveis de vendas, o número de postos de trabalho que oferece também é considerável, como demonstrado na Figura 6 [8, 9].

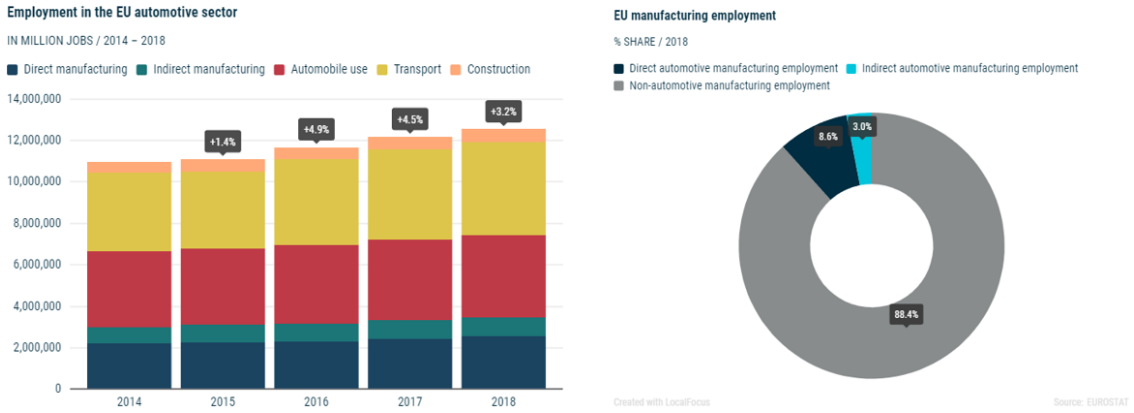


Figura 6 - Níveis de oportunidade de emprego no setor automóvel em 2018 [8, 9]

2.1.2 Importância da indústria automóvel a nível nacional

A indústria automóvel, segundo a Associação de Fabricantes para a Indústria Automóvel (AFIA) [10], é um dos maiores contribuidores para a economia portuguesa, sendo esta dividida em 3 grandes setores [11]:

- Fabrico de moldes;
- Fabrico de componentes;
- Fabrico de viaturas automóveis.

Dentro destes setores os maiores produtores a nível nacional são a Simoldes®, a Toyota Caetano®, a Mitsubishi Fuso Truck Europe®, a Peugeot Citroen® e a Autoeuropa® (Figura 7).

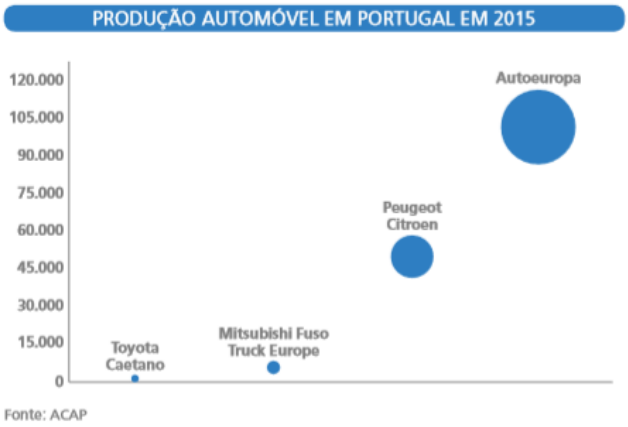


Figura 7 - Produção automóvel de diferentes empresas em Portugal, em 2015 [11]

Tabela 2 - Contributo do setor automóvel para o PIB nacional [11]

Contribuição da indústria automóvel para o PIB Português (%)	INE (2014)	INE (2015)
PIB	0.9	1.5
Exportações, bens e serviços	3.9	5.1

Esse mesmo estudo revela que em 2015, o setor automóvel foi responsável por 11% das exportações do país, sendo que ocupa cerca de 200 empresas e 42 mil postos de trabalho.

2.2 Indústria de Moldes

No Capítulo 2.1.2, foi mencionado que dois dos maiores setores da indústria automóvel são o fabrico de moldes e a produção de peças. Muitas vezes estes dois estão muito relacionados, pois um automóvel moderno possui várias peças poliméricas, que são produzidas através de um processo de injeção do respetivo material num molde que lhe confere a forma final. Um exemplo de uma empresa, em Portugal, que tem estes dois setores presentes é a Simoldes®, possuindo uma divisão

que trata da produção de moldes, e outra que trata da produção das peças, de maneira a possuir uma vantagem competitiva.

2.2.1 Moldes de Injeção

Um molde de injeção é uma ferramenta fulcral na produção de peças plásticas, pois é este componente o responsável por dar a forma final ao componente. Associado a este tem-se também uma máquina de injeção, para introduzir o material no interior do molde, um sistema de arrefecimento permite que o componente seja arrefecido durante o processo, e um sistema electro-hidráulico para garantir o fecho e a abertura do molde. Para além destes sistemas podem existir alguns extras, como um braço mecânico que permita a extração da peça no molde. Devido à quantidade de peças existentes, espera-se que este seja capaz de produzir pelo menos 100 mil peças, como indicado por Zabala [12].

2.2.2 Materiais utilizados na indústria

Na indústria de moldes, os materiais mais utilizados para a produção de peças são os de origem polimérica. Isto porque o seu baixo custo, versatilidade e produtividade alta, permitem que se possam obter peças de matriz polimérica com a capacidade de desempenhar diversas funções em setores diferentes, como o automóvel ou a indústria aeroespacial, a baixos custos de produção [13].

2.2.2.1 Materiais poliméricos

Os polímeros são uma classe de materiais formados por longas cadeias de unidades estruturais de menor dimensão (monómeros), que estão ligadas entre si através de ligações covalentes. A essa cadeia de monómeros dá-se a designação de macromoléculas [14]. Por sua vez os monómeros podem ser classificados de homopolímeros, se forem constituídos por monómeros do mesmo tipo, ou copolímeros, se estes forem de tipos diferentes. Dependendo da maneira como estes estão organizados, a sua estrutura pode ser classificada como linear ou ramificada, e isto influencia drasticamente a densidade do material.

Por sua vez as macromoléculas podem-se arranjar de maneira a possuírem uma estrutura tridimensional reticulada, que pode ser classificada como linear, ramificada ou em rede [13].

Quanto à estrutura, as macromoléculas podem ainda ser amorfas ou cristalinas. Os polímeros amorfos não têm ordenamento estrutural das cadeias, sendo a ligação carbono-carbono a região ordenada de maior dimensão. Como esta ligação possui um

comprimento de onda menor que o da luz visível, este tipo de polímeros é então transparente. Quando existe alinhamento das cadeias moleculares (acima de 90% de cristalinidade), considera-se que o material é formado por agregados de cristais (Esferulite) e o polímero diz-se cristalino. O aumento do grau cristalino leva a um aumento da densidade, resistência e dureza do material, mas diminui a flexibilidade e a resiliência [14]. Para além destes dois tipos, o material pode possuir um grau intermédio chamado de semi-cristalino, quando o grau de cristalinidade é abaixo dos 90%, isto resulta em propriedades intermédias entre os dois, e as esferulites designam-se de cristalites [14].

Para além das classificações mencionadas, os polímeros podem ainda ser materiais termoplásticos, termoendureciveis ou elastómeros, consoante o seu comportamento quando se introduz calor [13]. Os termoplásticos amolecem nestas condições e endurecem quando arrefecidos, e, portanto, possuem diferentes graus de ductilidade e não suportam temperaturas elevadas, contudo podem ser reciclados. Salvo certas exceções, os termoendureciveis já se tornam permanentemente rígidos quando submetidos a um ciclo de aquecimento-arrefecimento, tornando-se frágeis. Isto permite que eles suportem temperaturas elevadas, mas impedem a sua reciclagem. Os elastómeros, ou mais bem conhecidos como Borrachas, são polímeros com uma estrutura levemente reticulada, o que lhes confere a capacidade de recuperarem a sua forma inicial, quando são deformados [14].

Os polímeros raramente são utilizados por si só, normalmente são introduzidas impurezas na sua constituição, de maneira a alterar as suas propriedades mecânicas para aquelas que se pretendem para o trabalho. Essas impurezas resultam do seu processamento e da adição de aditivos. Quando um polímero possui essas impurezas, aí pode passar a ser considerado um plástico. Alguns dos aditivos utilizados na indústria são: os estabilizantes, as cargas, os plastificantes, os pigmentos e lubrificantes, expansores, ignifugantes, anti-estáticos, anti-fugantes, desodorizantes, modificadores de impacto e outros polímeros [15].

Alguns dos plásticos comerciais mais importantes são: as Poliamidas (PA), o Poliestireno (PS), os Acrílicos, os Policarbonatos (PC), o Poliéster, os Fluoropolimeros, as Poliolefinas, o Estireno e o Polipropileno (PP) [15].

2.2.2.2 Polipropileno e Polipropileno carregado

No âmbito deste trabalho destaca-se o PP, visto que este é o material utilizado na componente prática. O PP é um termoplástico, é muito utilizado na indústria, devido a ter: uma boa dureza, estabilidade dimensional, boa resistência à nucleação e propagação de fissuras, elevada resistência química, fácil capacidade de produção e baixo custo, ser resistente ao calor e possuir a menor densidade de todos os

termoplásticos. Isto torna-o apto para a produção de peças de interiores de automóveis, equipamentos hospitalares, produtos de laboratórios e brinquedos [16].

Para além disto a sua rigidez e facilidade de produção permite que sejam feitas peças de secções finas, que é algo que não é facilmente concebível com outros materiais [16].

Quanto ao seu estado bruto, este normalmente é fornecido sob a forma de partículas cúbicas ou cilíndricas, e pode ter a cor natural do polímero, ou vir numa vasta variedade de cores [16].

Quando se quer alterar as propriedades deste material, opta-se pela introdução de cargas na formulação, como o talco e o carboneto de cálcio, o que vai permitir aumentar a rigidez e a dureza [16]. No caso do talco, a sua introdução também faz com que seja mais fácil observar a existência de fissuras, pois estas passam a ser demarcadas por uma linha branca. Normalmente as cargas também são introduzidas para reduzir o preço das peças, visto que estas são mais baratas do que o PP puro. A introdução destas cargas leva ao que se designa na indústria de PP carregado [14].

2.2.2.3 Ensaios mecânicos em peças poliméricas

Os tipos de ensaios mecânicos que se podem praticar nos materiais poliméricos são praticamente os mesmo que se fazem nos outros tipos de materiais. Sendo assim, os testes mais comummente feitos nesta vertente são:

Ensaio de Tração – Este ensaio consiste em prender um provete, feito do material que se quer estudar, nas suas extremidades, e de seguida aplicar um deslocamento gradual a uma delas. Em cada instante do ensaio, são lidos e registados os valores de deslocamento e da força a atuar, de maneira a poder-se produzir a curva tensão--deformação característica do material. Após a rotura do mesmo analisa-se a curva obtida e retiram-se as propriedades necessárias, como a tensão de cedência, tensão de rotura, o módulo de elasticidade e o coeficiente de Poisson [17, 18].

Ensaio de Compressão – No ensaio de compressão é aplicada uma carga compressiva a um corpo de prova, sendo os resultados obtidos através da medição da distância entre as placas que geram o respetivo esforço e a força que elas provocam, em cada instante [19].

Ensaio de Flexão – No ensaio de flexão existem duas variantes o ensaio em 3 pontos e em 4 pontos. Nos dois casos o corpo de prova é posicionado sobre dois apoios, sendo de seguida aplicada uma carga transversal. Dependendo do tipo de ensaio, esta carga é aplicada em apenas num, ou em dois pontos da superfície da peça. As propriedades do material são obtidas através da análise da curva, carga aplicada-deslocamento da peça [19].

2.3 Método de Elementos Finitos

O MEF é um método numérico utilizado nos ramos da matemática e engenharia, que permite a análise de modelos bidimensionais e tridimensionais, quando nestes são impostas ações externas. Esses modelos podem provir de uma enorme variedade de campos diferentes da engenharia, como da análise estrutural de uma ponte, o estudo de uma transferência de calor e até mesmo para o estudo de campos magnéticos [20].

Antes do aparecimento do MEF, este tipo de problemas era estudado pela resolução de sistemas de equações e derivadas parciais, relativas ao respetivo campo científico [20]. Contudo, para problemas de elevada complexidade, não costuma ser possível obter soluções analíticas, e, portanto, o MEF subdivide os modelos em estudo, em elementos geométricos mais pequenos (Elementos Finitos), ligados uns aos outros através de pontos comuns. A este processo de simplificação dá-se o nome de discretização, podendo ver-se um exemplo deste processo na Figura 8 [21, 22].

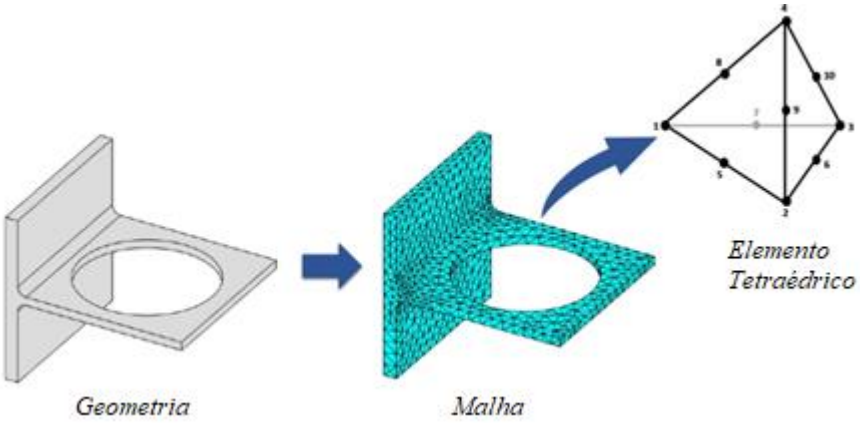


Figura 8- Exemplo de discretização

Devido à elevada complexidade matemática, inicialmente, este método não era preferido a outros correntes. Contudo, com o aparecimento e proliferação dos computadores, houve uma explosão na sua utilização, chegando mesmo ao ponto de os outros métodos concorrentes terem caído em desuso [20]. Com o passar dos anos, devido ao aumento da velocidade de processamento dos computadores, como verificado na Figura 9 [22], o tempo necessário para se obter os resultados de uma simulação tem diminuído, tornando o processo muito atrativo para as empresas analisarem os seus componentes. Para além disto, o MEF levou a que fossem necessários um menor número de protótipos de validação de componentes físicos, reduzindo assim os custos.

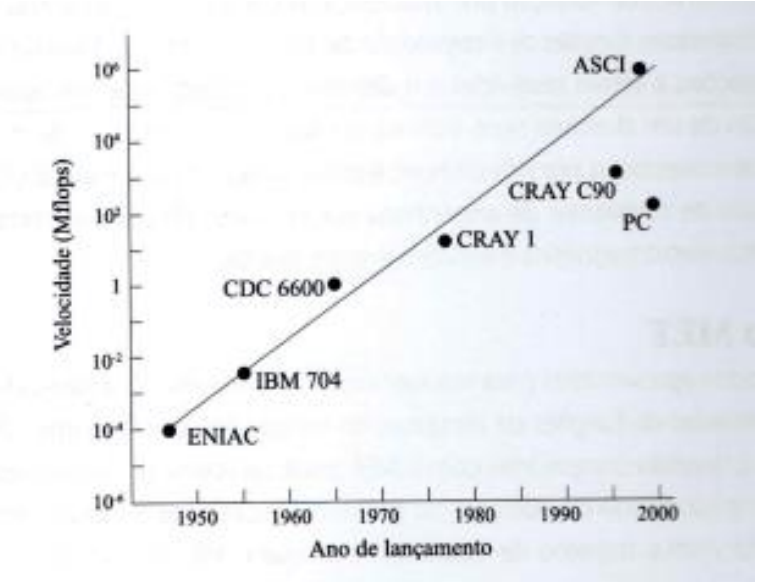


Figura 9 - Evolução da velocidade dos computadores [22]

2.3.1 Princípio de Funcionamento

No estudo de um determinado volume de material, cujas propriedades físicas são previamente conhecidas, a formulação matemática do MEF requer a existência de uma equação integral algébrica, que possa ser descrita através do somatório das equações, dos respetivos subdomínios que o constituem [20], como demonstrado na equação(1):

$$\int_V f \, dV = \sum_{i=1}^{n} \int_{V_i} f \, dV \tag{1}$$

onde, o volume de material é representado por V e o volume dos seus subdomínios por V_i, de acordo com a equação (2):

$$V = \sum_{i=1}^{n} V_i \tag{2}$$

Esta metodologia, que consiste em analisar um corpo complexo, através do estudo dos seus componentes mais simples, provém da aplicação do método dos resíduos ponderados (MRP). Este é utilizado devido ao facto de, quando se pretende estudar uma propriedade de cada um dos pontos de um corpo, o processo é praticamente impossível de se fazer, devido à infinidade de pontos existentes num sistema bidimensional ou tridimensional, sendo então necessário reparti-lo em porções mais pequenas, como representado na Figura 10 [23].

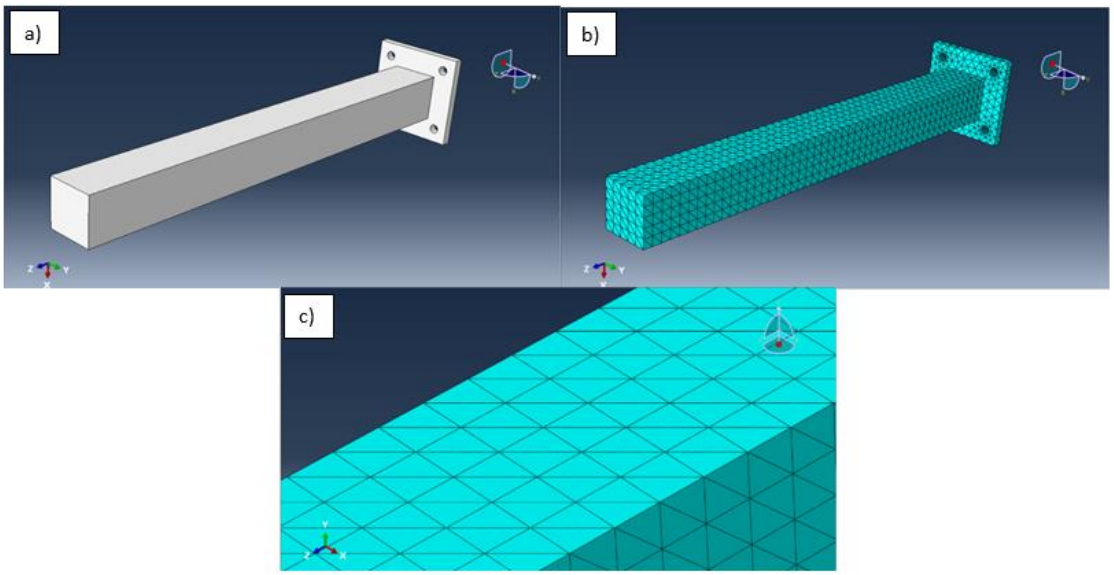


Figura 10 - Modelo dividido em porções tetraédricas a) Original, b) Repartido, c) Elemento (Fonte própria)

Neste processo de repartição, tanto a geometria, como o número de elementos utilizado, influenciam o resultado do estudo, sendo que, de uma forma geral, quanto maior for o número de elementos, maior será a precisão da simulação. Contudo, com uma maior precisão tem-se também uma maior complexidade do problema, o que resulta em tempos de análise computacional maior. Por isso, quando se pretende fazer uma análise através do MEF, é extremamente importante determinar os parâmetros da repartição, de maneira a ter-se um bom equilíbrio entre a precisão da resposta, e o tempo de simulação [20, 23].

Os vértices dos elementos geométricos utilizados na repartição da superfície são designados por nós. E é nesses pontos que são calculadas as variáveis de campo. Contudo é também necessário obter a distribuições nos restantes pontos dos elementos. Nesses casos, os valores são obtidos por intermédio de interpolação dos valores obtidos nos pontos nodais, como na equação (3) [22]:

$$\phi(x,y,z) = \sum_{i=1}^{n} N_i(x,y,z) * \phi_i \tag{3}$$

onde $\emptyset_i$ corresponde aos valores das variáveis de campo nos nós e N_i às funções de interpolação, que são normalmente polinomiais lineares, quadráticas ou cúbicas [23].

As variáveis de campo $\emptyset$, em contexto mecânico, são normalmente deslocamentos, e podem ter direções ortogonais u_i e v_i, caso o modelo seja bidimensional, ou u_i, v_i e z_i, em contexto tridimensional [22], porém podem também representar outras propriedades físicas. Como no exemplo demonstrado em [24], onde este representa a temperatura nas fronteiras, numa análise da transferência de calor de um prisma quadrangular.

Nos casos em que existem diversos elementos ligados a um mesmo nó, de maneira a conferir a continuidade do modelo, estes possuem os mesmos valores. Contudo esta continuidade pode não ocorrer nas derivadas em cada um dos pontos. Como no caso da deformação em análises estruturais, em que não existe continuidade, pois a deformação resulta da primeira derivada da variável de campo de deslocamento [22, 25].

Analisando com atenção a equação (3), pode-se também comprovar, que quanto maior for o número de nós que formam um domínio, maior será a precisão da simulação [23]. Isto pode ser visto, através da análise de um elemento cujas propriedades exatas já são nossas conhecidas. Para este exemplo demonstra-se um teste realizado na referência [22], em que se estuda a deformação imposta num cilindro cónico, quando neste é aplicado um carregamento de tração.

Como se pode ver na Figura 11, analisaram-se três exemplares do respetivo cilindro. Um com a geometria real, outro utilizando apenas um elemento finito na sua aproximação, e por fim um com dois elementos, obtendo-se assim os resultados apresentados na Figura 12.

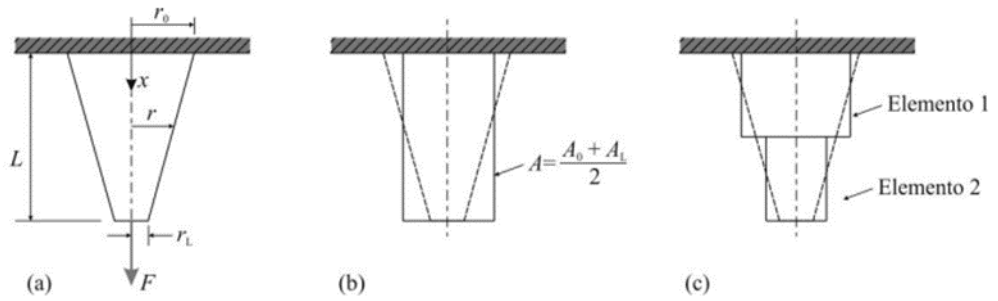


Figura 11 - Teste de um cilindro cónico a) Geometria Real b) Aproximação a 1 elemento c) Aproximação a 2 elementos [1]

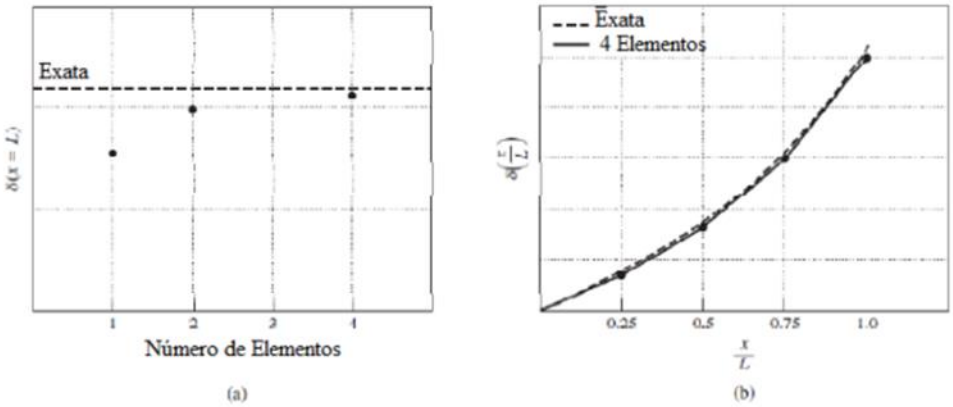


Figura 12 - Resultados da deformação do cilindro cónico a) Deslocamento na extremidade livre b) Deslocamento axial em função da posição ao longo do comprimento [1]

Como se observa nos resultados obtidos, quanto maior for o número de elementos utilizados, maior será a precisão da simulação, porém, a partir de certo ponto deixa de ser viável aumentar o seu número, pois o aumento de precisão passa a ser negligenciável, quando comparado com o aumento de tempo de resolução do problema e o aumento dos erros devido a arredondamentos.

Neste tipo de problemas, as variáveis que se pretendem estudar são as deformações e as tensões instaladas, e estas são derivadas das variáveis de campo. As deformações são obtidas através da Lei de Hooke, e as tensões pelas relações tensão-deformação da Teoria da Elasticidade [25]. Aplicando isto ao modelo do cilindro cónico, obtém-se os resultados da Figura 13.

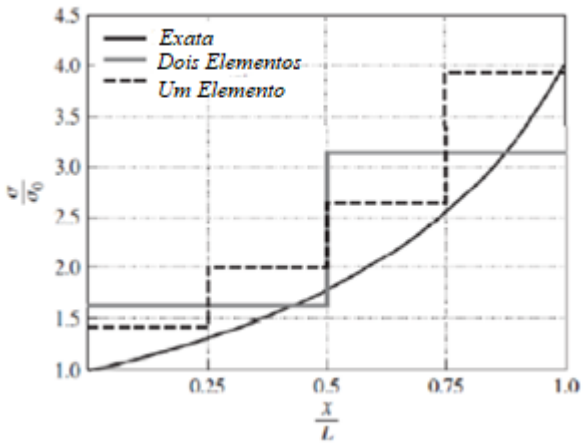


Figura 13 - Tensões axiais no cilindro ao longo do seu comprimento [22]

Analisando este gráfico, constata-se que as tensões são constantes ao longo de cada elemento, havendo, como já foi referido anteriormente, descontinuidades entre os valores de cada um. No entanto, com o aumento do número de elementos, essas descontinuidades vão diminuindo, sendo que quando o seu número tende para infinito, as discrepâncias tendem a ser nulas, e o gráfico apresenta um comportamento exponencial, que é igual ao do modelo real [22].

Com isto em mente, é extremamente importante avaliar sempre o modelo do MEF [26], quanto a:

- Veracidade dos resultados obtidos;
- Descontinuidade das variáveis de campo derivadas, nos pontos de fronteira;
- Convergência numérica do resultado obtido com o real;
- Consistência das leis físicas do problema em questão (se o caudal de entrada é igual ao de saída ou se a quantidade de energia que entra num sistema não é menor que a que sai);

2.3.2 Malha

Como já foi referido anteriormente, a discretização do sistema que se pretende estudar é um dos passos mais importantes da aplicação do MEF, e este tem de ser corretamente executado, de maneira que os resultados obtidos sejam fidedignos e a operação não seja morosa. Para isto recorre-se à divisão do sistema em porções mais pequenas, representadas por elementos geométricos bidimensionais ou tridimensionais, dependendo do modelo. Essas geometrias podem apresentar as formas demonstradas na Figura 14 [27], e têm as características representadas em [28].

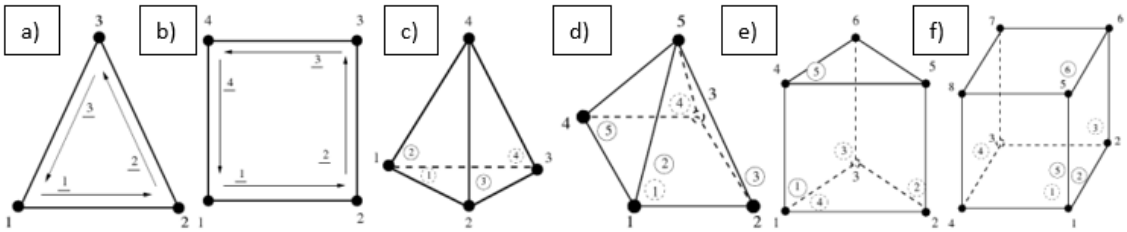


Figura 14 -Tipos de geometria dos elementos de malha [27]

Elementos 2D [28]:

- 2D – Triângulo (Figura 14 a) – Devido ao seu menor número de faces, este tipo de malha é o mais simples de todos, e, portanto, é utilizado apenas em simulações de grelhas não estruturadas.

- 2D – Quadrilátero (Figura 14 b) – Devido a possuir um maior número de arestas, e, portanto, maior complexidade que o Triângulo, este é utilizado em estruturas bidimensionais de grelha estruturada.

Elementos 3D [28]:

- 3D – Tetraédrica (Figura 14 c) - São os mais utilizados, sendo normalmente gerados automaticamente pelo próprio software, devido à sua fácil computabilidade, quando comparado com os outros elementos 3D.
- 3D – Piramidal (Figura 14 d) - Este tipo de elemento apenas serve como meio de transição entre tipos de malha com face triangular para retangular, ou vice-versa. Como por exemplo entre a estrutura de pirâmide e a do hexaedro.
- 3D - Prisma Triangular (Figura 14 e) - A vantagem deste tipo de elemento, é o facto de ser muito eficiente a tratar de condições de fronteira, na camada limite (*Boundary Layer*).
- 3D – Hexaédrica (Figura 14 f) - Devido ao seu maior número de arestas, e, portanto, maior complexidade, a hexaédrica é a que resulta num resultado mais preciso. A par disto, as estruturas de Pirâmide e Prisma Triangular são normalmente consideradas como formas mais simples da Hexaédrica.

Para além do tipo de geometria tem de se ter em conta a deformação existente nesses mesmo elementos. Pois caso estes estejam muito deformados, pode originar erros de interpolação no cálculo das variáveis de campo (Figura 15). Esta deformação é analisada através do rácio de dimensões (*aspect ratio)* da malha, que corresponde à razão existente entre as arestas de maior e menor dimensão do elemento. O seu valor deve ser o mais próximo possível de um [29]. Uma forma de controlar a distorção, é recorrendo ao refinamento local da malha, podendo-se utilizar dimensões mais pequenas em pontos de geometria mais complexa, e valores maiores em situações mais simples.

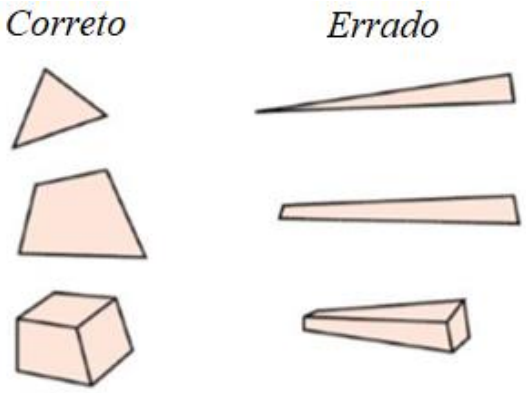


Figura 15 - Deformação da geometria da malha [29]

Outro fator importante na geração da malha é o próprio algoritmo matemático usado na sua síntese. Esta área é atualmente objeto de investigação sendo um alto foco de estudo de diferentes departamentos de engenharia, física, ciências da computação e matemática, de diversas partes do mundo. Na Tabela 3 estão representados alguns dos algoritmos desenvolvidos por essas mesmas entidades, bem como uma breve descrição de como funcionam.

Tabela 3 - Algoritmos de geração de malha [30-33]

Algoritmo	Descrição
A 2D advancing-front Delaunay mesh refinement algorithm [30]	*"I split the line segments of an input planar straight-line graph (PSLG) such that the lengths of split segments are asymptotically proportional to the local feature size at their endpoints. By employing prior algorithms, I then refine the truly or constrained Delaunay triangulation of the PSLG by inserting off-center Steiner vertices of "skinny" triangles while prioritizing triangles with shortest edges first. This technique inserts Steiner vertices in an advancing front manner such that we obtain a size-optimal, truly or constrained Delaunay mesh if the desired minimum angle is less than 30◦ (in the absence of small input angles)."*
Adaptive moving mesh algorithm based on local reaction rate [31]	*"Our new algorithm is based on the revelation, that in reaction-diffusion systems the high moving concentration gradients appear nearby to the region where the rate of reaction is maximal, thus the local reaction rate can be used to control the mesh adaption. We found that the main advantage of such a method is its simplicity and easy implementation."*
Improved block-Jacobi parallel algorithm for the SN nodal method with unstructured mesh [32]	*"In this study, we applied the block-Jacobi algorithm to the SN nodal method with a triangular-z mesh and proposed an improvement to achieve high parallel efficiency. The interface prediction (IP) method was developed to prevent iteration degradation. This method is based on extrapolating the interface information instead of using the information from the preceding iteration at the interfaces in sub- domains."*

A new topology optimization framework for stiffness design of beam structures based on the transformable triangular mesh algorithm [33]	*"In the proposed method, we first triangulate the initial geometry and organize it with the half edge data structure. Then we employ the Laplacian energy-controlled mesh deformation algorithm to realize smooth transition from the initial TTM. Furthermore, three geometrical mesh processing techniques (mesh subdivision, mesh split, and mesh refinement) are adopted to realize surface genus change."*

2.3.3 Aplicações do MEF

Devido à sua versatilidade, o MEF é utilizado em diversos tipos de indústria, e não está limitado apenas a aplicações no ramo da engenharia mecânica. Alguns exemplos de aplicabilidade do MEF são:

Crash Test **–** Principalmente na indústria automóvel, é vantajoso para a empresa responsável conseguir testar modelos das peças que produz, sem ter de recorrer a ensaios destrutivos dos modelos reais, de modo otimizar o design a um custo reduzido. Um exemplo destas aplicações são os *crash tests* em automóveis. Nestes testes, o veículo é totalmente ou parcialmente produzido, para depois se replicar uma colisão, com o intuito de estudar a sua resposta a um acidente, e se este cumpre as normas de segurança. Recorrendo ao MEF, é possível reproduzir o mesmo estudo, mas num contexto virtual, reduzindo assim a necessidade de recorrer a modelos físicos mais dispendioso. Em [34], como o objetivo era o de verificar se um novo material para o fabrico de postes luminosos é mais seguro numa situação de colisão com um veículo leve de passageiros, os responsáveis, em vez de realizarem diversos ensaios físicos, utilizaram o software Simulia Abaqus®, onde testaram vistualmente não só diferentes materiais, mas também diferentes espessuras para o poste (Figura 16).

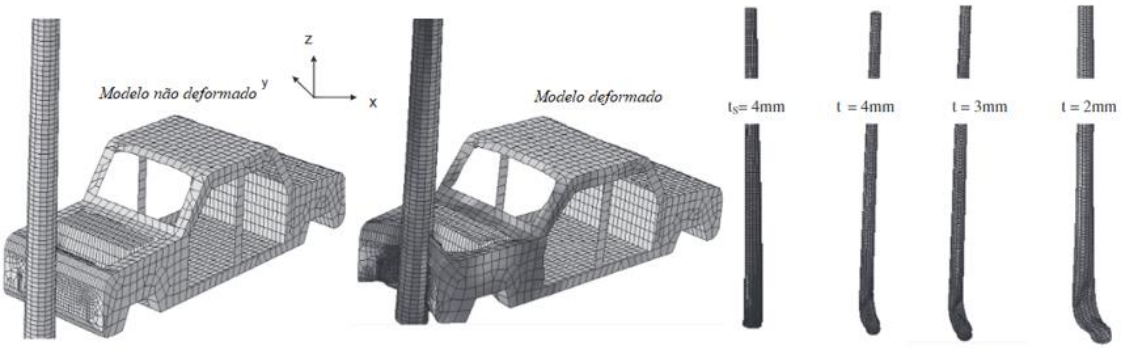


Figura 16 - Crash test com um poste de luminosidade [34]

Outro exemplo é o da referência [35], onde se estudou o comportamento mecânico de um novo tipo de para-brisas, numa situação de impacto com a cabeça de um pedestre, e portanto, foi utilizado o MEF, de maneira a não só se verificar os resultados da geometria convencional, mas também testar espessuras diferentes e novas fixações (Figura 17).

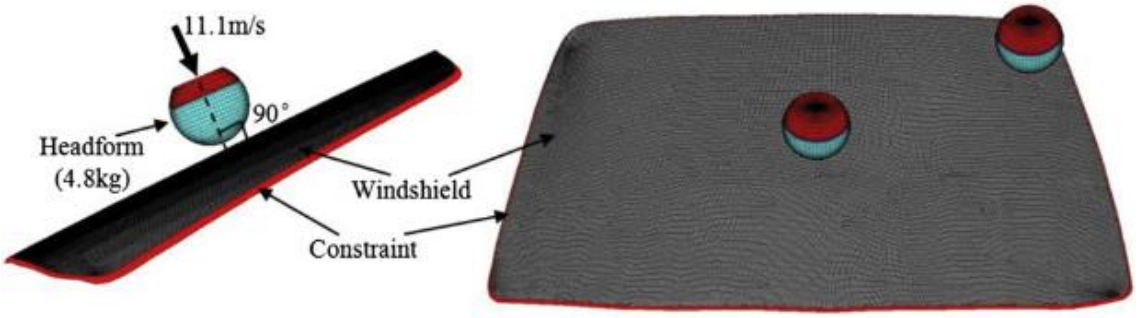


Figura 17 - Crash test de um para-brisas [35]

Análise de PCBs – *Printed Circuit Boards* (PCB), são circuitos eletrónicos com circuitos integrados *(Integrated Circuit* (IC)). Um exemplo deste tipo de sistemas são os componentes de computadores, como as *motherboards*. Apesar de estes serem do ramo da eletrotecnia, durante o seu desenvolvimento é necessário recorrer a diversas análises através do MEF. Um destes testes é a análise térmica dos diferentes componentes eletrónicos. Como parte da energia elétrica que entra no sistema é perdida sobre forma de calor, o sistema tem a tendência a aquecer. De maneira a confirmar que o calor gerado não põe em causa a integridade do PCB, são feitas análises térmicas através do MEF. Um exemplo disto é o estudo feito em [36], onde se pretendeu estudar se a energia libertada por um encapsulamento *Quad Flat No Leads* (QFN) punha em causa a integridade do PCB onde estava integrado (Figura 18).

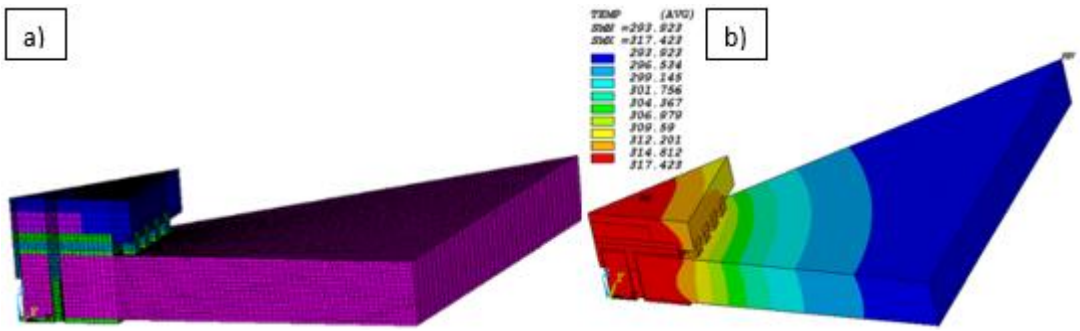


Figura 18 - Análise térmica de um PCB a) Malha b) Distribuição da temperatura [36]

Análise aerodinâmica em sistemas aeronáuticos – Na indústria aeronáutica, é muito importante que as condições de trabalho dos diversos componentes não levem à falha catastrófica da aeronave. Para que tal não aconteça, o MEF é utilizado para fazer análises estruturais dos diversos componentes. No artigo presente em [37], pode-se ver como é feita a análise mecânica de uma asa de um avião, e os diferentes passos envolvidos, desde a criação dos modelos, até à obtenção dos resultados (Figura 19 a)). Por outro lado, também são muito utilizadas as análises "Dinâmica dos Fluídos Computacional" (*Computational Fluid Dynamics (CFD)*), que permitem fazer um estudo aerodinâmico dos componentes, e estudar o comportamento do ar quando este passa pelo perfil da asa. Isto é muito importante, pois é a partir disto que se sabe se a geometria é capaz de gerar um gradiente de pressão suficientemente alto para que a aeronave levante voo. Na Figura 19 b) temos então o resultado de uma destas análises, apresentada em [38].

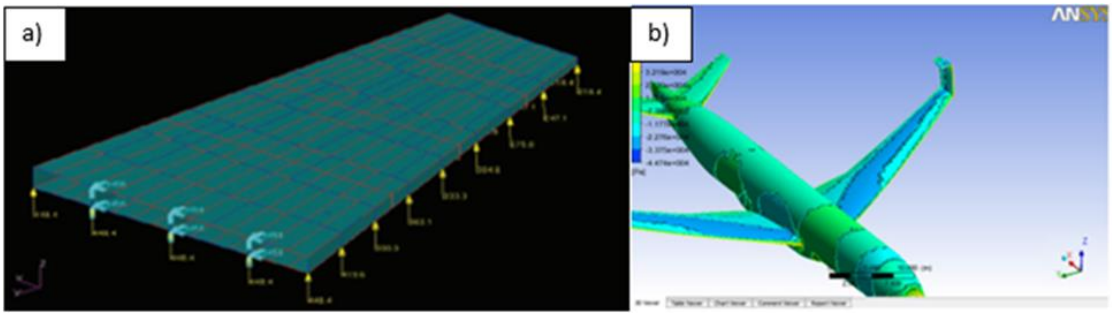


Figura 19 - a) Análise estrutural da asa de uma aeronave [37] b) Análise CFD de uma aeronave [38]

Para além destes exemplos, existem muitas mais aplicações do MEF, como por exemplo:

- Análise sísmicas de barragens [39];
- Análise eletromagnética de componentes eletrónicos [40];
- Análise de operações cirúrgicas, como a reconstrução facial [41];
- Análises acústicas [42];
- Análise de vibrações [43];
- Dispositivos médicos [44].

2.3.4 Tipos de simulações mecânicas no MEF

Dentro do próprio método de elementos finitos existem diferentes tipos de simulações que se podem fazer, dependendo da situação que se está a analisar. Para que a simulação seja o mais correta possível, é necessário estudar o evento a que o modelo vai estar sujeito, de maneira criteriosa e selecionar o tipo de simulação correta. Como

a vertente deste projeto está virada para a análise estrutural mecânica de componentes, apenas são explicadas as simulações deste tipo, e ignoradas outras como as Eletromagnéticas. Tendo isto em conta, existem dois tipos de simulações [20]:

- Simulação Estática;
- Simulação Dinâmica;

Sendo que, por sua vez cada uma delas pode ser uma:

- Análise Linear;
- Análise Não Linear.

2.3.4.1 Simulações Estáticas ou Dinâmicas

No mundo real, todos os corpos estão sujeitos a esforços dinâmicos como as forças de inércia associadas às acelerações dos seus elementos. De maneira que a simulação seja o mais fidedigna possível, considera-se que se trata de uma Simulação Dinâmica. Neste tipo de análise, o software considera a existência desses mesmos efeitos. Contudo como em muitas situações a velocidade de aplicação dos esforços é lenta, é razoável assumir que os efeitos dinâmicos são desprezáveis e procede-se com uma Simulação Estática, de maneira a reduzir a complexidade do sistema, e consequentemente o tempo de análise [20, 45].

2.3.4.2 Análise Linear ou Não Linear

Quando uma força é aplicada num componente, ao longo de um determinado período de tempo, este tende a deformar. Em praticamente todas as situações essa deformação influencia o modo de carregamento e ou outras características da simulação. Uma maneira de visualizar esta questão é com o exemplo na Figura 20, no qual está representada a deformação de uma cana de pesca ao longo do tempo. Como se pode ver, a distância que a força faz com o centro de rotação da cana, vai aumentando, resultando num maior momento atuante, neste elemento. A simulação não linear geométrica tem em atenção este efeito, e, portanto, permite que as propriedades da simulação se vão alterando ao longo do tempo, levando a um resultado mais próximo do real, ainda que o processo seja de uma complexidade maior.

Nas situações em que os deslocamentos resultantes, são pequenos, quando comparados com as dimensões dos componentes da estrutura, admite-se que não existe influência da modificação da geometria da estrutura, na distribuição dos esforços e tensões. Isto leva a que a simulação tenha um menor tempo de resolução.

Para além disto, o sistema linear é também escalável, ou seja, caso a força aplicada duplique, os resultados aumentam numa mesma proporção, o que leva a que seja necessário um número de simulações reduzido, graças aos resultados serem mais facilmente previsíveis [20].

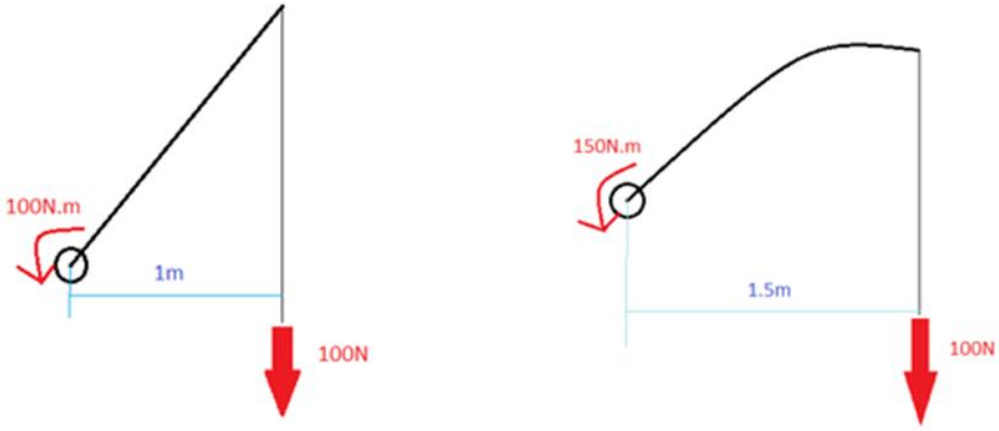


Figura 20 - Demonstração de um caso de não linearidade

2.3.5 Metodologias de aplicação do MEF

O objetivo de uma análise estrutural é calcular as tensões e deformações de uma estrutura quando nesta são aplicados esforços externos. Para tal, o MEF disponibiliza diversas abordagens que permitem obter as matrizes e os vetores característicos dos elementos finitos. Destas abordagens realçam-se os [22]:

- Métodos Diretos;
- Métodos Variacionais;
- Métodos dos Resíduos Ponderados;

***Métodos Diretos (MD)* –** Os MDs consistem em estabelecer as matrizes e vetores característicos dos elementos, através do uso da mecânica tradicional, sendo as forças ou as deformações, as variáveis a analisar. Contudo, devido à sua simplicidade, estes não podem ser aplicados a elementos 2D ou 3D, pois, quando são utilizados nestas situações normalmente surgem problemas intransponíveis. Apesar disto, o estudo deste método permite interpretar melhor a componente física do MEF [20, 23].

Dentro dos MDs, existem duas metodologias diferentes, o método das forças e o método dos deslocamentos, sendo que cada um tem como incógnitas do sistema, da equação (4), respetivamente, as forças aplicadas ou as deformações resultantes [21, 46].

$$\begin{Bmatrix} F_1 \\ F_1 \\ \vdots \\ F_1 \end{Bmatrix} = \begin{bmatrix} K_{11} & K_{11} & \cdots & K_{11} \\ K_{11} & K_{11} & \cdots & K_{11} \\ \vdots & \vdots & \ddots & \vdots \\ K_{11} & K_{11} & \cdots & K_{11} \end{bmatrix} = \begin{Bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{Bmatrix} \quad (4)$$

Métodos Variacionais (MV) - Nos ramos da matemática e da física, quando se tenta analisar um determinado problema na natureza, normalmente recorre-se a equações diferenciais que consigam descrever o respetivo fenómeno. Porém, fenómenos físicos podem também ser descritos em termos da minimização funcional da sua energia total. Este tipo de abordagem tem o nome de Teorema da Energia Potencial Mínima (TEPM), e é uma das vertentes dos MV. O MEF pode ser formulado a partir deste princípio, desde que o caso de estudo corresponda a um problema desta vertente. Isto leva a que, apesar de os MV serem amplamente utilizados, estes apresentam uma grande desvantagem, no sentido de não poderem ser aplicados em equações diferenciais que contenham termos da primeira derivada [47]. Um exemplo da aplicabilidade dos MV pode ser a análise de uma mola, sujeita à compressão como na Figura 21 [45], onde a deformação resultante deste elemento pode ser analisada através do equilíbrio das forças, ou a partir da análise da sua energia potencial.

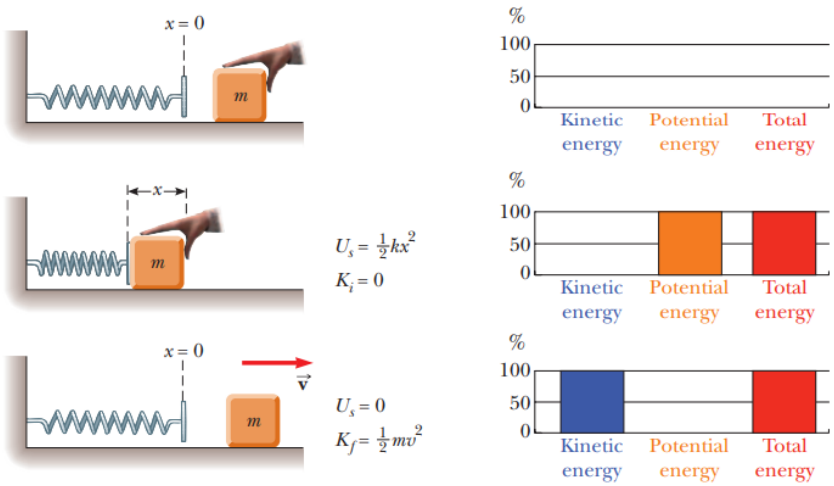


Figura 21 - Exemplo de aplicação do TEPM [45]

Para além do TEPM, dentro dos MV existe também o Princípio dos Deslocamentos Virtuais (PDV). Esta metodologia é muito versátil, pois pode ser aplicada a materiais com o comportamento linear ou não linear, e em simulações estáticas ou dinâmicas. Quando um corpo está sujeito a esforços internos e/ou externos, este tem a tendência em alterar a sua geometria, devido à instalação de deformações compatíveis com as condições de fronteira, a qual se dão o nome de deslocamentos virtuais. Estes deslocamentos são isentos de vazios e de sobreposições de material. Com isto, o PDV

estabelece que a energia de deformação virtual provocada pelas tensões internas é igual ao trabalho virtual causado pelas forças externas, ou seja, o trabalho interno do sistema é igual ao trabalho externo (equação(5)) [20, 22].

$$\text{Trabalho Interno} = \text{Trabalho Externo} \tag{5}$$

Método dos resíduos ponderados (MRP) **-** Quando os casos de estudo são muito complexos, e não é possível utilizar os MV devido à falta de informação sobre a energia potencial do sistema, utilizam-se então os MRP. Esta metodologia possui diversos tipos de formulações [21, 22], nomeadamente:

- Método de Galerkin;
- Método da Colocação;
- Método dos Menores Quadrados;
- Método do Subdomínio.

Apesar de serem mais utilizados quando não se possui informação sobre a energia do sistema, estes podem também ser usados nas mesmas condições que os MV e devolvem resultados idênticos [21]. Dentro destes quatro, o Método de Galerkin é o mais empregue, e este consiste em recorrer a uma função de aproximação, para estimar o valor da variável independente.

2.3.6 Tipos de elementos

2.3.6.1 Elementos 1D

Elementos como as treliças e as vigas, podem ter a sua representação tridimensional simplificada por apenas uma linha unidimensional (Figura 22). Sendo as secções retas omitidas, e introduzidas numa fase posterior do cálculo [22]. Quanto aos esforços, os elementos que constituem as treliças apenas são capazes de os suportar na direção axial, desde que se cumpram determinadas condições. As vigas, como já possuem algumas ligações rígidas capazes de transmitir os momentos, têm de ser capazes de suportar os mesmos. Em [23], esta informação foi organizada numa tabela, reproduzida na Figura 23.

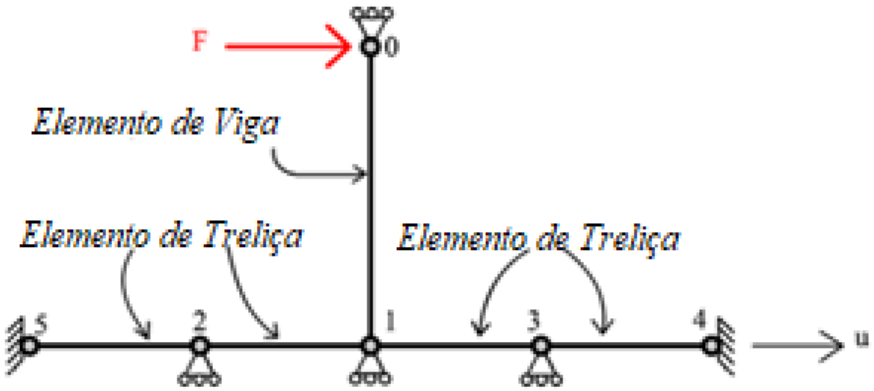


Figura 22- Representação de elementos 1D [48]

Outline			*Parte*		
Dimensão	*Estado de tensão prevalente*	*Secção Reta Variável*	*Correlação Força-Deslocamento*	*Exemplo*	No.
1	2	3	1	2	
1D	*Treliça*	*Não*	*Linear*		1
		Linear	*Logarítmico*		2
		Constante	*Linear*		3
	Viga	*Não*	*Cúbico*		4
	Viga à torçao	*Não*	*Linear*		5

Figura 23 - Características dos elementos 1D [23]

Com estes tipos de elementos, é possível construir estruturas complexas, desde que estas sejam reticuladas, contínuas ou articuladas, como a estrutura do caso de estudo apresentado em [49] (Figura 24).

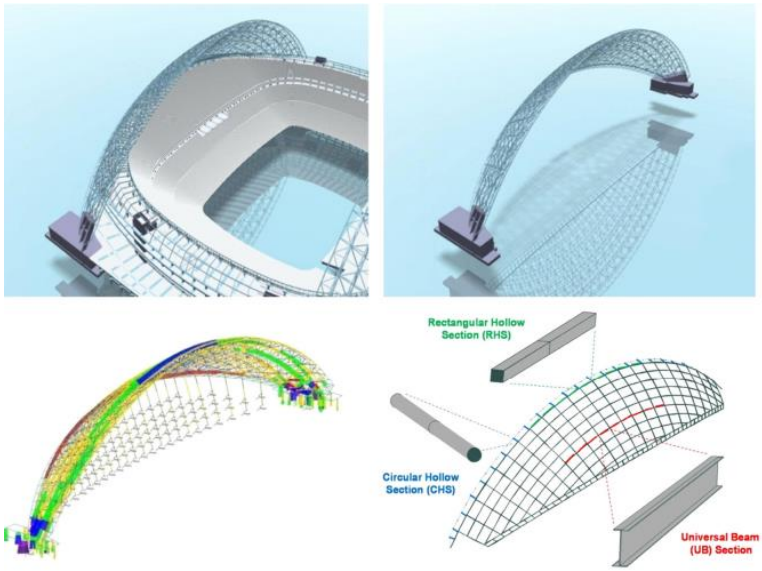


Figura 24 - Exemplo de uma estrutura treliçada [49]

2.3.6.2 Elementos 2D

Os modelos bidimensionais (Figura 25), são resultado da simplificação de um corpo tridimensional, através da supressão de uma das suas dimensões. Para que se possa proceder com este processo, o componente tem de ter uma dimensão maior que as outras [22]. O livro da referência [50], menciona que, tipicamente, nos casos em que esta é menor, ela deve ter no máximo 10% da menor distância existente no plano que se vai criar. Tal como os elementos 1D, a espessura vai ser apenas utilizada como parâmetro, no pré-processamento. Dentro deste tipo de elementos podem-se encontrar as membranas, as placas e as cascas.

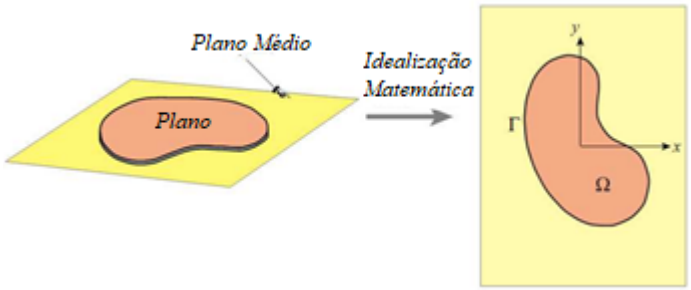


Figura 25 - Representação ideal de um elemento 2D [50]

Membranas - Os elementos de membrana são elementos bidimensionais, cujos esforços atuantes são aplicados apenas na direção do plano médio, todas as condições de fixação são simétricas, a espessura é constante ao longo do plano e tanto as tensões como deformações são consideradas uniformes por toda a área [50].

Placas - As placas são semelhantes às membranas, contudo ao contrário das membranas, apenas são capazes de suportar esforços transversos ao plano, o que resulta na sua flexão e rotação em dois eixos [22]. Quanto ao número de graus de liberdade, estas apenas possuem deformação na direção dos esforços, e dois níveis de rotação, por nó [50].

Cascas – Os elementos de casca, são como uma junção das membranas e das placas, pois, apesar de suportarem predominantemente esforços transversos de flexão, também são capazes de trabalhar com esforços de membrana (no plano) [22]. Para além disto, dentro dos elementos 2D, são os únicos capazes de curvarem no espaço, conferindo-lhes uma geometria pseudo-tridimensional ou 2.5D [50].

2.3.6.3 Elementos 3D

Os elementos tridimensionais (Figura 26) tal como o nome indica, possuem geometria em três dimensões, e são os elementos mais fidedignos à realidade. Contudo, em troca de uma maior fidelidade, estes elementos requerem um maior número de recursos computacionais, tornando as simulações mais pesadas e demoradas. De maneira a combater este problema, muitas vezes os elementos 3D são substituídos por 1D ou 2D, dependendo das características do modelo em questão [24].

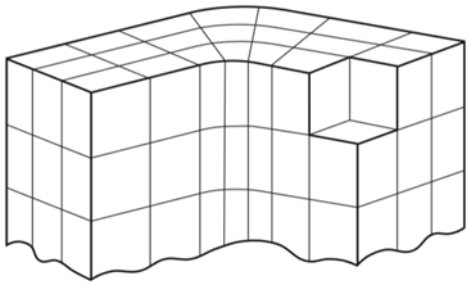


Figura 26 - Modelo tridimensional dividido por elementos 3D [24]

2.3.7 Redução dimensional

Na análise de um sistema através do MEF, muitas vezes a complexidade confrontada é muito elevada, o que pode levar a que o tempo de computação da solução seja demasiado alto, ou que o hardware não tenha capacidade de o resolver. Sendo assim, são empregues algumas técnicas de simplificação dos modelos, de maneira que haja uma redução na complexidade.

Muitas das técnicas empregadas na indústria, visam a redução das estruturas para uma ordem dimensional menor, como por exemplo de 3D para 2D ou de 2D para 1D, simplificando drasticamente a complexidade, mas de maneira que não seja comprometida a integridade do modelo [50].

2.3.7.1 Redução de 3D para 1D

Em diversos tipos de estruturas tridimensionais, podem existir elementos que não sofram esforços de corte ou qualquer tipo de momentos (Torsores ou Fletores), e apenas atuam esforços axiais, como acontece nas treliças e nas vigas axissimétricas. Neste tipo de elementos pode-se proceder com a simplificação tridimensional do modelo para uma linha unidirecional, sendo as propriedades relativas às secções das linhas introduzidas no pré-processamento [50].

Um exemplo de utilização desta metodologia, está presente no estudo [51], onde, de maneira a estudar-se a instabilidade da alma de um perfil em "I", quando são aplicados esforços nos banzos superiores (Figura 25), recorreram-se a duas metodologias. A primeira consistia em simplificar o modelo tridimensional num modelo de casca bidimensional, e a segunda num modelo de viga unidimensional. Quando analisadas as cargas críticas que levavam à instabilidade, observou-se que para larguras de viga suficientemente altas, o método unidimensional devolvia resultados semelhantes ao bidimensionais.

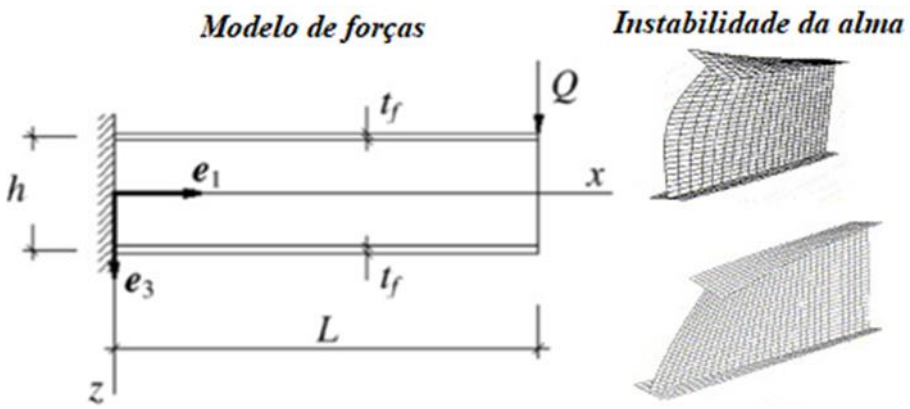


Figura 27 - Modelo de carregamento utilizado no estudo [51] para os dados da Tabela 4

Tabela 4 - Valores de carga critica obtidos por diferentes tipos de redução dimensional [51]

L (m)	$Q_{Crítico}$ (KN)		Diferença (%)
	Casca 2D	Viga 1D	
2.0	163.06	253.90	55.71
4.0	42.01	43.68	3.97
6.0	18.05	18.46	2.26
8.0	10.57	10.75	1.67
10.0	7.14	7.23	1.25

2.3.7.2 Redução de 3D para 2D

Neste tipo de simplificações, o que se faz é reduzir um elemento 3D para um 2D, através da remoção de um dos parâmetros da estrutura, sendo este introduzido como um *input* numa etapa mais tardia de cálculo. Normalmente essa propriedade é a espessura do material (Figura 28) [50].

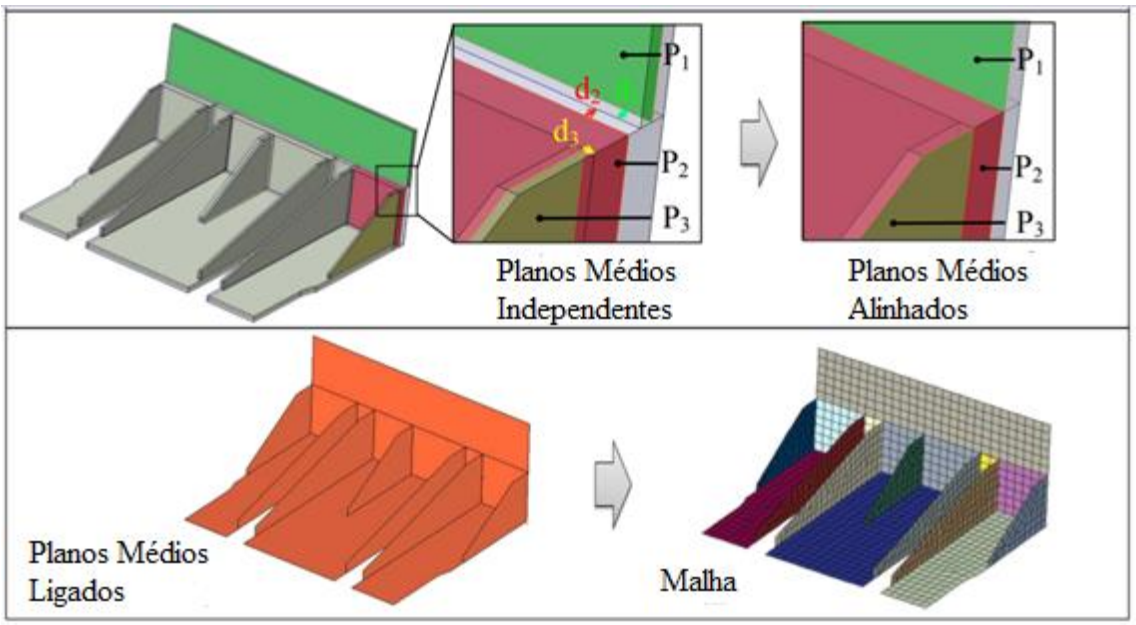


Figura 28 – Simplificação por planos médios [52]

Quanto aos diferentes tipos de redução 3D para 2D existem:

Estados planos de tensão - Quando um modelo possui uma dimensão muito inferior às outras duas, e a tensão na sua direção é nula. Pode-se omitir este parâmetro, originando assim um modelo 2D. Com este tipo de redução instala-se então um Estado Plano de Tensão, o que causa uma simplificação na matriz de cálculo que relaciona as tensões e as deformações de um corpo (equação (6)) [50, 53].

$$\begin{Bmatrix} \sigma_X \\ \sigma_y \\ \tau_{Xy} \end{Bmatrix} = \frac{E}{1-\nu^2} * \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} * \begin{Bmatrix} \varepsilon_X \\ \varepsilon_y \\ \varepsilon_{Xy} \end{Bmatrix} \tag{6}$$

Estados planos de deformação - Quando um modelo possui uma dimensão muito superior às outras duas, pode-se assumir que a deformação nessa direção é aproximadamente 0, e reduzir o elemento às duas dimensões de menor valor, originando assim um modelo 2D. Com este tipo de redução instala-se então um Estado Plano de Deformação, causando uma simplificação na matriz de cálculo que relaciona as tensões e as de formações de um corpo (equação (7)) [50, 53].

$$\begin{Bmatrix} \sigma_X \\ \sigma_y \\ \tau_{Xy} \end{Bmatrix} = \frac{E}{(1+\nu)(1-2\nu)} * \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} * \begin{Bmatrix} \varepsilon_X \\ \varepsilon_y \\ \varepsilon_{Xy} \end{Bmatrix} \tag{7}$$

Plano de axissimetria - Elementos tridimensionais, que possam ser formados através da revolução de um plano bidimensional, podem ser representados através desse mesmo plano, desde que as condições de fronteira aplicadas sejam também axissimetricas. No caso das forças aplicadas, estas já não precisam de o ser, pois podem ser representadas através de uma série de Fourier, onde cada termo é calculado em separado. Isto gera um Estado Plano de Axissimetria, levando a que a equação que correlaciona as deformações radiais com as tensões possa ser representada como na equação (8) [50],

$$\begin{Bmatrix} \sigma_r \\ \sigma_\theta \\ \sigma_z \\ \sigma_{zr} \end{Bmatrix} = \begin{bmatrix} (1-\nu)c & \nu c & \nu c & 0 \\ \nu c & 1-\nu & \nu c & 0 \\ \nu c & \nu c & (1-\nu)c & 0 \\ 0 & 0 & 0 & G \end{bmatrix} * \begin{Bmatrix} \varepsilon_r \\ \varepsilon_\theta \\ \varepsilon_z \\ \varepsilon_{zr} \end{Bmatrix} \tag{8}$$

onde:

$$c = \frac{E_y}{(1+\nu)*(1-2\nu)} \tag{9}$$

e

$$G = \frac{E_y}{2(1+\nu)} \tag{10}$$

ν - "Coeficiente de Poisson"

E_y - "Modulo de elasticidade"

2.3.7.3 Combinação de elementos

Em modelos complexos, não se está limitado ao uso de apenas uma metodologia de simplificação. No mesmo caso de estudo, a geometria pode ser composta tanto por elementos tridimensionais, bidimensionais e unidimensionais em simultâneo. Contudo, há que ter em atenção que elementos de níveis dimensionais diferentes possuem graus de liberdade diferentes, e, portanto, tem de se alterar a geometria de maneira que isto não seja um problema [50].

O artigo [48], procurou analisar os erros que podem ocorrer nas ligações entre elementos 3D com vigas 1D, e desenvolver metodologias de fixação para os corrigir. Na Figura 29, pode-se ver um destes casos, e neste tipo de situações tem de se ter em atenção o facto de a viga possuir seis graus de liberdade, enquanto que os elementos tridimensionais sólidos em que esta se inserem apenas possuem três, o que resulta numa ligação articulada. Uma das soluções apresentadas no artigo consiste em ligar "mini-vigas" à viga que se quer acoplar, isto fará com que seja evitada a formulação da rotula, e passe a haver congruência entre os graus de liberdade.

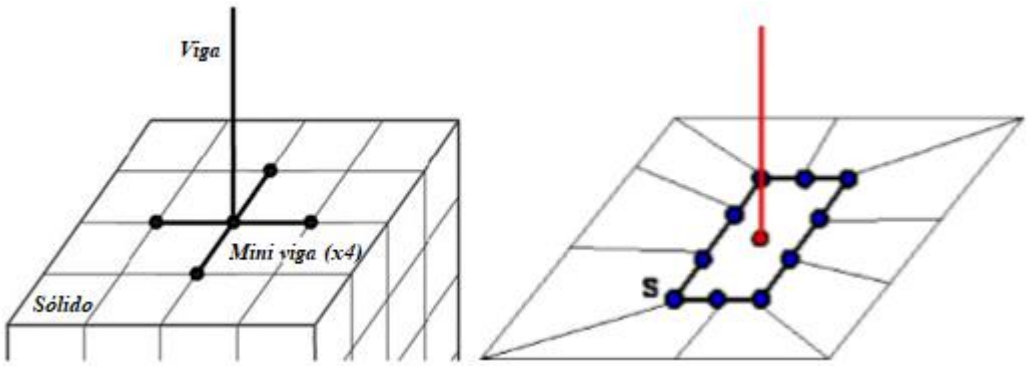


Figura 29 - Combinação de elementos 3D e 1D [48]

No caso da acoplação de elementos 2D a 1D, como no exemplo demonstrado na Figura 30, do livro [50], devido ao facto de a viga possuir 2 graus de liberdade de deformação e 1 de rotação, e a membrana apenas ter 2 graus de liberdade de deformação, para que as condições de fronteira sejam congruentes, a viga não pode ser colocada na fronteira da membrana, esta tem de ser estendida para o seu interior.

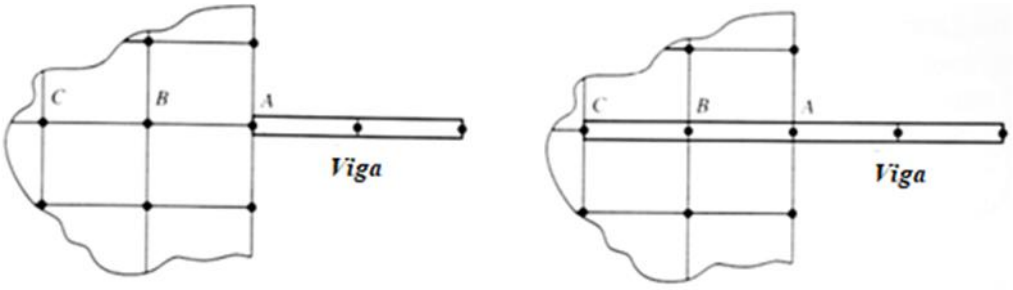


Figura 30 - Combinação de elementos 2D e 1D [50]

2.3.8 Formulação do MEF

Uma análise bem feita através do MEF deve seguir os três passos seguintes [21], independentemente da metodologia que foi selecionada para o caso de estudo:

1. Pré-processamento;
2. Obtenção da solução;
3. Pós-processamento.

2.3.8.1 Pré-Processamento

Esta primeira fase foca-se na discretização correta do caso de estudo. Ou seja, na divisão do modelo a estudar em diversos elementos de complexidade mais baixa. Neste processo deve-se então analisar que simplificações é que se podem fazer ao modelo original, de modo que o processo seja o mais simples e rápido possível, em temos computacionais, mas que não comprometa a geometria e leve a resultados muito diferentes dos reais. Um dos passos mais importantes é determinar que tipo de elemento se deve utilizar em cada componente do sistema, podendo este ser 3D, 2D ou 1D, e que dimensões/refinamento deve ser imposto à malha, tendo esta última de estar em conformidade com as condições de fronteira, carregamento e geometria da peça. Como esta é a fase mais importante do processo, Cook e Hutton [54, 55] referem os seguintes passos que devem ter de ser tidos em conta nesta etapa, e que estão mencionados no livro “Método de Elementos Finitos” [22]:

- “Definição do domínio geométrico do problema”,
- “Definição do tipo/tipos de elementos a utilizar > formulação de elemento”,
- “Definição das propriedades materiais dos elementos”,
- “Definição das propriedades geométricas dos elementos (comprimento, área)”,
- “Definição das conectividades entre elementos (malha do modelo)”,
- “Definição das restrições às variáveis de campo (condições de fronteira)”,
- “Definição dos carregamentos (forças, fluxos de calor, etc…)”.

Por sua vez, a formulação dos elementos possui as suas próprias fases, desta vez mencionadas por Bhatti e Rao [23, 56]:

Fase 0 – Definir as variáveis de campo necessárias para o caso de estudo;

Fase 1 – Criar os vetores que conferem a formas dos elementos, localização e as variáveis de cada nó:

$$a_e \; e \; A_e = Vetor \; de \; variáveis \; nodais$$

$$f_e \; e \; F_e = Vetor \; de \; forças \; nodais$$

Fase 2 – Através das variáveis nodais a_e e das funções de interpolação N, mencionadas no capítulo 2.3.1, exprimir os campos de deslocamentos no interior de elemento u_e.

$$u_e = Na_e \tag{11}$$

Fase 3 - Definir o vetor de deformação ε, através da relação entre a matriz de deformação B e as variáveis nodais a_e.

$$\varepsilon_e = Ba_e \tag{12}$$

Fase 4 – Definir o vetor de tensões σ, através da relação entre a matriz de elasticidade D_m e ε.

$$\sigma_e = D_m \varepsilon_e \tag{13}$$

Fase 5 -Desenvolver a matriz de rigidez do elemento k_e e o vetor de forças nodais f_e.

$$k_e = \int_V B^T D_m B v \tag{14}$$

$$f_e = k_e a_e \tag{15}$$

2.3.8.2 Obtenção da solução

No caso de análises lineares, nesta fase procede-se com síntese das equações algébricas que regem o sistema a ser estudado, podendo este ser escrito na forma matricial, ou como um sistema de equações (equação (16)). Resolvendo isto obtêm-se as soluções para as variáveis de campo locais a_e, ou as globais A_e se se proceder com a transformação de coordenadas [22].

$$K_e a_e = f_e \Leftrightarrow \begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1n} \\ K_{21} & K_{22} & \cdots & K_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ K_{m1} & K_{m2} & \cdots & K_{mn} \end{bmatrix} * \begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{Bmatrix} \tag{16}$$

Após a transformação de coordenadas, a totalidade de elementos fica,

$$KA = R + F \tag{17}$$

Onde

- K representa a matriz de rigidez global,
- A o vetor global das variáveis de campo,
- R as reações na estrutura
- F as forças nodais.

2.3.8.3 Pós-processamento

Após obtidos os resultados da fase anterior, estes são analisados pelo utilizador, para averiguar a veracidade dos mesmos e se nenhum erro foi cometido nalguma das fases anteriores. Atualmente a maioria dos softwares de análise possuem ferramentas de visualização gráfica de resultados, onde é mostrado como a peça deforma e onde se situam as tensões máximas do sistema. Para além disto, também já têm a capacidade de informar automaticamente o operador de erros existentes e, em alguns casos, repará-los [22].

2.4 Software de simulação

2.4.1 Softwares de simulação existentes

Hoje em dia existem diversos programas capazes de realizar análises pelo MEF, que são utilizados em diversas indústrias, como uma forma de minimizar os custos associados na análise dos diferentes tipos de componentes que cada empresa produz. Efetivamente há dois tipos de programas, aqueles que possuem uma interface gráfica, permitindo ao utilizador visualizar um modelo e interagir com ele, sem que tenha de entrar em contacto com o algoritmo de cálculo, e programas que se focam mais no código matemático da simulação, e cujas bases gráficas são limitadas, ou inexistentes [21]. Entre estes dois, os que têm capacidades gráficas são os mais utilizados pois permitem que o utilizador não necessite de ter conhecimento da parte do software ou do hardware, e apenas tenha do respetivo ramo a que o problema se insere. Para além disto as representações gráficas dos resultados e dos processos são ótimas para a documentação da análise, e torna-se mais fácil a sua demonstração a um cliente. É também importante de mencionar que os preços associados às licenças são mais elevados, podendo apenas uma rondar os 50000 USD, como no caso do Simulia Abaqus®.

Algumas das funcionalidades que estão presentes em todos os softwares gráficos são [22]:

- A capacidade de produzir as peças de raiz, através de funções de desenho e modelação, em diferentes dimensões;
- A aptidão para fazer sistemas de diversas peças;
- Ferramentas para analisar o tipo de contacto entre peças, podendo estas ser automáticas ou manuais;
- Definir e atribuir diferentes tipos de materiais a diferentes peças;
- Parametrização da malha, podendo ser selecionados o seu tipo, tamanho e algoritmo de geração;
- Definir as condições de fronteira do sistema (esforços, deslocamentos e fixações);
- Selecionar o tipo de análise;
- Interface gráfica para análise dos resultados obtidos e ferramentas de resolução de problemas.

Os softwares algorítmicos necessitam de um maior conhecimento por parte do utilizador nos campos mencionados, mas estes permitem desempenhar estudos mais complexos, devido ao facto de o operador ter um maior controlo de como a simulação

é feita, mas também por poder recorrer a recursos externos, como por exemplo a bases de dados, envio automático de emails, e a retirar ou a escrever informação diretamente de folhas em Excel [23]. Atualmente já existem alguns que permitem ter as duas funcionalidades, como o Simulia Abaqus® e o Ansys®, que, para além de serem gráficos, permitem ao operador alterar os parâmetros da simulação e criar *scripts*, sendo que cada um utiliza as linguagens, como o Python ou C/C++. Dentro destes três tipos de softwares tem-se os seguintes exemplos:

- **Softwares gráficos:** Solidworks®, Catia® e o Fusion 360®;
- **Softwares gráficos com capacidade de *scripting*:** Abaqus® e o Ansys®;
- **Softwares algorítmicos:** Mathematica® e o Maple®.

Nos últimos anos, com o avanço da inteligência artificial (AI) têm vindos a ser feitos estudos na integração desta nova ferramenta no MEF. Um exemplo disto é um estudo feito pela IEEE [57], onde utilizaram um modelo de *Machine Learning* (ML) para otimizar o design de um conjunto de chips, através de diversas análises do MEF.

2.4.2 Princípios de automatização

Como foi mencionado no capítulo 2.4.1, já existem softwares gráficos que permitem ao utilizador manipular o código base da simulação, e até mesmo formular *scripts*. Ora, isto é possível pois algumas das linguagens de programação já possuem livrarias que permitem aceder aos comandos gráficos e analíticos dos respetivos softwares. Usando a interação entre o Abaqus® e o Python como exemplo, pode-se ver na Figura 31 algumas das livrarias existentes. Dentro destas, as mais importantes são a "*abaqus*"e a "*abaqusConstants*". A primeira, cria referências a todos os objetos públicos do Abaqus®, tornando-os acessíveis ao código. Isto quer dizer que a partir do Python, é possível ativar os diferentes comandos do Abaqus® que estão presentes na parte gráfica. A segunda permite acesso às constantes presentes no software, como valores de Pi e do número de Neper [58]. O resto dá acesso a certas funcionalidades mais específicas dentro de cada uma das áreas presentes [59].

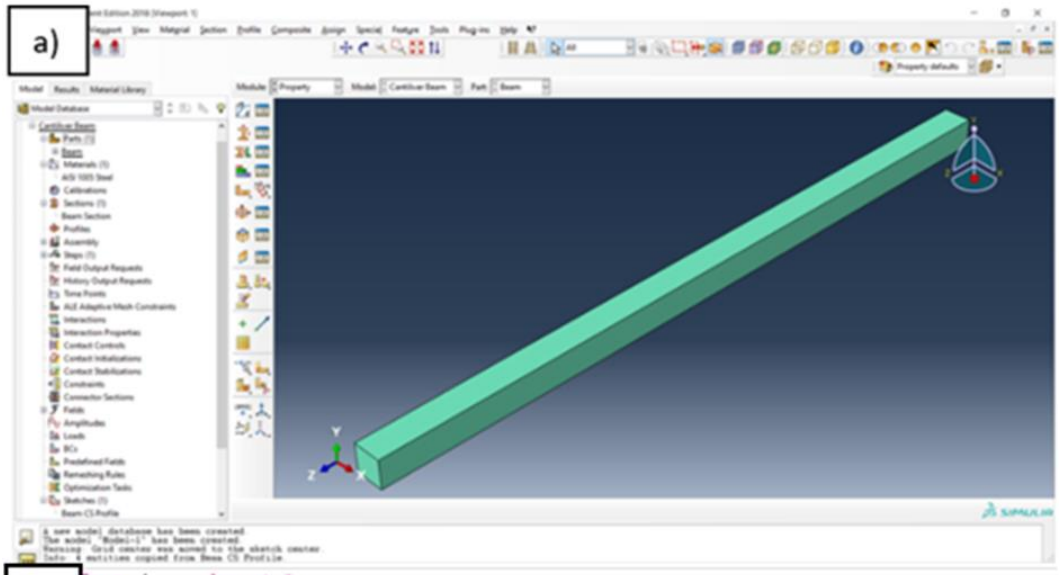
```
# Livrarias de Interação do Python com o Abaqus
from abaqus import *
from abaqusConstants import *
import part
import material
import assembly
import step
import interaction
import load
import mesh
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
```

Figura 31 - Livrarias de acesso às funcionalidades do Abaqus® no Python [58, 59]

Com este sistema é então possível fazer análises MEF, sem nunca interagir diretamente com o Simulia Abaqus®. Na Figura 32, pode-se ver o exemplo de um modelo de uma viga em aço AISI 1005, feita através de um script em Python.

Esta funcionalidade tem permitido aos utilizadores desempenhar diversos tipos de estudos que, ou não seriam possíveis ou seriam muito demorados por meios normais. No artigo [60] utilizou-se esta capacidade de *scripting* do Abaqus® para analisar a geometria de diversos tipos de reforços de materiais diferentes. Como era necessário realizar várias simulações, em que em todas elas se alteravam as dimensões do reforço, a equipa responsável, em vez de fazer o estudo por meios convencionais, desenvolveu um algoritmo que faz um ciclo de uma simulação base, e em cada iteração alterava os valores das medidas dos frisos, tendo-se obtido os resultados pretendidos, mais facilmente e de uma maneira mais rápida.

a)



b)

```
from abaqus import *
from abaqusConstants import *
import regionToolset

# Criar o Viewport (Universo onde vai ocurrer a simulação)
session.viewports['Viewport: 1 '].setValues(displayObject=None)

# Criar um modelo de peça na árvore do abaqus
mdb.models.changeKey(fromName='Model-1', toName = 'Cantiliver Beam')
beamModel = mdb.models ['Cantiliver Beam']

# Criar a peça
import sketch
import part

#a) Desenhar a secção reta da viga, através da ferramenta de retângulos
beamProfileSketch = beamModel.ConstrainedSketch(name = 'Beam CS Profile', sheetSize = 5)
beamProfileSketch.retangle(point1 = (0.1,0.1),point2 = (0.3,-0.1))

#b) Extrudir a secção reta e conferir-lhe propriedades
beamPart = beamModel.Part(name='Beam', dimensionality = THREE_D, type=DEFORMABLE_BODY)
beamPart.BaseSolidExtrude(sketch=beamProfileSketch, depth=5)

# Criar o material
import material

# Criar o material Aço AISI 1005
# Modulo de Elasticidade e Coeficiente de Poisson
beamMaterial = beamModel.Material(name='AISI 1005 Steel')
beamMaterial.Density(table=(7872,),)

# Criar uma secção sólida e passá-la para a viga
beamSection = beamModel.HomogeneousSolidSection(name = 'Beam Section', material = 'A')
```

Figura 32 - Desenvolvimento de uma viga a) Versão gráfica b) Código Python (Fonte própria)

DESENVOLVIMENTO

3 DESENVOLVIMENTO

3.1 Contextualização e requisitos do problema

3.1.1 Apresentação do problema

No que toca ao desenvolvimento de componentes para interiores de veículos, a Simoldes Plásticos® está presente em todas as fases de conceptualização e produção das peças que pretende produzir, portanto, quando esta projeta uma peça, é necessário saber quais são as melhores características que o componente deve possuir para desempenhar a função que é pretendida. Uma característica que quase todas as peças que a Simoldes Plásticos® produz têm, é a existência de frisos que servem de reforço para o componente em questão (Figura 33). Estes reforços são muito importantes, porque permitem aumentar a rigidez de um determinado componente, através de pequenos incrementos no volume total. Apesar de os frisos serem uma parte de elevada importância para o desenvolvimento de um componente, a empresa não possui metodologias internas para os dimensionar, levando a que normalmente tenham de ser desenvolvidos componentes com determinadas propriedades de frisos, e depois através de processos de análise computacional e/ou manual, tenham de ter os seus parâmetros alterados de forma a satisfazer os requisitos mecânicos impostos pelo cliente.

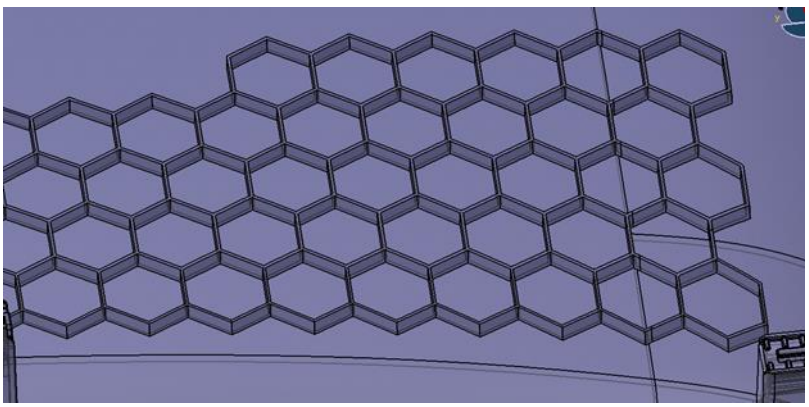


Figura 33 – Reforço de um componente através de frisos

3.1.2 Trabalho proposto

Face ao problema mencionado, o trabalho proposto pela Simoldes Plásticos®, consiste em desenvolver um programa em Python, que, quando executado no software de simulação Abaqus®, permite selecionar uma determinada peça e um intervalo de parâmetros para os frisos (altura, espessura, largura e área de aplicação). Com estes *inputs* do utilizador, o software deve então desenvolver diversas peças, com todas as combinações possíveis dos parâmetros dos frisos, executar dois ensaios mecânicos em cada uma delas (ensaio estático e *Ball drop*) e fazer o tratamento dos dados obtidos, de maneira a poder-se determinar qual é a melhor geometria para uma dada peça.

3.1.3 Organização da componente prática

Com o objetivo de se explicar o trabalho realizado, este capítulo de desenvolvimento divide-se em três partes, e procura abordar na primeira, o desenvolvimento do modelo da peça que se quer estudar, no software CATIA V5® e Abaqus®, bem como as simplificações feitas e o tratamento de malha realizado (capítulo 3.2 e 3.3). Na segunda parte aborda-se então as condições de simulação que devem ser impostas à peça, para que o estudo seja rápido e preciso na obtenção dos resultados (capítulo 3.4), e para terminar trata-se então do software desenvolvido, onde se explica o algoritmo desenvolvido e como este se traduz no código em Python (capítulo 3.5).

3.2 Modelo da peça e tratamento da geometria

Com o desenvolvimento do software, a Simoldes® pretende que seja analisada um componente de porta da bagageira de um dos seus clientes. Este componente é o representado na Figura 34, e como pode ser visto, esta peça possui frisos ao longo da sua superfície, que lhe conferem rigidez. Como o software tem de realizar diversos ensaios, para validar a peça, o modelo tridimensional apresentado teve de ter a sua complexidade reduzida.

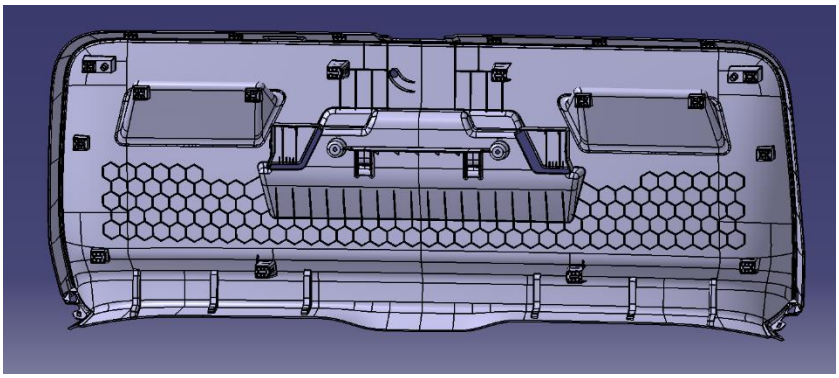


Figura 34 – Componente a analisar

A peça em questão possui uma largura de vão de aproximadamente 1200 mm, e uma espessura máxima de 3 mm. Como a espessura é inferior a 10% da maior dimensão da peça, o modelo tridimensional pode ser simplificado através das suas superfícies médias (Figura 35) [50]. Isto reduz consideravelmente o tempo de análise, contudo é possível simplificar ainda mais o modelo. Como também se pode ver na Figura 35, algumas das características geométricas da peça, como furos e extrusões, foram removidos do modelo. Isto foi feito, pois estes elementos, apesar de serem necessários para o funcionamento da peça, não influenciam o seu desempenho mecânico. Para além disto, foram removidos os frisos, para mais tarde serem desenhados pelo software, com diferentes parâmetros.

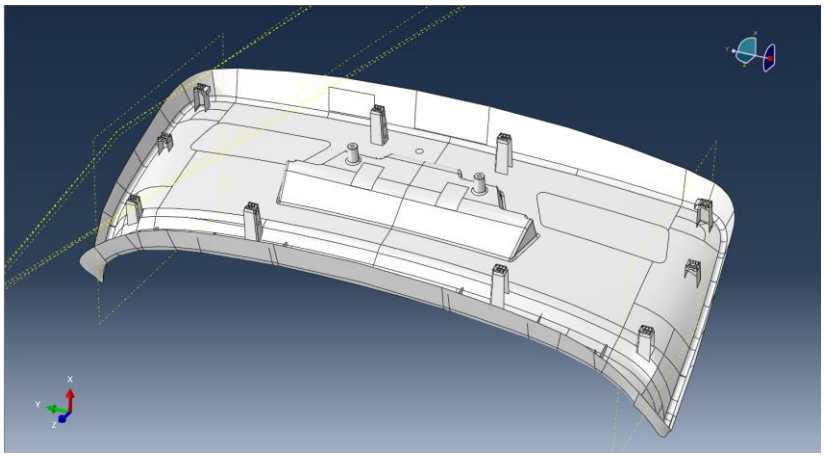


Figura 35 - Modelo simplificado da peça

Com a geometria da peça tratada, analisou-se também qual seria o ponto mais crítico para aplicação da carga. Segundo o caderno de encargos, as cargas devem ser aplicadas nas zonas de reforço da peça, na face contrária onde estes são colocados. Sendo assim, escolheram-se 3 zonas diferentes onde se podiam aplicar a carga e compararam-se os resultados entre elas, sendo estas zonas são as que estão representadas na Figura 36. Na Figura 37 à Figura 45 e na Tabela 5, encontram-se os

resultados obtidos para cada ensaio. Os parâmetros do material utilizados nestes ensaios, foram os determinados no capítulo 3.5.3.8, para o ensaio estático.

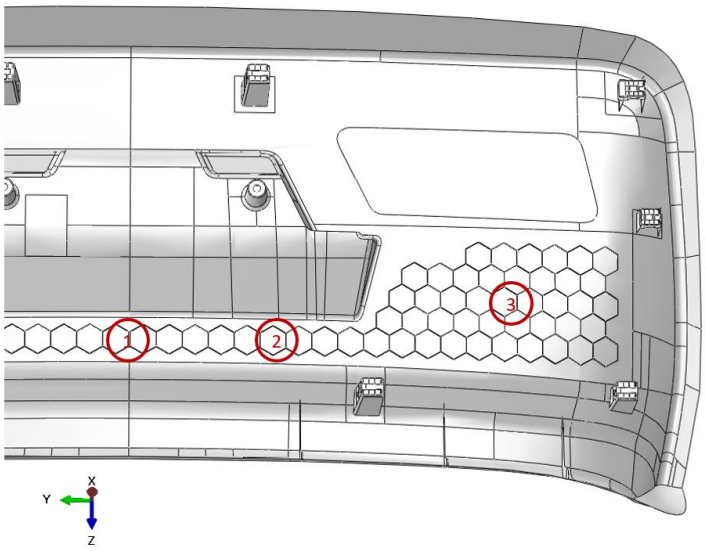


Figura 36 - Pontos de Aplicação da carga

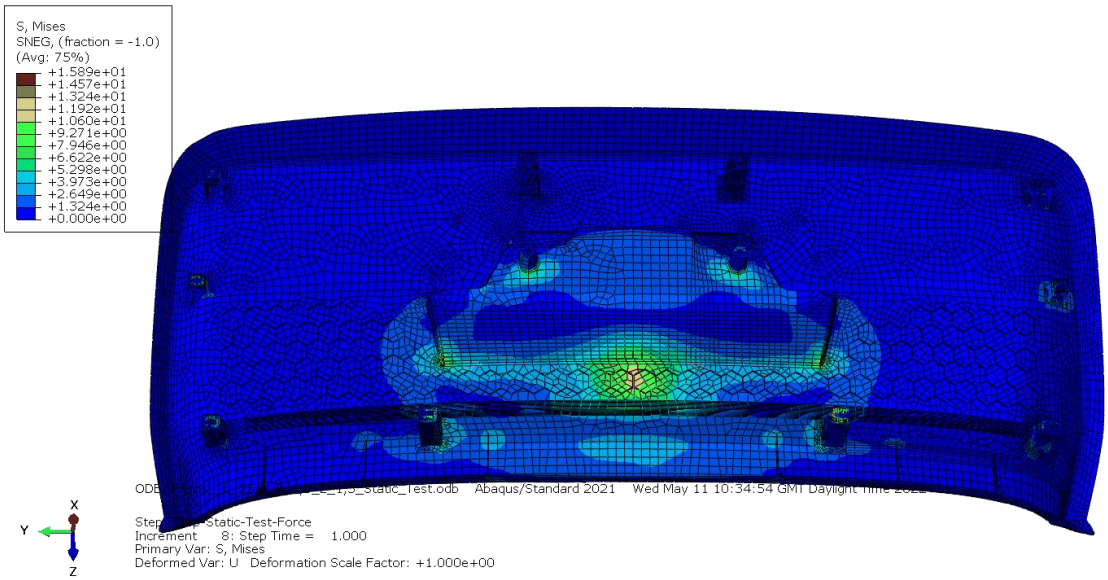


Figura 37 – Resultados de Tensão – Ponto 1

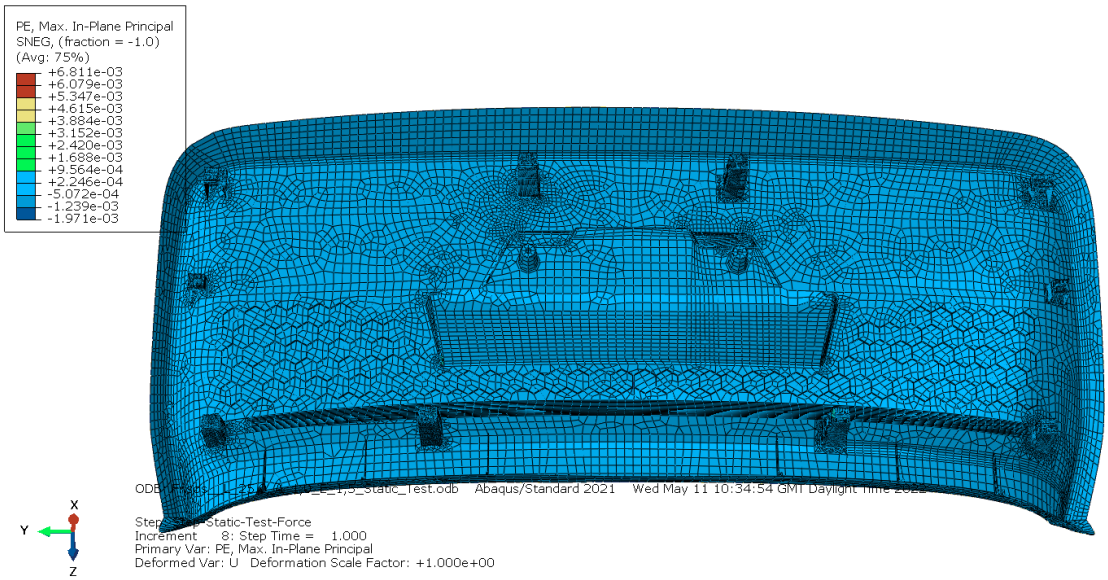


Figura 38 - Resultados da Deformação Plástica – Ponto 1

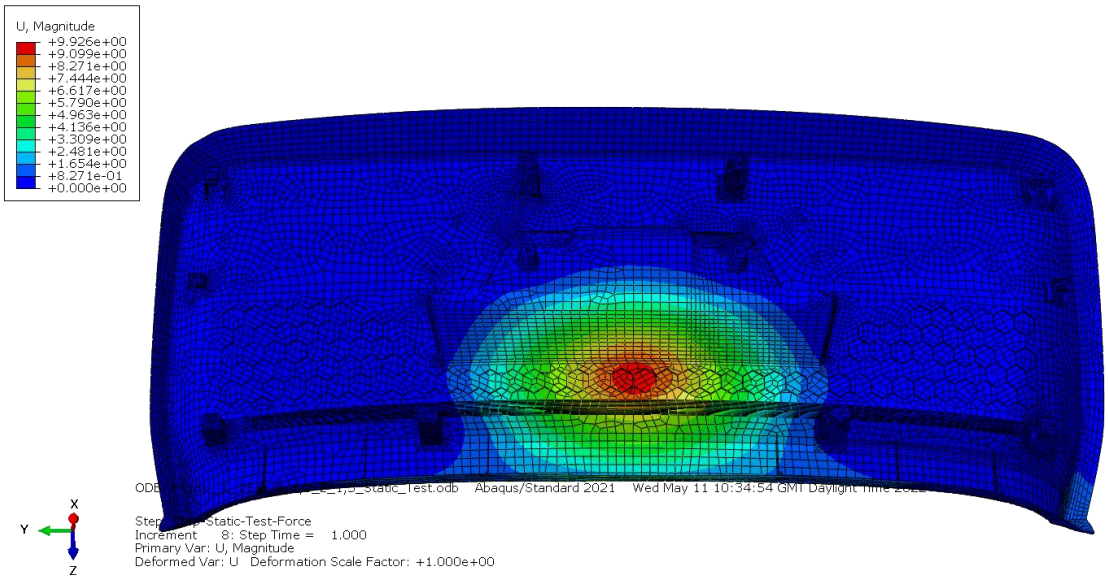


Figura 39 - Resultados do Deslocamento– Ponto 1

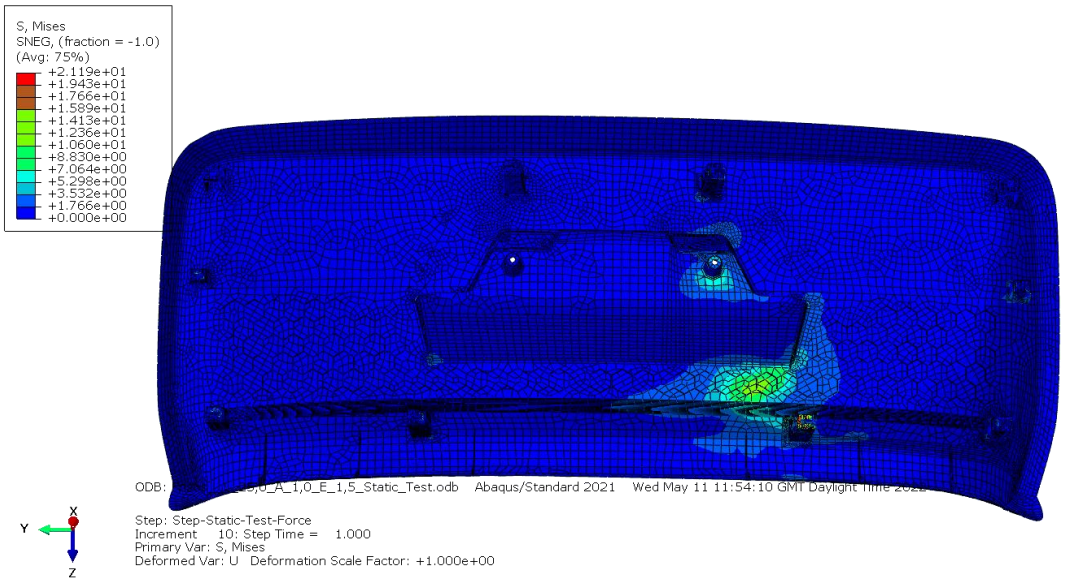


Figura 40 – Resultados de Tensão – Ponto 2

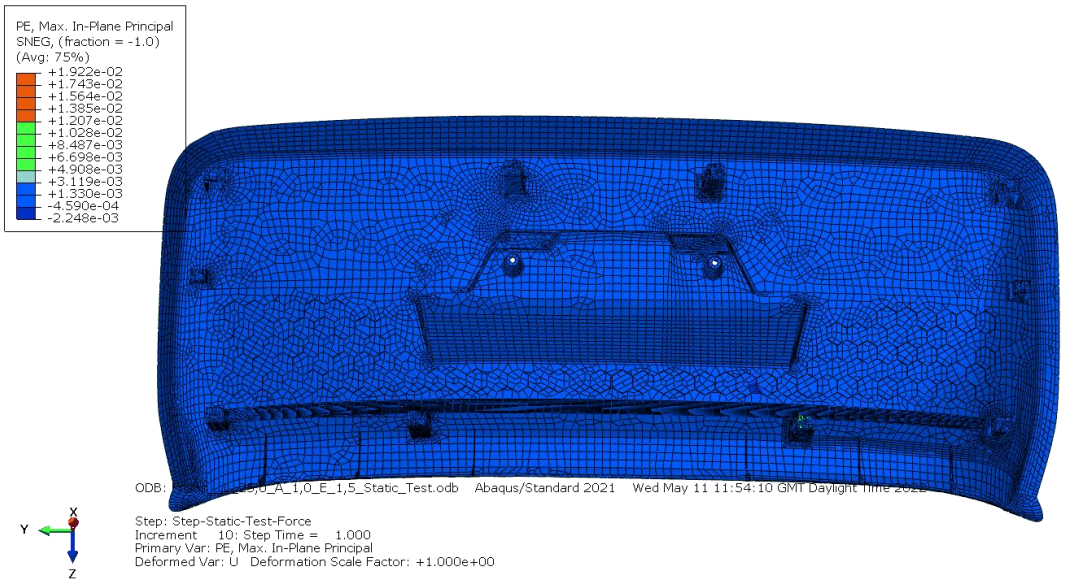


Figura 41 - Resultados da Deformação Plástica – Ponto 2

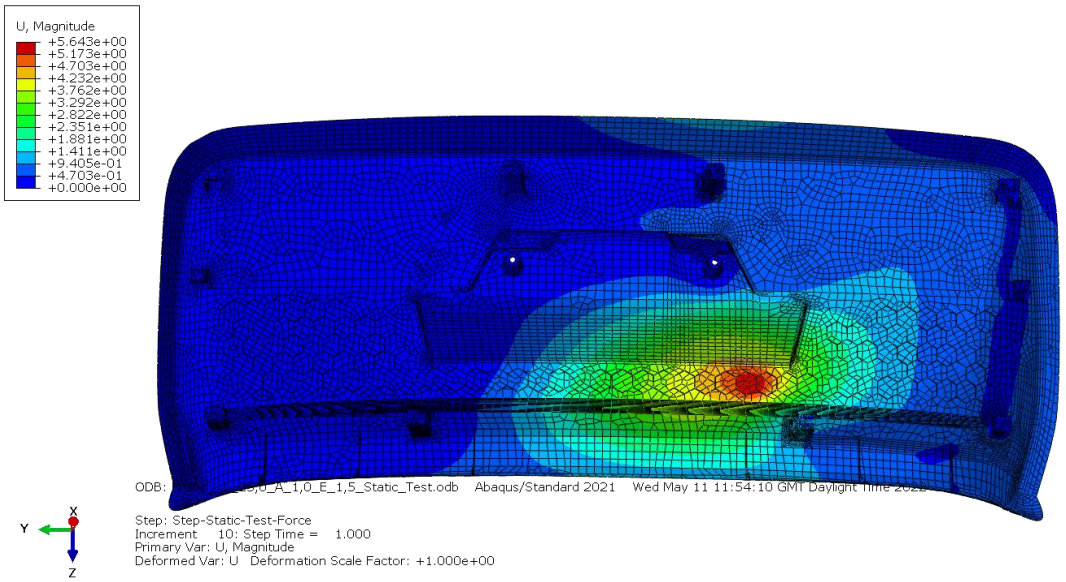


Figura 42 - Resultados do Deslocamento– Ponto 2

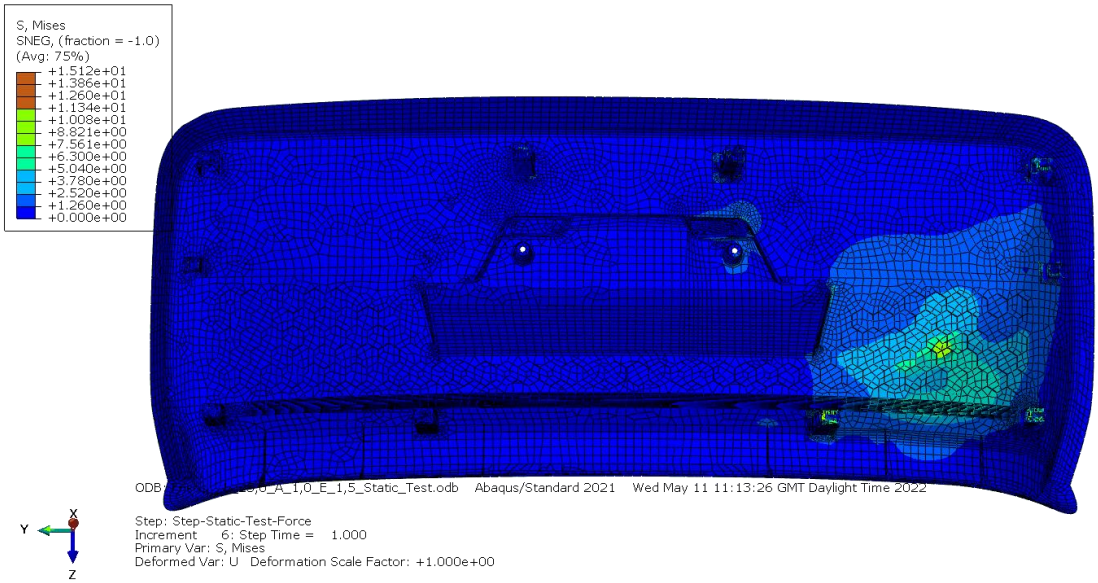


Figura 43 - Resultados do Deslocamento– Ponto 3

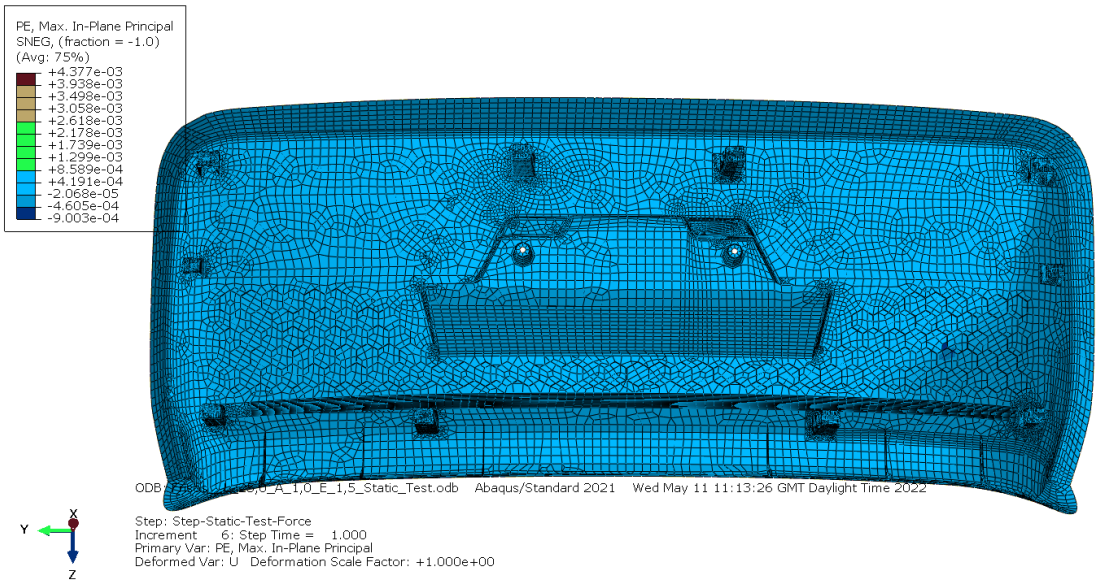


Figura 44 -- Resultados da Deformação Plástica – Ponto 3

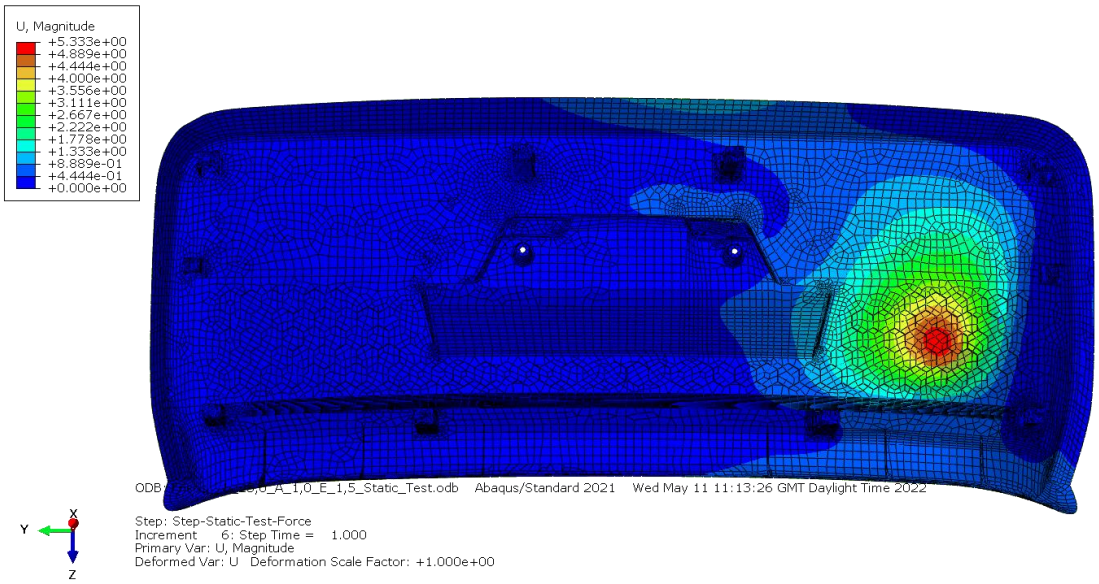


Figura 45 - Resultados do Deslocamento– Ponto 3

Tabela 5 – Resultados do estudo dos 3 pontos

	Tensão Equivalente de Von Mises (MPa)	Deformação Plástica (%)	Deslocamento (mm)
Ponto 1	15.89	0.68%	9.93
Ponto 2	21.19	1.92%	5.64
Ponto 3	15.12	0.44%	5.33

Observando-se os resultados obtidos na Tabela 5, no que toca à deformação plástica e à tensão equivalente de Von Mises, o ponto 2 é o mais crítico, porém como a concentração de tensões se encontra num dos porta agrafos da peça, e o seu desempenho mecânico não é influenciado pelos frisos da peça base, optou-se por estudar o ponto 1, já que este é mais severo para a peça base.

Como o ponto de aplicação da carga se encontra sobre o plano de simetria da peça, e tanto a geometria como as condições de fronteira são simétricas, pode simplificar-se o modelo de cálculo, utilizando-se apenas metade da peça (Figura 46).

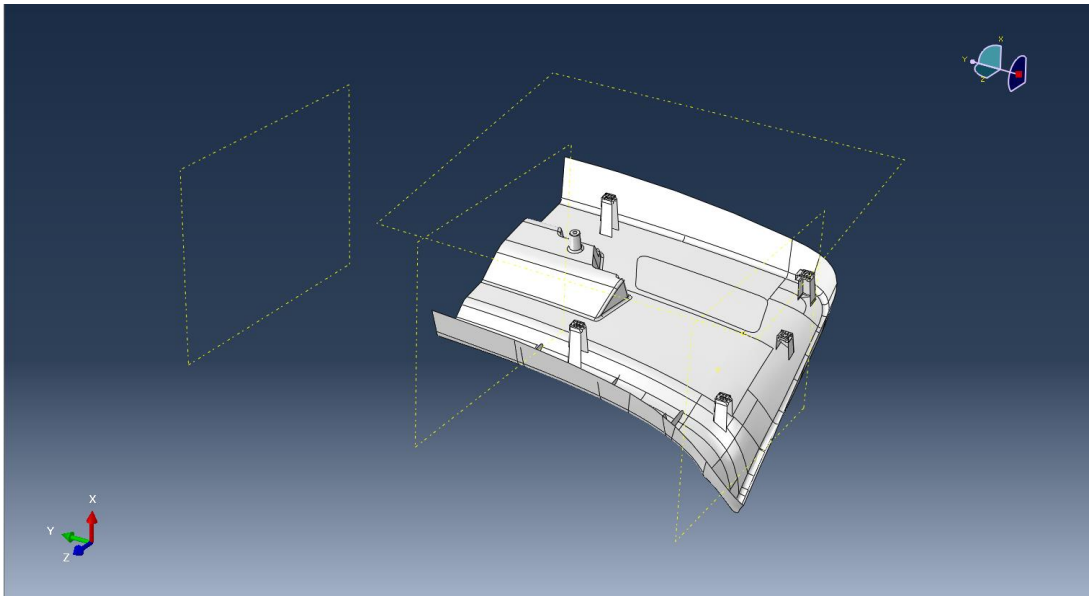


Figura 46 - Modelo simplificado da peça (Simetria)

3.3 Malha da peça de base

Quando uma peça que já tem a sua malha devidamente tratada, sofre alterações na sua geometria, o Abaqus® não apaga por completo todo o tratamento de malha que foi previamente realizado, apenas apaga o tratamento da zona que sofreu a alteração. Este processo pode ser evidenciado no provete da Figura 47, que, previamente a ter a sua geometria alterada, tinha a sua malha definida como estruturada, porém após a alteração, a malha das extremidades continua com os mesmos parâmetros, mas a da zona alterada passou a ser do tipo *Sweep*. Ora, como a peça base apenas tem a sua geometria a ser alterada numa única face, todas as outras partes do modelo podem ter a sua malha previamente tratada, pois o Abaqus® não vai apagar estas predefinições. Com isto em mente fez-se então o tratamento do modelo da peça base, para que a malha fosse o mais organizada possível.

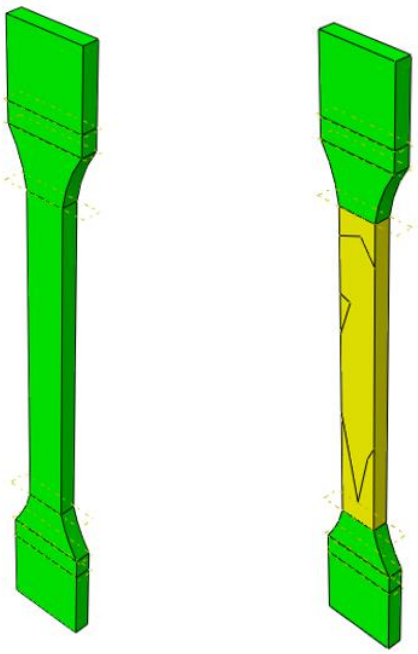


Figura 47 - Alteração do tratamento de malha

Para tornar a malha o mais correta possível, empregaram-se diferentes tipos de algoritmos de estruturação de malha, nos locais onde é mais correta a sua utilização. Os algoritmos existentes dentro do Abaqus® são os seguintes [61]:

- *Structured* – aplica padrões de malha já estabelecidos para geometrias mais simples. Nos modelos mais complexos, a geometria tem de ser particionada de forma a que a zona a malhar fique subdividida em secções de geometria mais simples, para o qual o Abaqus® já possua os padrões pré-definidos. Elementos com esta estrutura são representados a verde no Abaqus®;

- *Sweep* – o algoritmo *Sweep*, extrude uma malha gerada internamente, segundo um eixo de extrusão, ou de revolução. Tal como na *Structured*, este algoritmo está limitado a certas geometrias simples, pelo que em modelos complexos, este tem de ser repartido. Elementos com esta estrutura são representados a amarelo no Abaqus®;
- *Free Mesh* – a *Free Mesh*, é o algoritmo mais flexível para a geração de malha, pois não é baseado em padrões pré-estabelecidos, contudo o tipo de malha gerado, normalmente não é o mais otimizado para a geometria em questão. Elementos com esta estrutura são representados a rosa no Abaqus®;
- *Bottom-up* – este algoritmo é utilizado para malhar regiões sólidas com malha hexaédrica, que são impossíveis, ou muito difíceis de malhar através de metodologias automáticas de *top-down meshing*. Elementos com esta estrutura são representados a laranja no Abaqus®;

Estas metodologias foram então empregues no modelo da peça em questão, de forma a originar uma malha mais direita e leve em termos computacionais. Este arranjo da malha pode ser visto na Figura 48 à Figura 55.

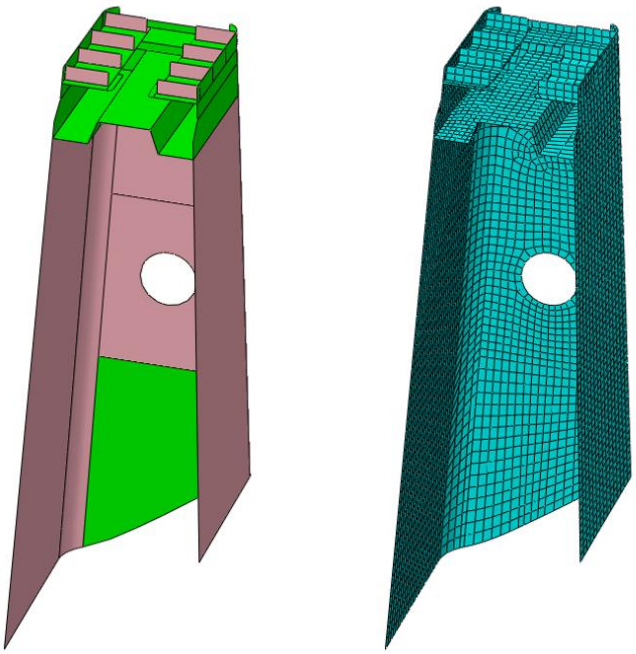


Figura 48 - Exemplo do tratamento de malha de um porta agrafos

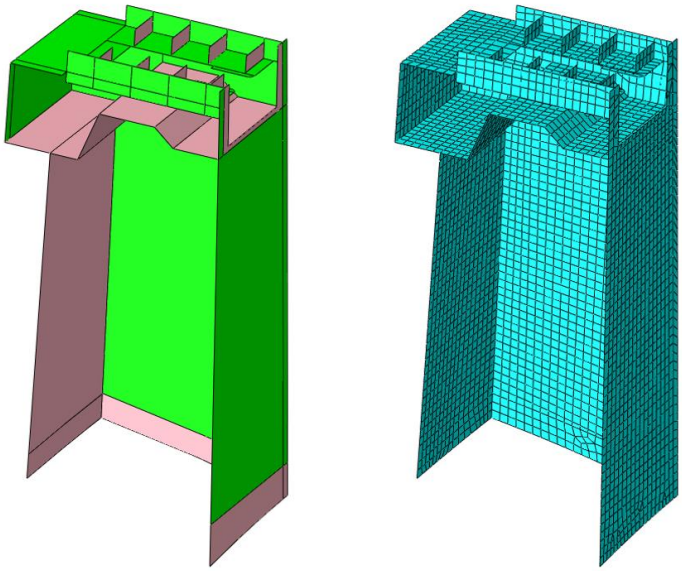


Figura 49 - Exemplo do tratamento de malha de um porta agrafos

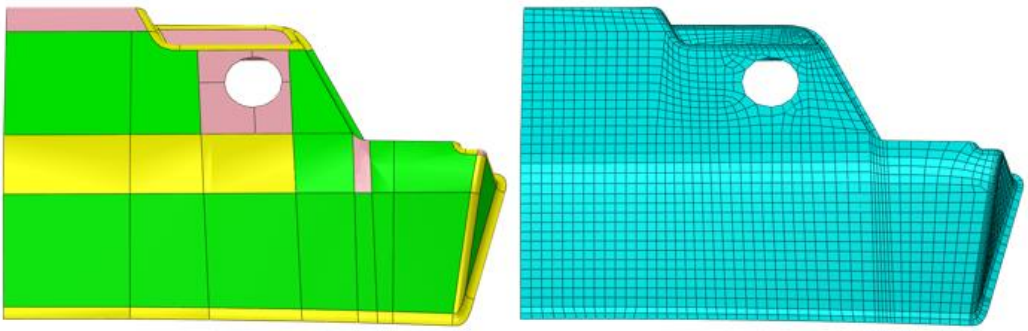


Figura 50 - Exemplo do tratamento de malha da saliência da base

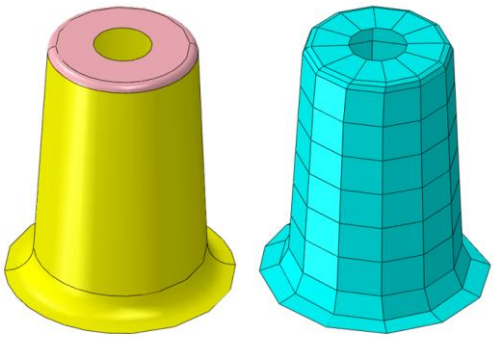


Figura 51 – Exemplo do tratamento da malha do alojamento de um parafuso

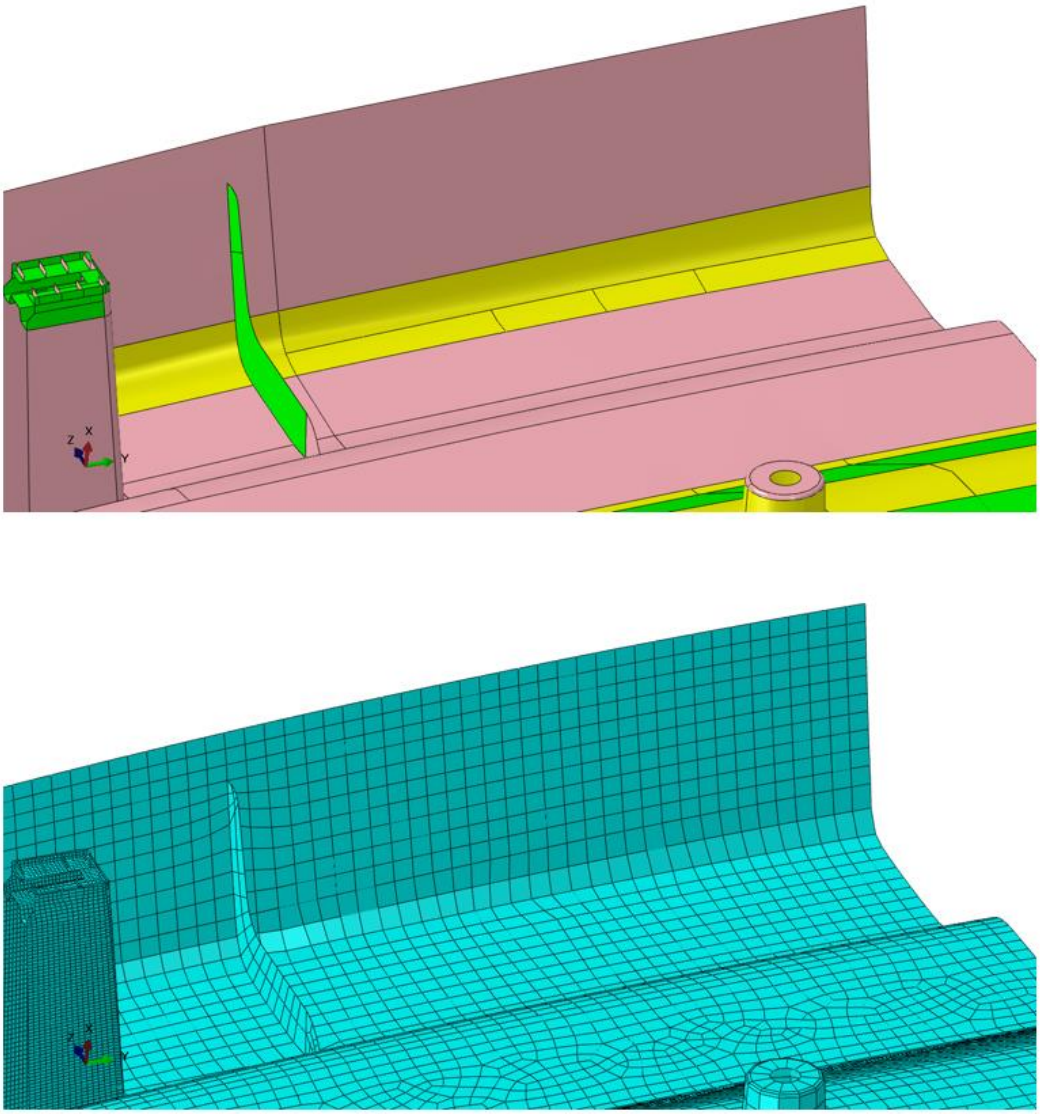


Figura 52 - Exemplo do tratamento da malha da aba frontal

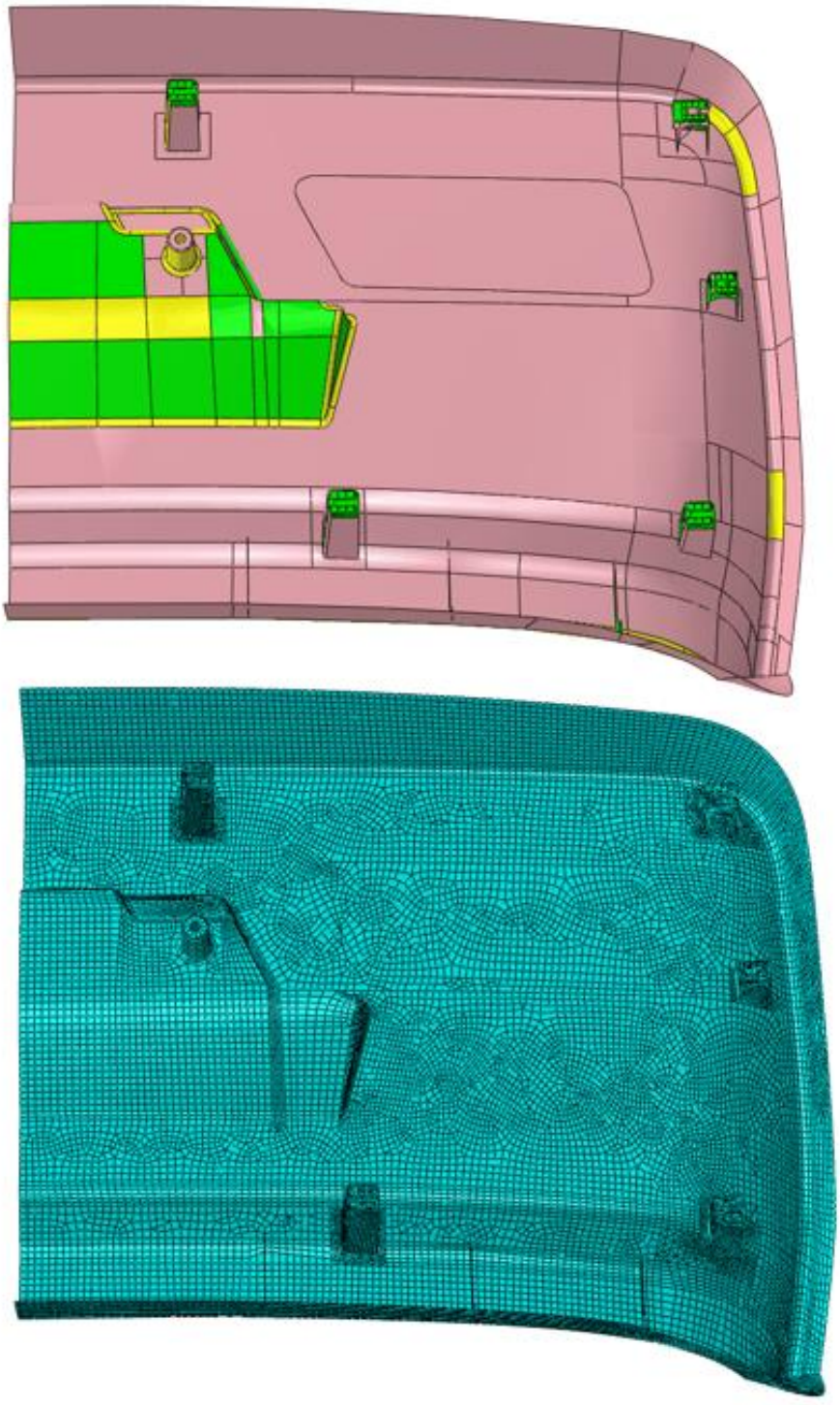


Figura 53 - Exemplo do tratamento da peça completa

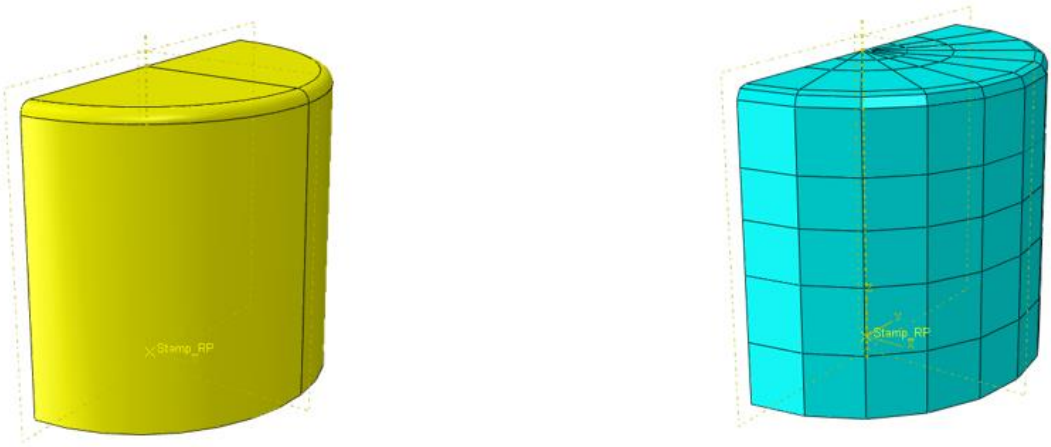


Figura 54– Exemplo do tratamento do impactor do ensaio estático

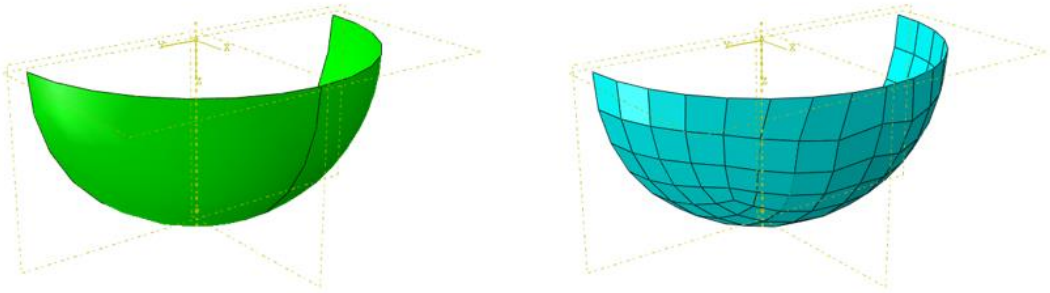


Figura 55 – Exemplo do tratamento do impactor do teste *Ball Drop*

Para além do algoritmo de malha a aplicar em cada parte da peça, é necessário também saber qual é o tamanho de malha que se deve utilizar. Para isto, realizou-se um teste de convergência de malha, no qual se utilizou o software desenvolvido nesta tese, e correu-se para uma mesma geometria de frisos, mas para parâmetros de malha diferentes, e analisou-se o deslocamento do impactor do ensaio estático. Os resultados obtidos estão representados na Figura 56, e aqui é possível presenciar-se que a partir dos 5mm de malha a variação dos resultados obtidos deixa de ser significativa, e, portanto, este foi o tamanho de malha que foi utilizado no estudo paramétrico da peça.

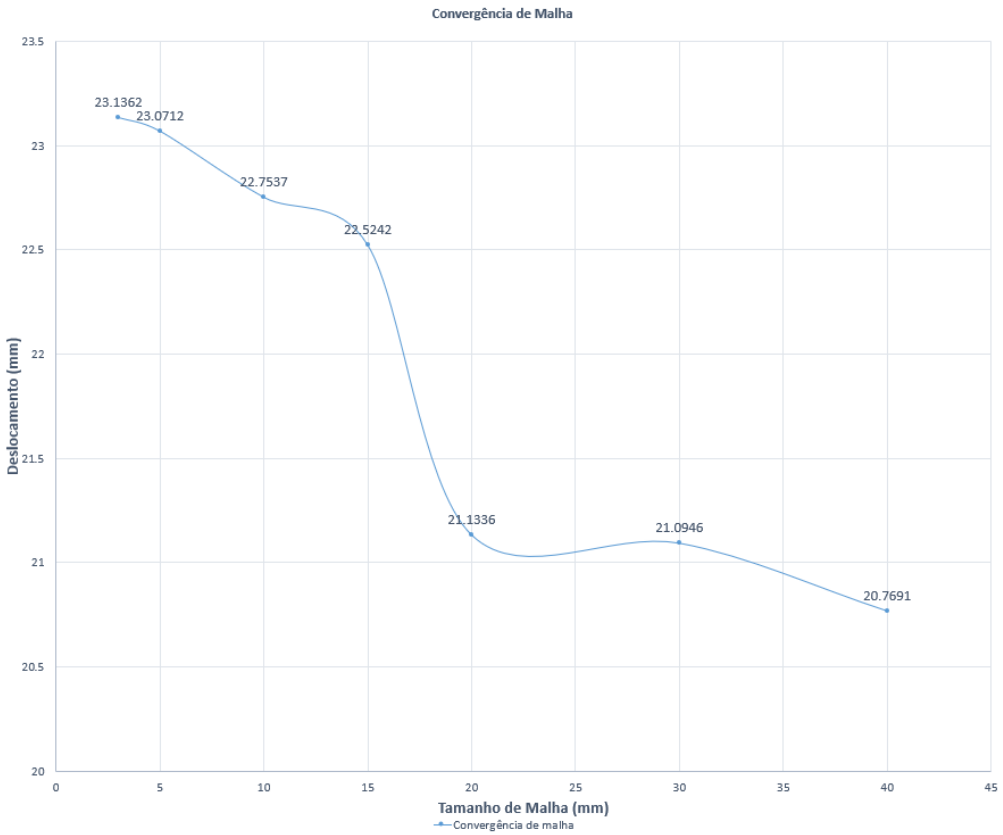


Figura 56 – Convergência do tamanho de malha do modelo

No que toca ao tipo de elemento a ser utilizado, como a espessura do modelo é muito inferior às restantes dimensões (inferior a 10%) [50], o elemento tem de ser do tipo de casca. Dentro deste conjunto, devido à complexidade da geometria utilizou-se o S4R quando possível, mas em determinadas partes do modelo teve de se utilizar o S3R. Preferiram-se estes tipos de elementos, pois são capazes de suportar esforços transversos de flexão e esforços no plano [22].

3.4 Testes de análise da peça

Para a validação do componente, o cliente impôs que fossem realizados os seguintes ensaios:

1. Ensaio estático – Consiste em aplicar uma força de 200N com um impactor cilíndrico, no qual a força demora 1s a chegar ao seu valor máximo, à temperatura de 23ºC, e, após a realização deste, a peça não pode ultrapassar a tensão máxima do material, e nem pode haver uma deformação permanente por parte do material.
2. *Ball Drop* – Consiste em largar uma esfera de 50 mm de diâmetro e massa de 500 g, a uma altura de 500 mm. Esta queda deve ser realizada a uma temperatura de 23ºC e não pode levar a que na peça base seja ultrapassada a tensão máxima do material, e não deve existir uma deformação permanente na peça.

Com estes dois ensaios, a melhor geometria de frisos possíveis, corresponde aquela que permite à peça passar nos dois ensaios, gastando-se a menor quantidade de material possível no seu desenvolvimento.

3.5 Desenvolvimento do software

3.5.1 Algoritmo geral

Em termos gerais de funcionamento, as funções que o software necessita de desempenhar são relativamente simples, pois, como este trabalha dentro do ambiente do Abaqus® e, portanto, tem acesso a todas as suas funcionalidades, o processamento sequencial das diferentes etapas é o mesmo, caso se estivesse a desempenhar as mesmas funções manualmente na interface gráfica do Abaqus®. Sendo assim, em termos gerais, o que o software necessita de fazer é essencialmente:

1. Abrir o ficheiro CAE que contém o modelo da peça que se pretende estudar;
2. Abrir uma instância de desenho;
3. Desenhar os frisos conforme os parâmetros introduzidos;
4. Extrudir os frisos;
5. Impor o material e as respetivas secções;
6. Criar a malha;
7. Impor as condições de ensaio;
8. Realizar a respetiva simulação;
9. Extrair os resultados.

Este processo deve ser repetido para todas as geometrias dos frisos que se pretendem estudar, e com os resultados obtidos, pode-se então chegar à conclusão de qual delas é a melhor para a respetiva peça.

Uma característica muito importante do algoritmo geral, é o facto de quando é gerada uma peça com uma dada geometria de frisos, o programa não realiza de imediato os dois ensaios (Estático e *Ball Drop*), em vez disso, o programa realiza apenas o ensaio estático, e depois procede com o desenho de uma nova geometria, onde novamente, irá apenas realizar o estático, e assim sucessivamente até todas as geometrias do caso de estudo terem sido tratadas. Com o fim da análise de todos os ensaios estáticos, então aí é que o programa passa para a análise do *Ball Drop*. O algoritmo funciona desta forma, pois, em média, o conjunto das tarefas de se desenhar os frisos, realizar o ensaio estático e analisar os dados obtidos demora cerca de 10-15 minutos para cada geometria, enquanto que, o um ensaio de *Ball Drop* sozinho demora cerca de 6 horas. Ora se se quiser estudar 50 geometrias, não é muito fiável realizarem-se 50 ensaios de *Ball Drop*, pois só esta componente demoraria 300 horas. Para se combater este fenómeno, como o objetivo do estudo é encontrar a geometria de frisos que permita à peça cumprir com os requisitos do cliente, utilizando-se a menor quantidade de material possível, o algoritmo primeiro realiza os 50 ensaios de estáticos, demorando entre 8-13 horas, e depois organiza as geometrias que passaram no ensaio estático, por ordem crescente da sua massa, assim quando se for a realizar o *Ball Drop*, o programa segue a ordem mencionada, e assim que encontrar uma geometria que passe no ensaio de *Ball Drop*, o estudo é terminado e essa mesma geometria é considerada a melhor, pois não existe outra com menor massa, que passe nos dois ensaios.

Traduzindo todo este processo para uma representação gráfica obteve-se o fluxograma da Figura 157. Para além do que é feito dentro do Abaqus®, também é necessário obter a informação através dos inputs do utilizador, organizar a informação toda num sistema de pastas bem documentado e apagar os ficheiros CAE e Odb das simulações, para que estas não ocupem a memória do computador.

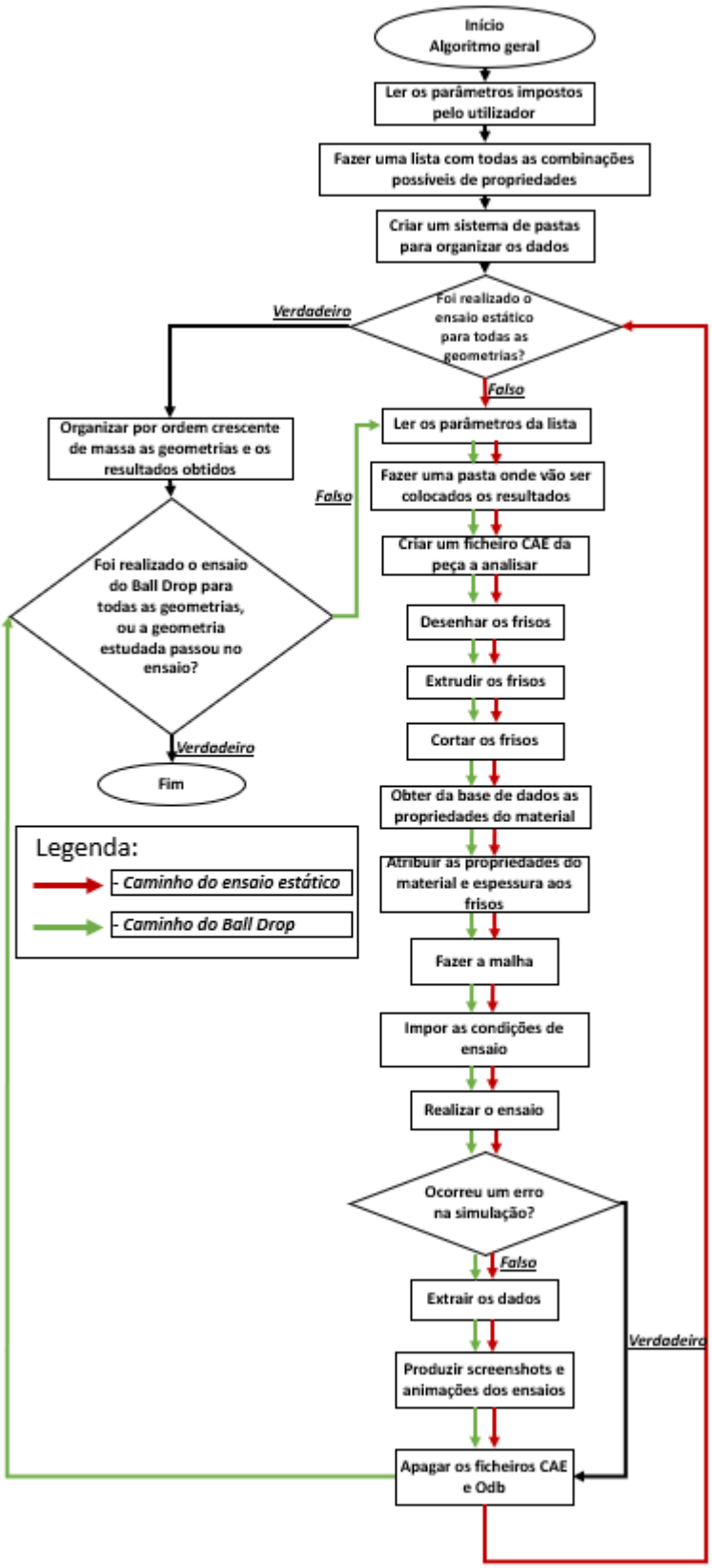


Figura 57 - Fluxograma do algoritmo geral do software

3.5.2 Algoritmo de desenho

Dentro do algoritmo geral, a parte mais complexa e importante do software é a metodologia de cálculo utilizada no desenho paramétrico dos frisos. Como estes podem possuir uma infinidade de geometrias e zonas de aplicação, foi necessário desenvolver um algoritmo matemático que os consiga desenhar, independentemente da peça e da área em que estes são introduzidos. Tendo isto em mente, desenvolveu-se uma função dentro do *script* que permite ao Abaqus® compensar o aumento ou redução do tamanho dos frisos, através da quantidade total que é introduzida na peça. Para se conseguir fazer a análise de quantos frisos são necessários, primeiro delimitou-se a área da peça onde estes se iriam inserir, e impôs-se a condição de que os frisos teriam de preencher ao máximo esta zona, sem ultrapassar os limites delimitados a verde na Figura 58.

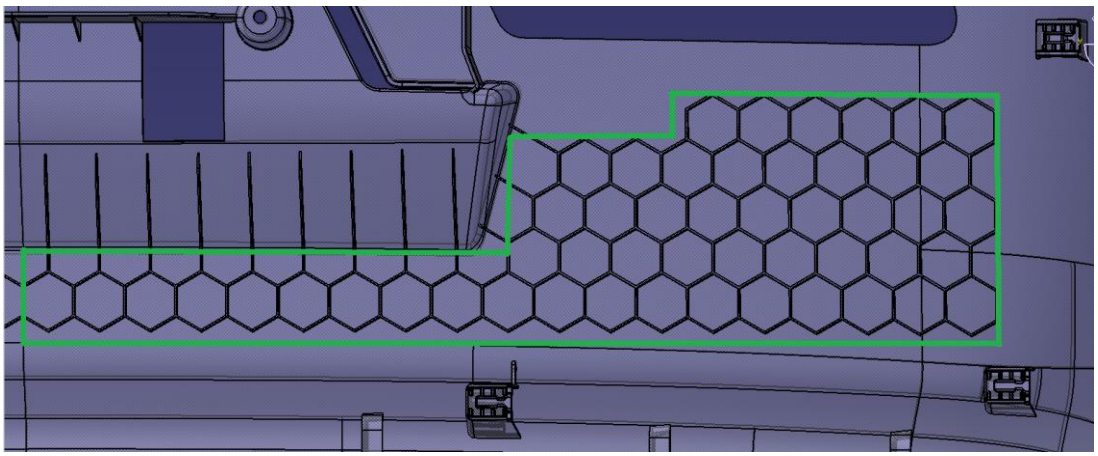


Figura 58 – Área de construção dos frisos

Tendo a área de construção delimitada, analisaram-se então diferentes maneiras de proceder com o desenho do conjunto de frisos, chegando-se assim a duas metodologias base para o seu desenho.

3.5.2.1 Metodologia de desenho

Para o desenho do conjunto dos frisos, obtiveram-se duas metodologias que seriam relativamente simples para implementar em Python. A primeira consiste em fazer o desenho segundo as linhas horizontais da geometria. Ou seja, o algoritmo iria desenhar cada uma das linhas representadas na Figura 59, numa sequência horizontal, iniciando-se pela fila mais baixa, seguindo o sentido da esquerda para a direita, e mais tarde fazer no sentido contrário (Figura 60). Esta troca deve-se ao facto de ter de se garantir a colinearidade entre a aresta esquerda do primeiro hexágono de cada uma

das linhas que se insere na zona estreita, e o eixo de simetria da peça, para que não existam descontinuidades entre as duas metades (Figura 61). Nos casos em que a linha a criar, não passa por esta zona estreita, os limites passam a ser a linha vertical á direita e uma linha diagonal á esquerda. Como o limite esquerdo tem de seguir uma diagonal ao longo das linhas que se encontram nesta segunda zona, torna-se mais fácil iniciar o desenho pelo limite direito (Figura 62). Esta tendência diagonal, leva a que os cálculos internos do programa sejam mais complexos e, portanto, que haja uma maior oportunidade de erro devido aos arredondamentos dos valores obtidos. Um dos erros que pode surgir, é a existência de pequenas interseções entre os diferentes hexágonos do reforço (Figura 63). Para além disto, apesar da metodologia ser bastante simples e conseguir desempenhar o desenho para a peça que se encontra em estudo, não permite uma grande alteração da geometria da zona de reforço, limitando a sua utilidade a peças cuja zona seja relativamente semelhante à demonstrada.

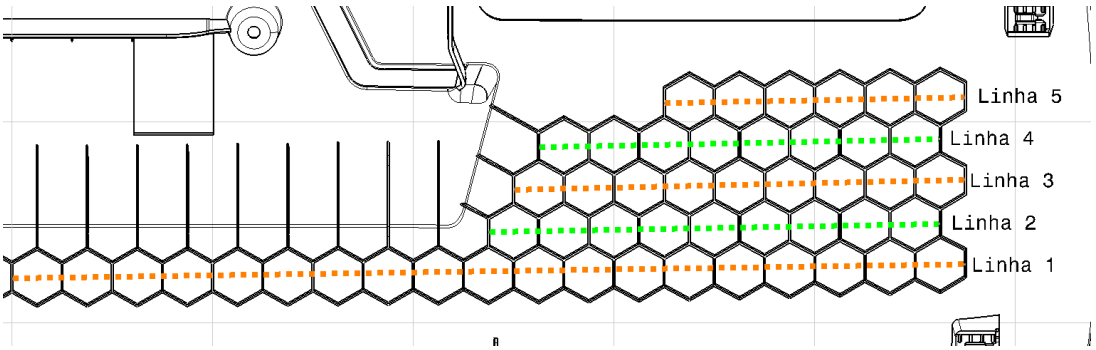


Figura 59 - Linhas do algoritmo horizontal de desenho

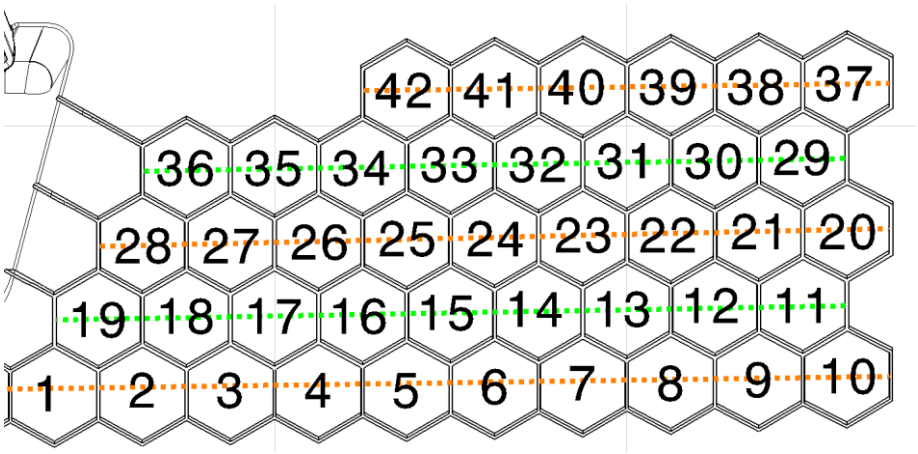


Figura 60 - Sequência de desenho do algoritmo horizontal

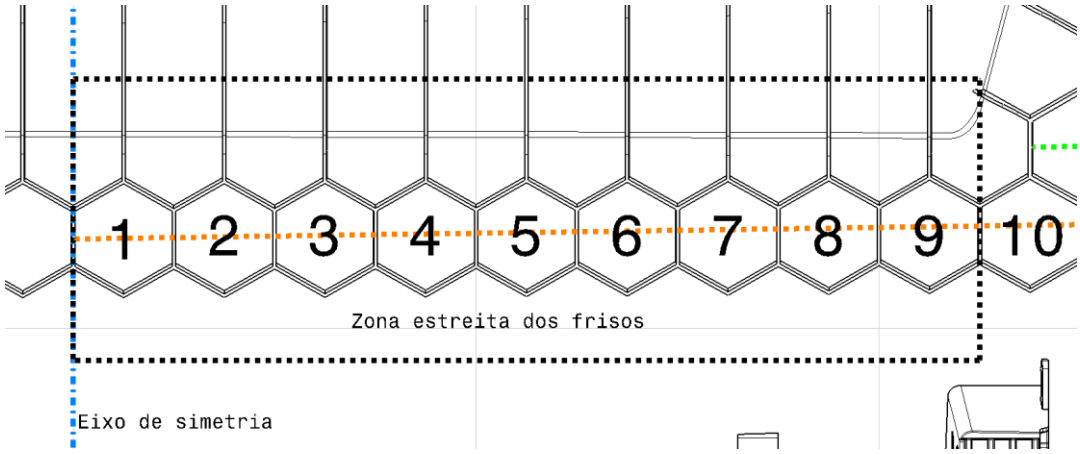


Figura 61 - Colinearidade entre o primeiro hexágono e o eixo de simetria da peça (vertical)

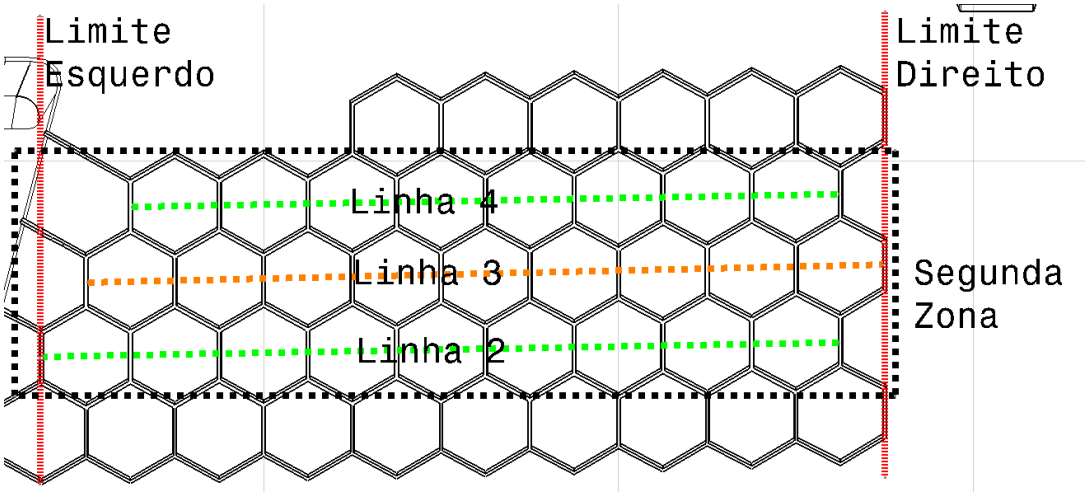


Figura 62 - Limites das Linhas 2, 3 e 4

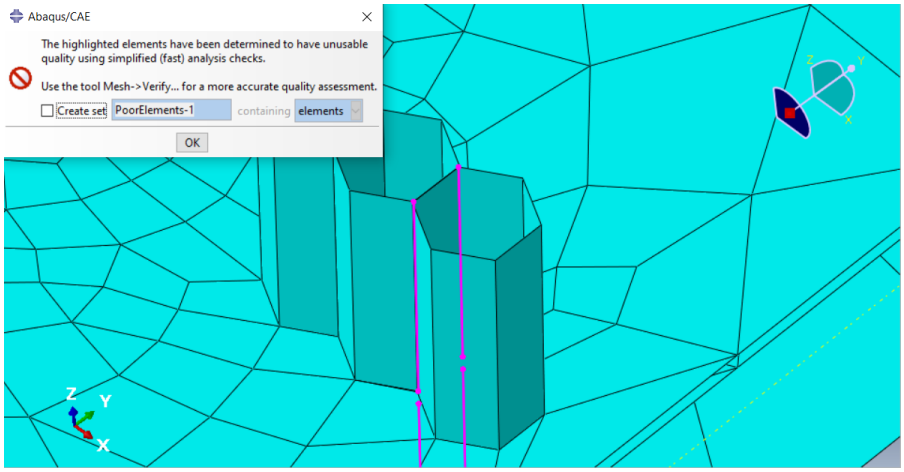


Figura 63 – Erro de malha causado por arredondamentos

O algoritmo diagonal funciona de maneira ligeiramente diferente do horizontal. Inicia-se na mesma o desenho pelo eixo de simetria da peça (Figura 66), para garantir a colinearidade, mas agora, em vez de se fazer a linha horizontal numa única passagem, cada vez que se desenha um dos hexágonos desta linha, fazem-se os hexágonos diagonais, como demonstrado nas Figura 64 e Figura 65. Algo que se tem de ter em conta com esta metodologia é o facto de, nos casos em que o algoritmo desenha mais que uma linha na zona estreita, em vez dos primeiros hexágonos de cada diagonal serem feitos na horizontal, estes terão de ser feitos na vertical, segundo o eixo de simetria da peça (Figura 67). Apesar de ter uma maior complexidade, o facto de se desenhar na diagonal, permite que os cálculos internos não sejam muito complexos, o que minimiza o risco de ocorrência de interseção entre hexágonos. Outra vantagem desta metodologia é o facto de ser mais flexível que a horizontal, no que diz respeito às zonas que se conseguem desenhar, podendo assim ser analisado um maior leque de peças diferentes através deste algoritmo.

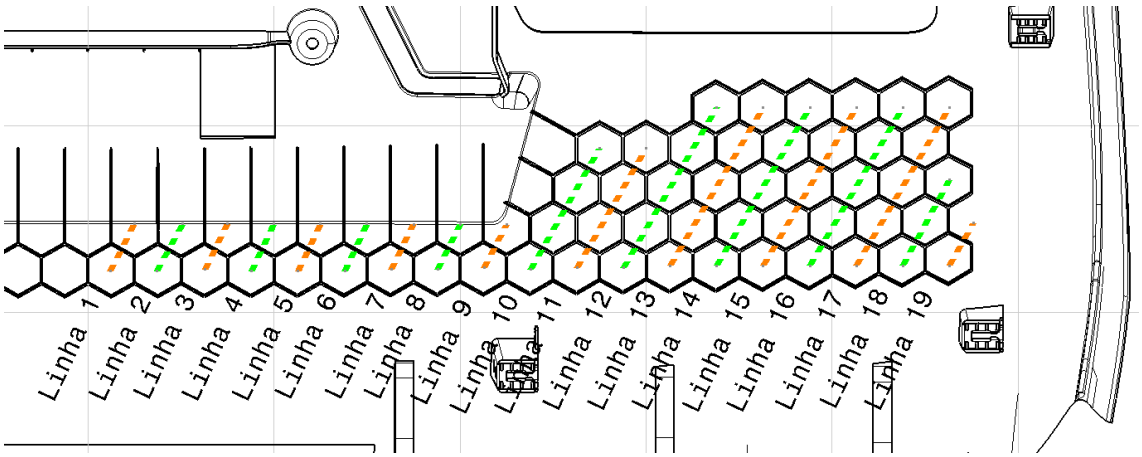


Figura 64 - Linhas do algoritmo diagonal de desenho

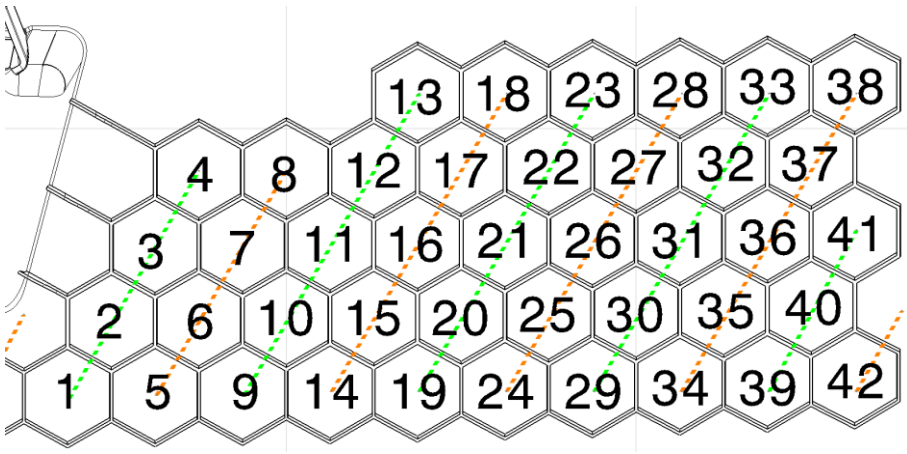


Figura 65 - Sequência de desenho do algoritmo diagonal

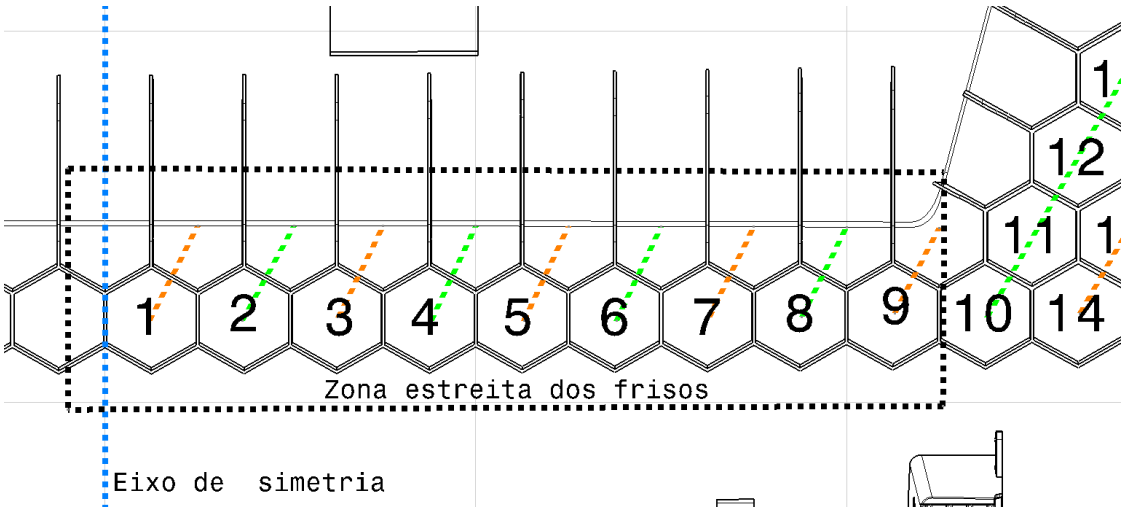


Figura 66 - Colinearidade entre o primeiro hexágono e o eixo de simetria da peça (diagonal)

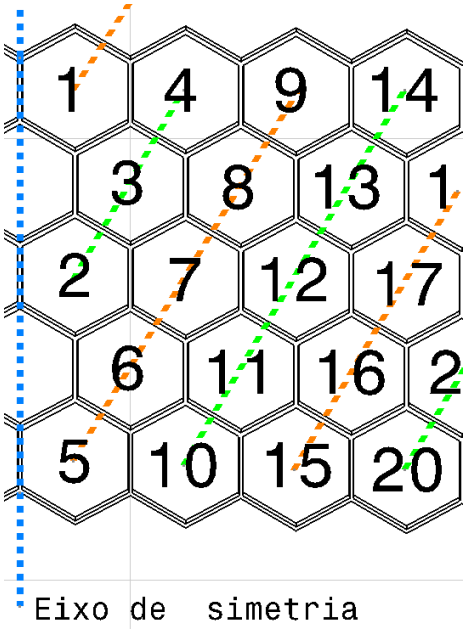


Figura 67 - Exemplo da síntese da primeira linha vertical do algoritmo diagonal

De entre estas duas metodologias, apesar de o algoritmo diagonal ter mais situações em que se deve ter um maior cuidado, e, como se verá mais à frente no algoritmo de deteção de limites, levar a que este seja utilizado mais vezes, esta metodologia permite uma maior flexibilidade da geometria de zonas que é possível de desenhar, permitindo assim uma transição mais fácil do algoritmo, caso se queiram estudar outro tipo de peças, com áreas de colocação de frisos diferentes.

3.5.2.2 Técnicas de redução de complexidade do algoritmo de desenho

De maneira a reduzir o tempo que demora ao programa a desenhar todos os frisos da peça, foram empregues algumas estratégias que permitem reduzir o número de ações que são desempenhadas durante o processo:

1. Assumir a simetria da peça;
2. Suprimir os hexágonos desnecessários para a geometria;

3.5.2.2.1 Simplificação do algoritmo através da simetria da peça

Apesar de a peça não ser completamente simétrica, após a simplificação da geometria no Software CATIA V5®, os elementos que impunham a dissimetria foram suprimidos, devido a não influenciarem os dados obtidos durante a simulação (Figura 68). Com isto em mente, para o caso em específico da peça que se está a analisar, pode-se apenas utilizar o algoritmo de desenho para desenhar metade dos frisos totais da peça, reduzindo assim o número de ações do programa para metade (Figura 69).

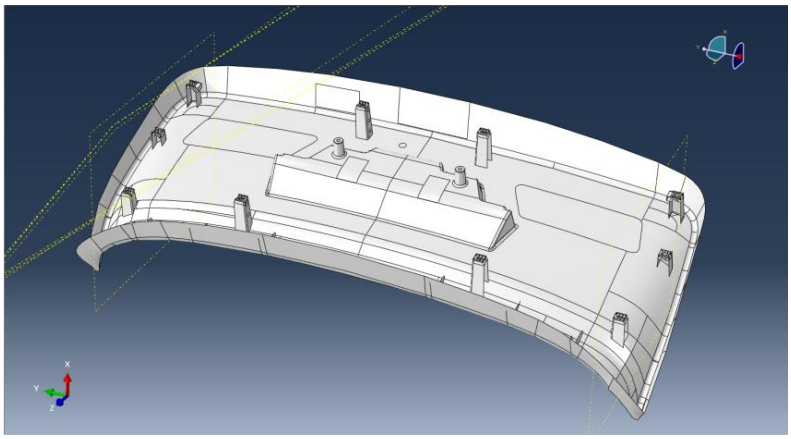


Figura 68 - Peça simplificada em Abaqus®

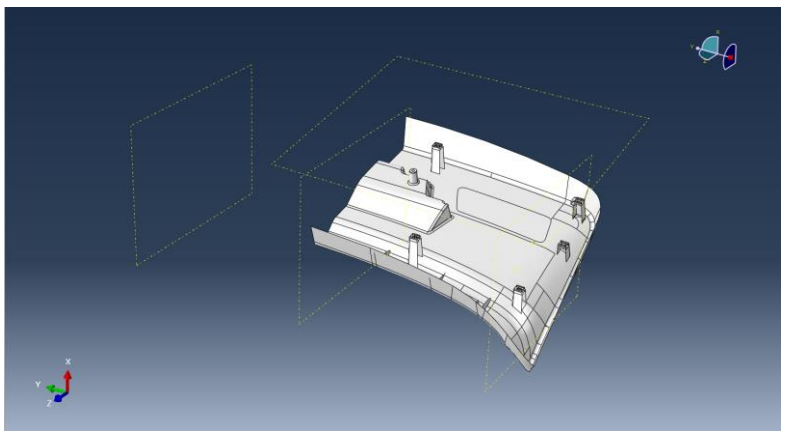


Figura 69 - Peça simplificada em Abaqus®, através do seu plano de simetria

3.5.2.2.2 Simplificação do algoritmo através da supressão de hexágonos

Numa estrutura de frisos hexagonais, como a da peça que se está a estudar (Figura 70), se se olhar com atenção é possível evidenciar que certos hexágonos da estrutura não necessitam de ser desenhados. Observando a Figura 71 a), evidencia-se um destes casos. O hexágono com o número 15 já é obtido através das arestas dos que o circundam (hexágono 10, 11, 14, 16, 19 e 20), não havendo assim necessidade do algoritmo de o desenhar (Figura 71 b)). Uma maneira de converter isto em código, seria de, quando se está a desenhar os hexágonos diagonais em vez de se ir aumentando o contador com incrementos de um, passam a ser feitos incrementos de dois, porém esta alteração só pode ser feita de duas em duas linhas, como demonstrado na Figura 72. Aplicando-se as correções mencionadas à geometria da Figura 70, espera-se então obter o desenho da Figura 73, onde os frisos desnecessários não são aplicados.

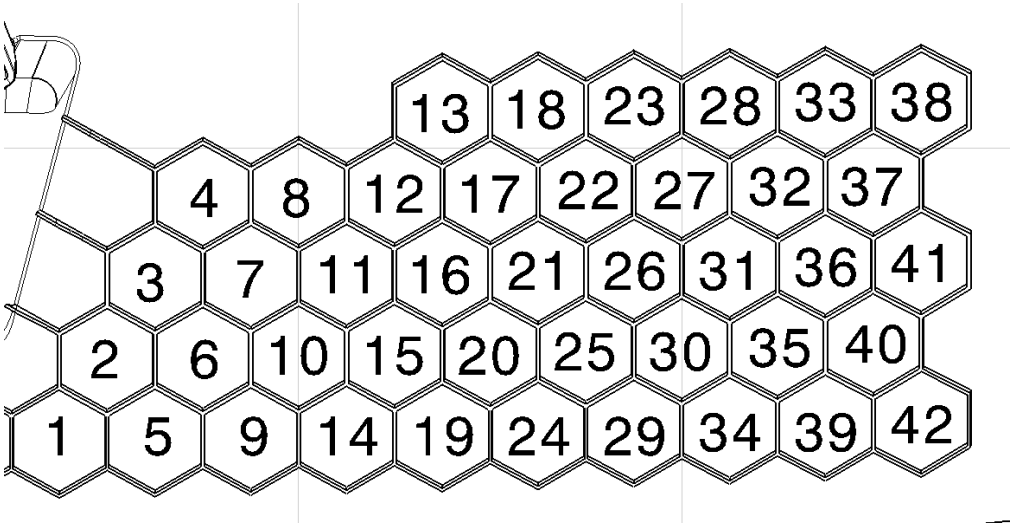


Figura 70 - Estrutura de frisos

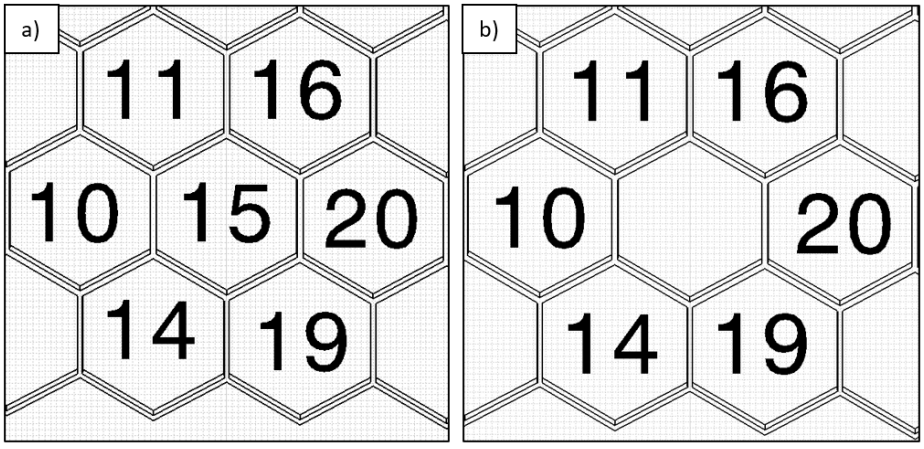


Figura 71 – Exemplo de uma situação de supressão

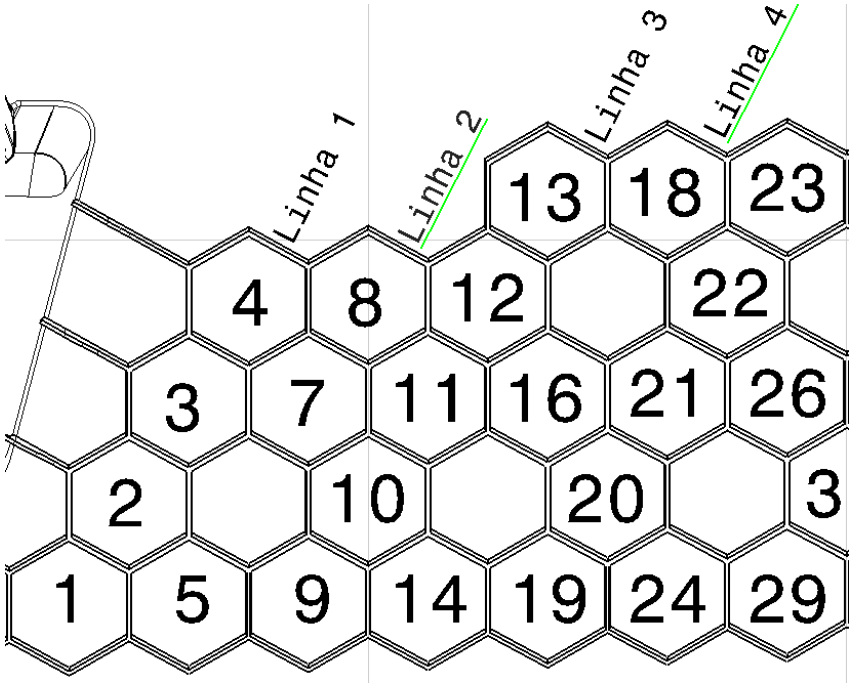


Figura 72 - Demonstração do incremento

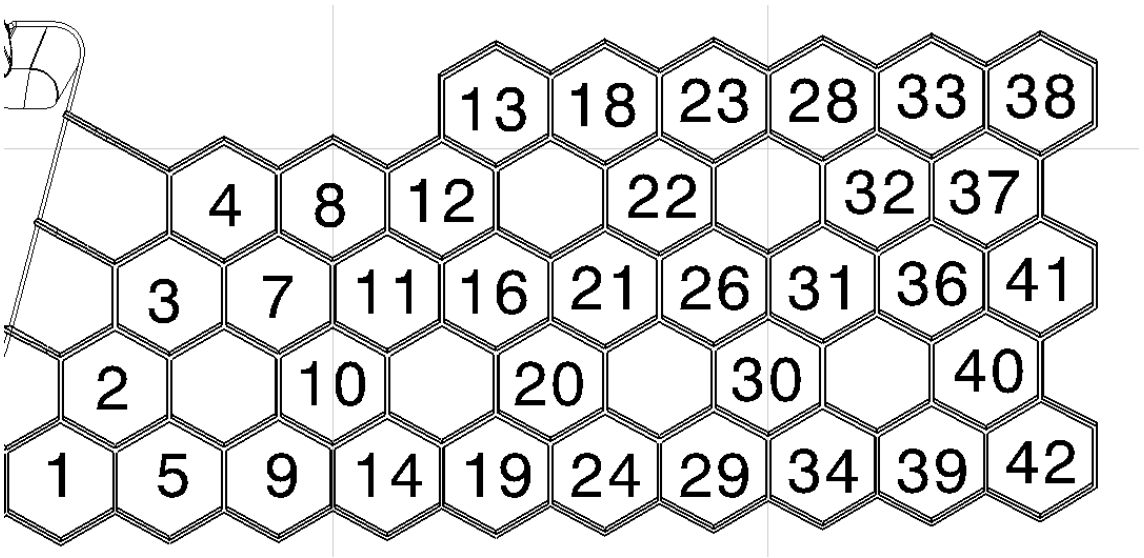


Figura 73 – Desenho do algoritmo diagonal com a correção

Apesar de ao saltar de duas em duas posições ser possível eliminar os hexágonos desnecessários, existem certas exceções ao longo da geometria. Observando com atenção a Figura 73, pode-se comprovar que, se fosse implementada a metodologia intercalar dos hexágonos, por si só, o número 8 e o número 37 seriam eliminados. Olhando para cada um deles pode-se inferir que isto é errado, pois nem o hexágono 8, nem o 37 têm arestas soltas á sua volta, que permitam que a sua geometria seja

produzida na sua totalidade. Com isto em mente esta metodologia não pode ser aplicada em vazio, o próprio algoritmo tem de ter funções de segurança implementadas, para permitir que situações como esta não acontecem, como por exemplo, não aplicar o incremento de dois, quando o hexágono gerado for o último da linha diagonal que está a ser desenhada. Este ponto das seguranças do software tem várias vertentes que também se enredam com outros componentes do software. A par disto, todas elas são abordadas mais à frente no Capítulo 3.5.3.6.3, onde se entra com maior detalhe e se explica como foram implementados em código Python.

3.5.2.3 Divisão do desenho em etapas

Uma limitação que tem de se ter em conta é o facto de o Abaqus® não conseguir extrudir a geometria dos frisos toda de uma vez. Isto advém do facto de, quando se tem quatro hexágonos dispostos como representado na Figura 74, devido ao número de arestas em comum, o software não consegue extrudir a geometria. De maneira a ultrapassar esta situação, a geometria é extrudida em dois passos, de colunas intercaladas duas a duas, como demonstrado na Figura 75, onde a estrutura da Figura 75 b) é obtida através da sobreposição das representadas nas linhas c) e d).

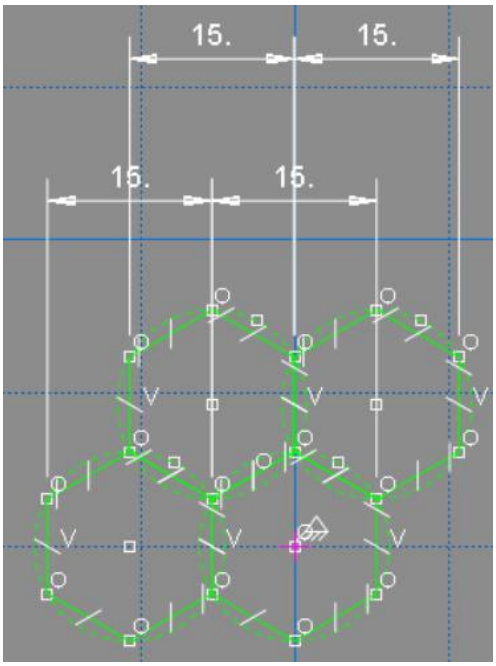


Figura 74 – Geometria de hexágonos impossível de extrudir

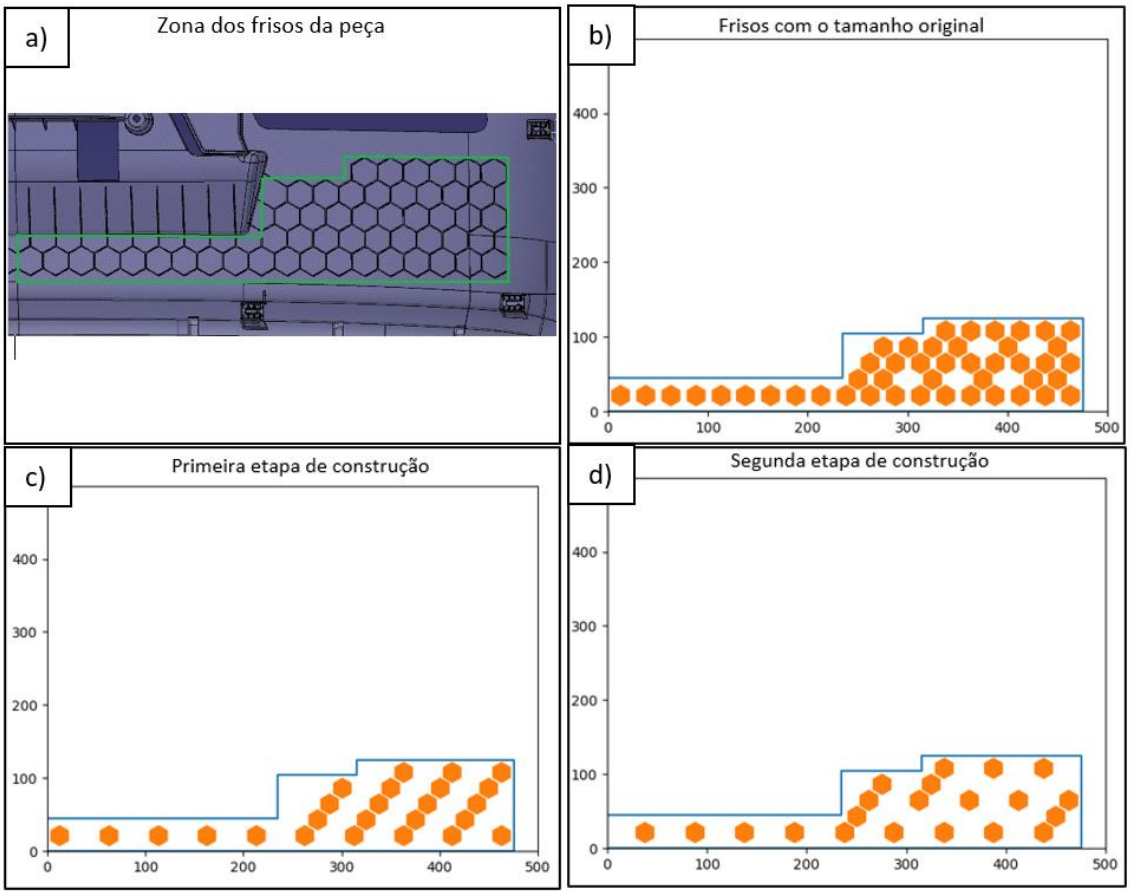


Figura 75 – Passos do algoritmo de desenho a) Zona de reforço b) Geometria gerada c) 1º Passo de extrusão d) 2º Passo de extrusão

3.5.3 Programação do algoritmo geral

3.5.3.1 Graphical User Interface (GUI)

Para permitir ao utilizador impor as condições dos frisos que deseja estudar, o programa, quando é iniciado, dispõe de um GUI onde a referida informação pode ser colocada nas respetivas janelas (Figura 76). Neste GUI, o utilizador pode colocar o intervalo de valores da largura, espessura e altura dos frisos que pretende estudar, bem como o número de valores que pretende utilizar desse intervalo, também pode especificar o valor da espessura da base da peça, o tamanho de malha a utilizar e pode ainda dizer ao programa se pretende realizar os ensaios ou apenas produzir o modelo com os frisos. Caso um valor inválido seja introduzido nesta janela, o GUI introduz uma janela de erro, e pergunta ao utilizador, se pretende introduzir novamente os valores (Figura 77).

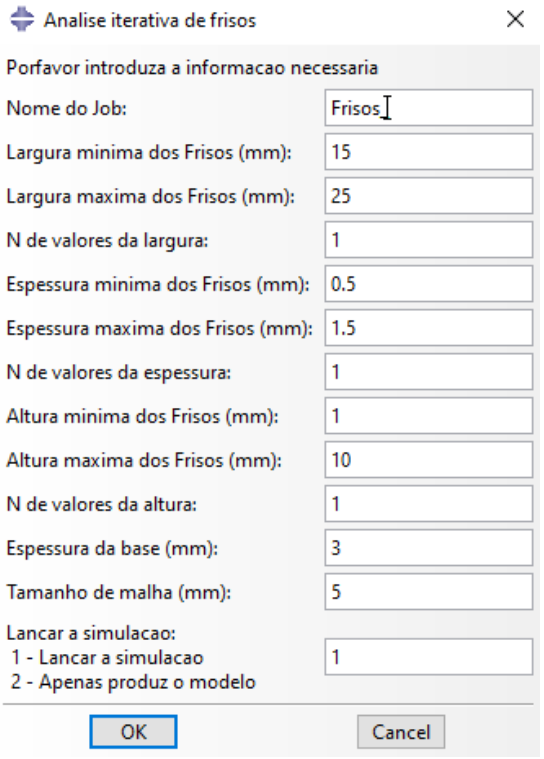


Figura 76 – Janela do GUI

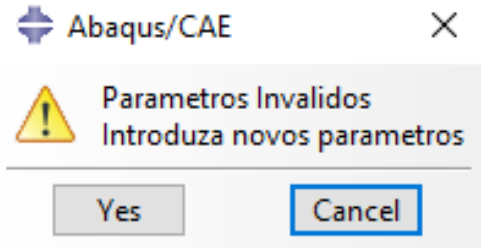


Figura 77 – Janela de erro do GUI

A função que trata do GUI do programa é a *GUI_Frisos* da Figura 78 e Figura 79, que utiliza a função *getInputs* do Abaqus®, que permite criar uma janela, onde é possível introduzir valores, que são depois armazenados numa lista (Figura 78 Linhas 5-21) [62]. Para tornar a escrita e leitura do software mais intuitiva, a lista é depois convertida para um dicionário, em que cada uma das variáveis é designada por um nome chave, em vez de um índice de uma lista [63] (Figura 78 linhas 24-36). Este dicionário tem depois os seus valores avaliados, para conferir que nenhum valor inválido é introduzido no *GUI*, caso isto aconteça, a função *getWarningReply* cria a janela de erro, com o texto que lhe é introduzido como argumento [63]. Se o utilizador introduzir os valores corretamente, ou carregar no botão de cancelar, o *GUI* é desligado (Figura 79 Linhas 40-59).

Com os dados guardados, o dicionário resultante é devolvido à função principal do programa e este depois recorre à função *Create_Geometry_Variable_List* para criar uma lista com cada um dos possíveis valores para uma determinada variável (Figura 80). Esta função é utilizada para a altura, espessura e largura dos frisos, e depois as três listas geradas são levadas para a *Geometry_Test_Combination*, que cria um dicionário com todas as permutações possíveis de valores para a geometria dos frisos (Figura 81).

```
# Graphical User Interface
def GUI_Frisos():
    while True:
        # Opens the Graphical User Interface (GUI) window
        fields = [('Nome do Job:    ', 'Frisos_'), #0
                ('Largura minima dos Frisos (mm):', '15'),#1
                ('Largura maxima dos Frisos (mm):', '25'),#2
                ('N de valores da largura:', '1'),#3
                ('Espessura minima dos Frisos (mm): ', '0.5'),#4
                ('Espessura maxima dos Frisos (mm): ', '1.5'),#5
                ('N de valores da espessura:', '1'),#6
                ('Altura minima dos Frisos (mm):', '1'),#7
                ('Altura maxima dos Frisos (mm):', '10'),#8
                ('N de valores da altura:', '1'),#9
                ('Espessura da base (mm):    ', '3'),#10
                ('Tamanho de malha (mm):    ', '5'),#11
                ('Lancar a simulacao:\n 1 - Lancar a simulacao \n \
                    2 - Apenas produz o modelo    ', '1'),]#12
        user_inputs = getInputs(fields,label='Porfavor \
            introduza a informacao necessaria',
            dialogTitle ='Analise iterativa de frisos')

        # Saves the GUI values into a dictionary
        user_values = {'job_name_GUI' : user_inputs[0],
        'Lenght_max_GUI' : float(user_inputs[2]),
        'Lenght_min_GUI' : float(user_inputs[1]),
        'Lenght_n_val_GUI' : int(user_inputs[3]),
        'Thickness_max_GUI' : float(user_inputs[5]),
        'Thickness_min_GUI' : float(user_inputs[4]),
        'Thickness_n_val_GUI' : int(user_inputs[6]),
        'Height_max_GUI' : float(user_inputs[8]),
        'Height_min_GUI' : float(user_inputs[7]),
        'Height_n_val_GUI' : int(user_inputs[9]),
        'base_thickness_GUI' : float(user_inputs[10]),
        'mesh_GUI' : float(user_inputs[11]),
        'simulation_GUI' : int(user_inputs[12]),}
```

Figura 78 – GUI (*GUI_Frisos*) (1/2)

```
38          # Validates the GUI parameters, in order to check if any of them
39          # are out of the ordinary
40          if (user_values['Lenght_max_GUI']<=0) or \
41              (user_values['Lenght_min_GUI']<=0)  or \
42              (user_values['Lenght_n_val_GUI']<=0) or \
43              (user_values['Thickness_max_GUI']<=0) or \
44              (user_values['Thickness_min_GUI']<=0) or \
45              (user_values['Thickness_n_val_GUI']<=0) or \
46              (user_values['Height_max_GUI']<=0) or \
47              (user_values['Height_min_GUI']<=0) or  \
48              (user_values['Height_n_val_GUI']<=0) or \
49              (user_values['base_thickness_GUI']<=0) or \
50              (user_values['mesh_GUI']<=0) or \
51              ((user_values['simulation_GUI'] != 1 and \
52                  user_values['simulation_GUI'] != 2)):
53              print('Parametros Invalidos')
54              answer = getWarningReply('Parametros Invalidos\n\
55                  Introduza novos parametros',(YES, CANCEL))
56              if answer == CANCEL:
57                  raise Exception('A analise foi cancelada')
58          else:
59              return user_values
```

Figura 79 – GUI (*GUI_Frisos*) (2/2)

```
# Create a list with all the parameters of a given variable
def Create_Geometry_Variable_List(var_max_value,
    var_min_value, var_n_val):

    return np.linspace(var_min_value,
        var_max_value, var_n_val)
```

Figura 80 – Criar uma lista com todas geometrias possíveis (*Create_Geometry_Variable_List*)

```
# Create a dictionary with all possible
# combinations of the friezes geometry
def Geometry_Test_Combinations(lenght_list,
    thickness_list, height_list):

    # Create a dictionary with all the geometry permutations
    test_permutations = [dict(zip(('Lenght','Thickness','Height'),
        (i,j,z))) for i,j,z in product(lenght_list,
        thickness_list, height_list)]

    return test_permutations
```

Figura 81 – Criar um dicionário com todas as combinações possíveis de parâmetros (*Geometry_Test_Combinations*)

3.5.3.2 *Organização de ficheiros*

Como o script gera vários ficheiros, para diversos tipos de geometrias e testes diferentes, é muito importante que a informação esteja bem organizada e documentada. Para tal, desenvolveu-se um sistema de pastas organizadas, que o script gera no momento em que é iniciado. A organização das pastas segue então a estrutura ilustrada na Figura 82.

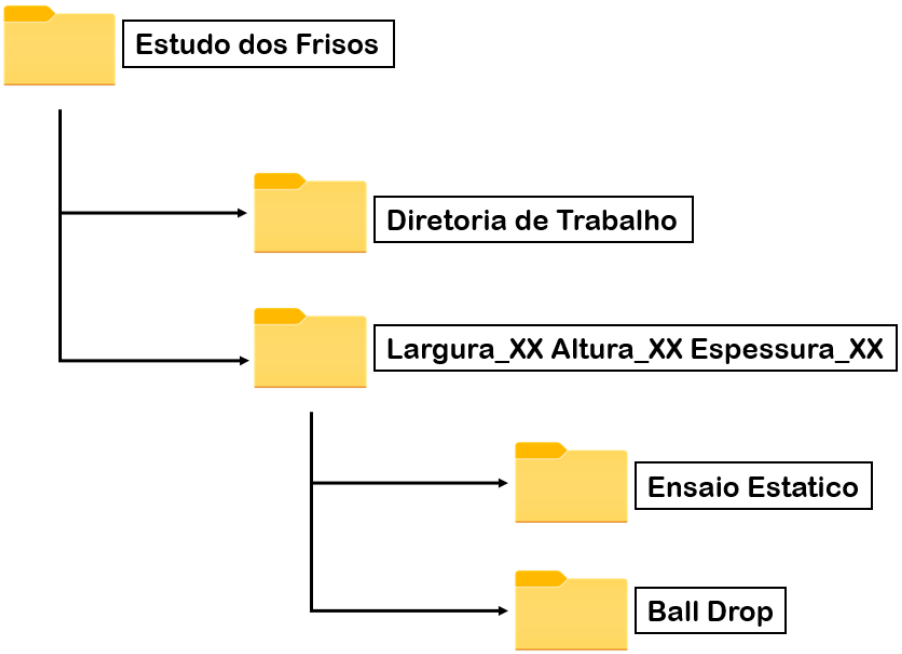


Figura 82 – Estrutura da organização de pastas

Seguindo a estrutura da Figura 82, o primeiro passo que se deve fazer é criar a pasta principal *"Estudo dos Frisos"*, esta pasta deve ser criada na diretoria em que se inicia o Abaqus®, de modo que o utilizador saiba onde esta se encontra. Como o Abaqus® pode ser aberto em qualquer localização do sistema operativo do computador, esta não pode ser introduzida como uma variável fixa, o programa tem de procurar onde é que o Abaqus® está aberto no sistema. Para isso recorre-se então à livraria *"os"*, que permite utilizar o método *getcwd*, que devolve a diretoria que está de momento a ser utilizada no trabalho (*Current working directory*), nessa mesma localização é onde vai ser criada a estrutura de pastas. Na Figura 83 está então representada a função do *script* que desempenha a procura da diretoria de trabalho, e a criação da pasta principal.

```
1 # Create the folder "Estudo dos Frisos_XX"
2 def Create_Folder_Structure():
3 
4     # Get current working directory
5     path = os.getcwd()
6 
7     # Create folder directory
8     newpath = path + '\Estudo dos Frisos'
9 
10    # Check if a folder with the same name already exists
11    if not (os.path.exists(newpath)):
12        os.mkdir(newpath)
13    else:
14        i = 2
15        while True:
16            newpath_2 = newpath + '_' + str(i) + ''
17            if not (os.path.exists(newpath_2)):
18                os.mkdir(newpath_2)
19                return newpath_2
20            else:
21                i += 1
22 
23    return newpath
```

Figura 83 - Função para criar a pasta principal (*Create_Folder_Structure*)

A função *Create_Folder_Structure* não recebe nenhum argumento, apenas se limita a criar uma pasta na diretoria de trabalho. Quando é iniciada, na linha 2 o *script* vai buscar a localização da diretoria de trabalho através do comando *getcwd*, guardando-a na variável *path* e depois concatena o nome da pasta que se pretende criar, que neste caso chama-se *"Estudo dos Frisos"*, guardando assim a nova diretoria na variável *newpath* (Linha 8). Com a diretoria da pasta que se pretende criar, formulada, resta então verificar se existe alguma pasta com o mesmo nome na diretoria de trabalho, para isso emprega-se o método *os.path.exists*. O *os.path.exists* recebe como argumento uma *string* que representa uma diretoria, e caso esta já exista dentro do sistema operativo devolve um booleano com o valor *"True"*. Como numa mesma localização, não se podem criar pastas com o mesmo nome, caso o valor devolvido pelo *os.path.exists* seja *"True"*, a função tem de alterar o valor guardado em *newpath* para outro que não exista. Sendo assim, caso a localização em newpath já exista, a função entra num *loop* onde adiciona um valor numérico a newpath até que a diretoria formulada não exista dentro da localização de trabalho. Ou seja, caso a pasta *"Estudo dos Frisos"* já exista, a função altera o nome para *"Estudo dos Frisos_2"*, e caso esta também já exista reformula para *"Estudo dos Frisos_3"* e assim em diante. Quando a diretoria formulada não existir no local onde se pretende criar, então a função procede com a criação da nova pasta através do *os.mkdir*. O *os.mkdir* pega numa diretoria que

recebe como argumento, e cria a pasta na localização especificada. Este processo de reformulação da diretoria e criação da pasta principal é feita entre as linhas 11 e 21. Quando a pasta é criada a função é então terminada, devolvendo a diretoria guardada em *newpath* ou *newpath_2*, dependendo se a primeira diretoria formulada existe ou não no sistema. Na Figura 84 está então representado o fluxograma do algoritmo da função *Create_Folder_Structure*.

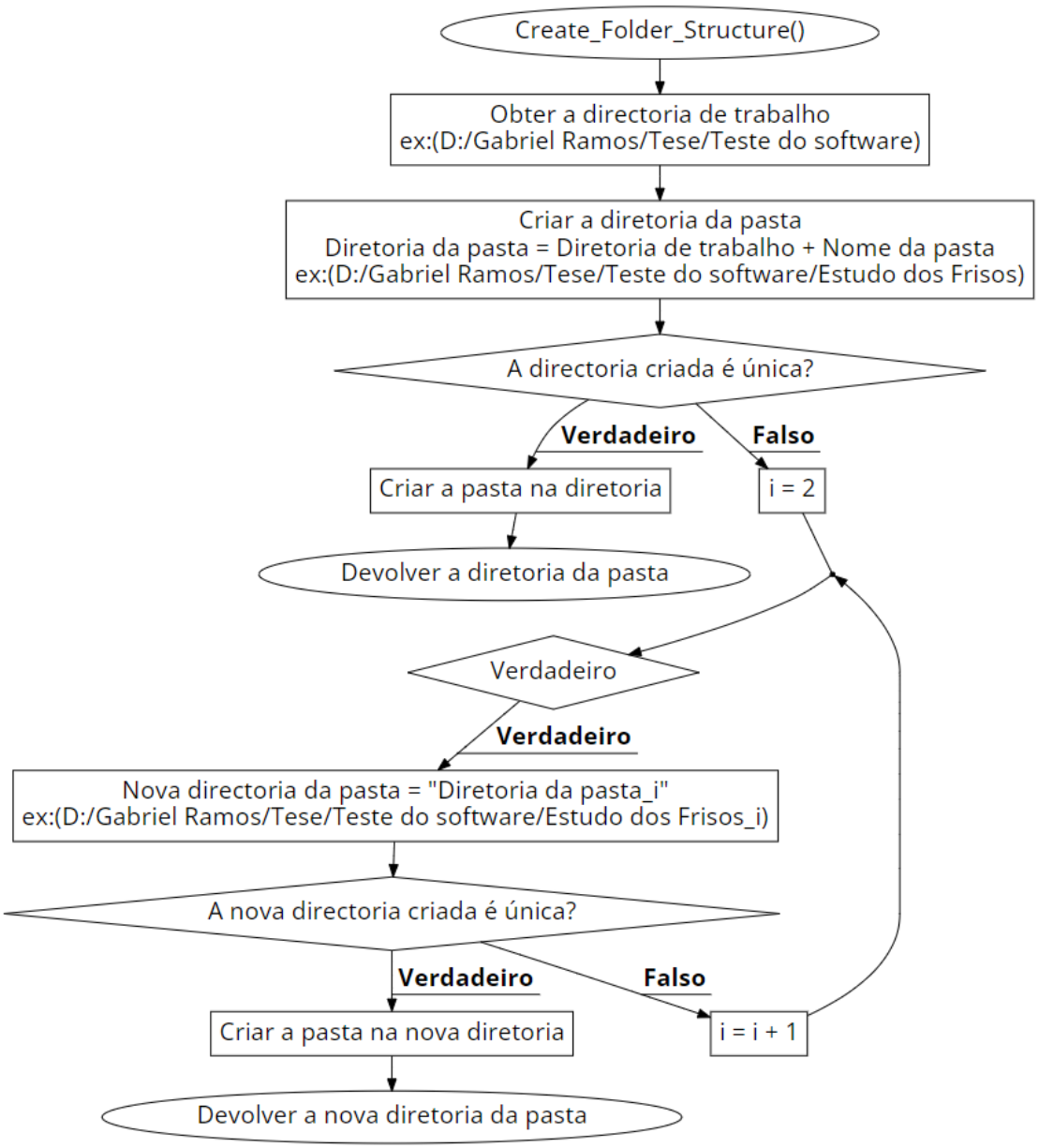


Figura 84 – *Create_Folder_Structure* (Fluxograma)

Com a pasta principal criada é necessário produzir a pasta onde vai ser guardado o ficheiro CAE original, bem como as pastas para cada uma das geometrias da peça, e subsequentemente, para cada um dos ensaios que cada uma das peças é sujeita. No que diz respeito ao local onde vai ser guardado o ficheiro original, desenvolveu-se a função *Create_Working_Directory_Folder*, demonstrado na Figura 85 e no fluxograma da Figura 86, e que recebe como argumento uma variável com o nome *path*, sendo esta uma string que especifica a diretoria onde vai ser criada a pasta. A variável *path* é a mesma diretoria que a função *Create_Folder_Structure* devolve, visto que a nova pasta vai ser criada dentro da pasta principal. Na linha 5, à diretoria que é recebida como argumento é então concatenado o nome da nova pasta e guardada a nova diretoria na variável *newpath*. Com a diretoria desenvolvida cria-se então a pasta através do *os.mkdir* (Figura 85 Linha 8), e devolve-se o valor da variável newpath para o programa para ser utilizada mais tarde (Figura 85 Linha 10).

```
1 # Create the folder "Diretoria de trabalho"
2 def Create_Working_Directory_Folder(path):
3 
4     # Create folder directory
5     newpath = path + '\Diretoria de Trabalho'
6 
7     # Create folder
8     os.mkdir(newpath)
9 
10    return newpath
```

Figura 85 – Função para criar a pasta de análise (*Create_Working_Directory_Folder*)

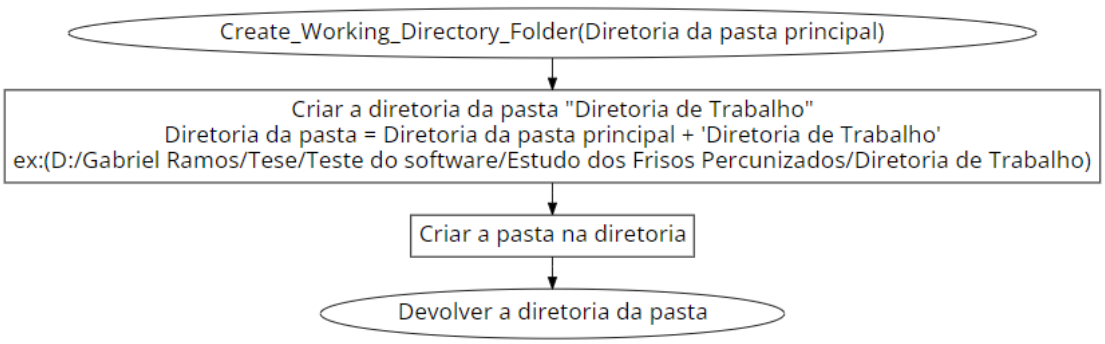


Figura 86 - *Create_Working_Directory_Folder* (Fluxograma)

A formulação das pastas das geometrias e dos ensaios, é feita de maneira diferente das mencionadas anteriormente. Como se quer criar uma pasta para uma dada geometria, e outras duas para cada um dos ensaios, dentro da primeira, em vez de se fazer a da geometria primeiro e depois as dos ensaios, pode-se só fazer as dos ensaios, mas em vez de se utilizar o método *mkdir* utiliza-se o *makedirs*. Esta última difere da primeira, no sentido que, se lhe for introduzida uma diretoria em que os valores intermédios não existam, o método *makedirs* cria as pastas desses mesmos valores, enquanto que a *mkdir* devolve um erro. Isto permite fazer as três pastas com apenas dois comandos, enquanto que com o *mkdir* seriam necessários três (Figura 87).

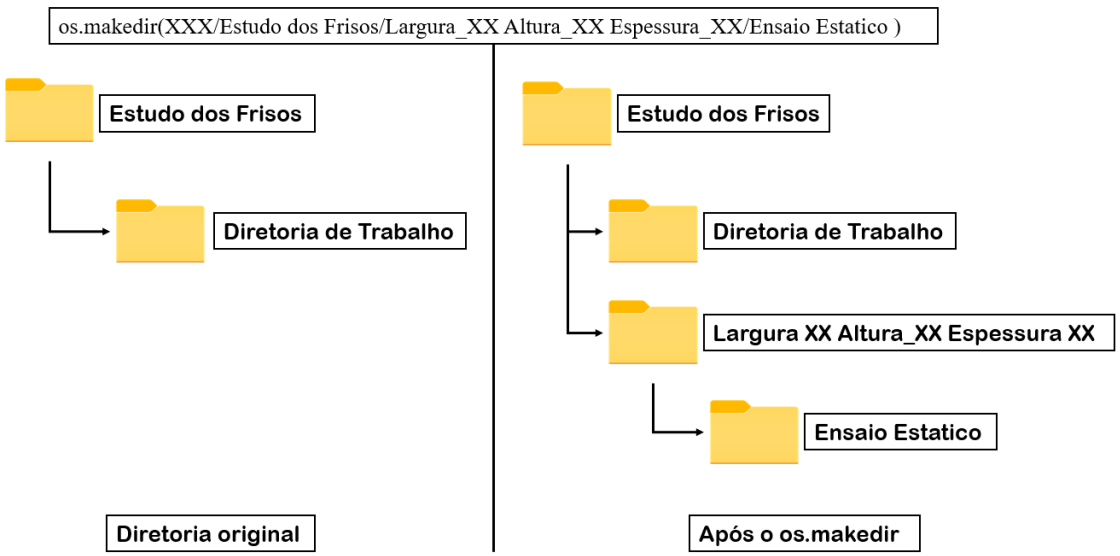


Figura 87 - Método *os.makedir*()

Traduzindo a metodologia da Figura 87 para código Python, obteve-se a função *Create_Test_Folder* da Figura 88 e da Figura 89. A referente função recebe como argumentos, a largura, altura e espessura dos frisos do respetivo estudo (*hex_size, hex_height* e *hex_thickness*, respetivamente) e a diretoria onde se vai criar as pastas (*dir_path*). A dir_path é a mesma diretoria devolvida pela função *Create_Folder_Structure*, visto que as pastas são criadas dentro da pasta principal *Estudo dos Frisos*. Tal como nas outras duas funções da estrutura de pastas, a função primeiro cria as diretorias de cada uma das pastas, através da concatenação da diretoria recebida como argumento, e os respetivos nomes de cada pasta (Figura 88 Linhas 4-13). Com as diretorias resultantes cria-se então as pastas, mas desta vez a partir do *makedir*, devolvendo as diretorias de cada uma delas.

```
# Create the folder for the respective tests
def Create_Test_Folder(hex_size, hex_height, hex_thickness, dir_path):

    # Create folder directory
    folder_name = ('Largura_' + str(hex_size)
        + ' Altura_' + str(hex_height)
        + ' Espessura_' + str(hex_thickness) + '')

    path_static = (str(dir_path) + '\\'
        + str(folder_name) + '\Teste Estatico')

    path_ball_drop = (str(dir_path) +'\\'
        + str(folder_name) + '\Ball Drop')

    # Create folders
    os.makedirs(path_static)
    os.makedirs(path_ball_drop)

    return path_static, path_ball_drop
```

Figura 88 – Função para criar as subpastas dos testes e dos ensaios (*Create_Test_Folder*)

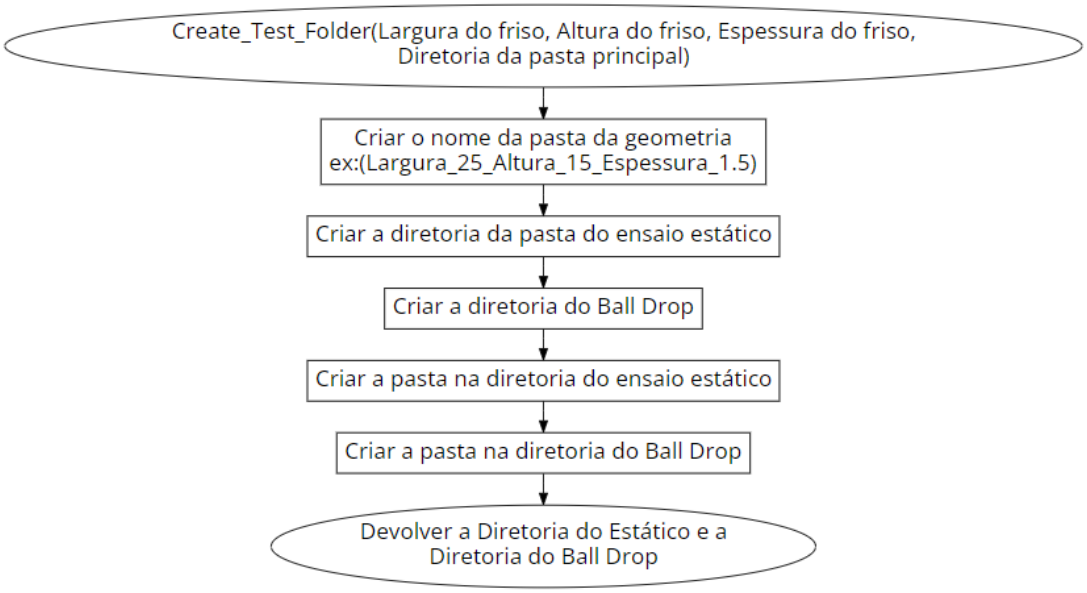


Figura 89 – *Create_Test_Folder* (Fluxograma)

3.5.3.3 *Função de importação do ficheiro base da peça*

Dentro do código responsável por desenhar os frisos na peça base, o primeiro passo a ser feito é então abrir o ficheiro do modelo da peça que se quer utilizar nos ensaios. Para isto quer se então descrever o processo de, na interface gráfica do Abaqus®:

1. ir a *file*;
2. *Import*;
3. *Model*;
4. Filtrar por ficheiro;
5. Escolher o tipo de ficheiro;
6. Localização do ficheiro e abrir, como demonstrado na Figura 90.

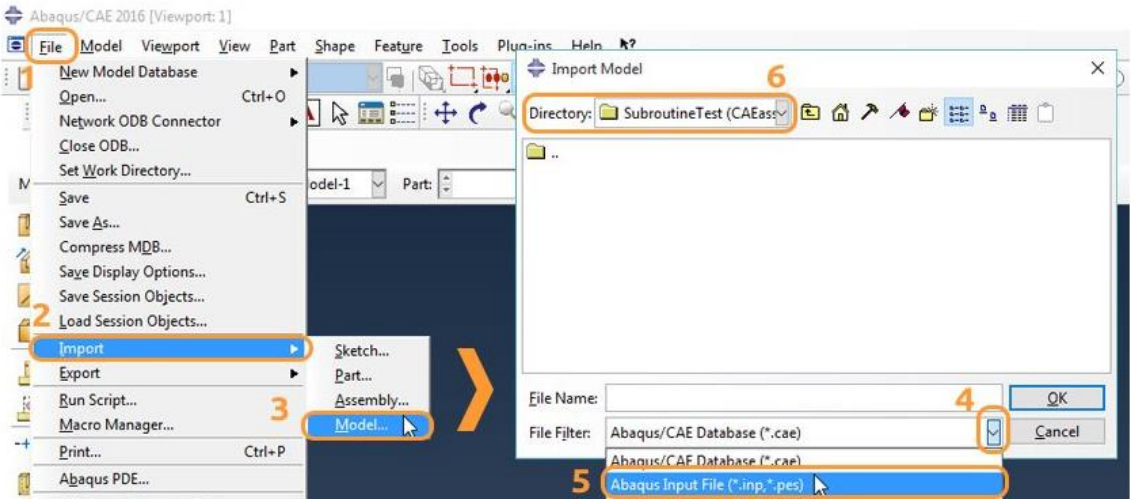


Figura 90 – Exemplo da importação de um modelo no Abaqus®

Para se conseguir o equivalente a isto, o que se faz normalmente, é utilizar o comando *openAuxMdb*, do objeto MDB (*Model Database*) do Abaqus®. Isto permite abrir uma MDB numa localização específica do disco do computador, como um ficheiro auxiliar [64]. A MDB pode ser um ficheiro onde são guardados os resultados de uma simulação, mais especificamente uma odb (*Output Database*), ou o próprio ficheiro CAE (*Complete Abaqus Environment*) onde se encontra o modelo da simulação e todas as suas especificações. Neste caso quer-se abrir o ficheiro CAE que contém a peça para a simulação. Para isto dá-se-lhe então, como argumento a diretoria que leva à localização do ficheiro: *"C:/Users/Gabriel Ramos/Desktop/Tese/Tese Simoldes/Teste do Software/Modelo_prototipo_base.cae"*. Com o ficheiro da peça base aberto quer-se então copiar todo o modelo de simulação, para o local onde se está a trabalhar. Para isto recorre-se ao *copyAuxMdbModel*, onde tem de se dar o nome do ficheiro auxiliar que se pretende copiar, e o nome que se quer que este passe a ter no ficheiro de trabalho. No contexto deste relatório, o modelo original tem o nome *"Model-1"* e quer-se que este continue a ser o mesmo (Linha 5). Com isto terminado fecha-se então

a MDB aberta na linha 6 da Figura 91 com o *closeAuxMDB*. Com estes comandos todos formulou-se a função que se encontra na Figura 91. Assim, em qualquer parte do script que se queira copiar o modelo base, basta enunciar a função *"Open_base_model"*.

```
1 def Open_base_model():
2     mdb.openAuxMdb(pathName='D:\Simulia_work\Projectos_2021'\
3         '\Estagio_Frisos_Preconizacao\Modelo Base'\
4         '\ShellEPortaAgrafos.cae')
5     mdb.copyAuxMdbModel(fromName='Model-1', toName='Model-1')
6     mdb.closeAuxMdb()
```

Figura 91 – Importação de um ficheiro CAE

Apesar da metodologia pelo *openAuxMdb* ser o que o Abaqus® emprega internamente no código, isto tem um problema que pode ser grave se a peça que se quer analisar for relativamente complexa. Quando o Abaqus® importa um ficheiro através do *copyAuxMdbModel*, ele tenta fazer uma reformulação automática da geometria da peça. Esta reformulação, em certos casos pode ser prejudicial para o *script*, pois, como não é possível prever as alterações que serão feitas, estas podem levar a um erro nalguma das funções internas. Um exemplo disto é se o Abaqus® tentar fazer uma partição na face onde os frisos vão ser introduzidos, como a face passa a ser constituída por diversas faces diferentes, a linha de repartição em conjunto com as linhas dos frisos pode levar a que se gerem secções pequenas que não sejam possíveis de gerar malha, ocorrendo assim um erro durante a execução do ficheiro (Figura 92). Para circundar o problema da reformulação, o que foi feito foi, em vez de se importar o modelo CAE para o Abaqus®, o ficheiro original é copiado para uma determinada diretoria do computador, onde este vai ser aberto e sujeito às funcionalidades do script (Figura 93). Assim, como se evita a utilização do *copyAuxMdbModel* a geometria da peça vai ser a mesma que a do modelo base.

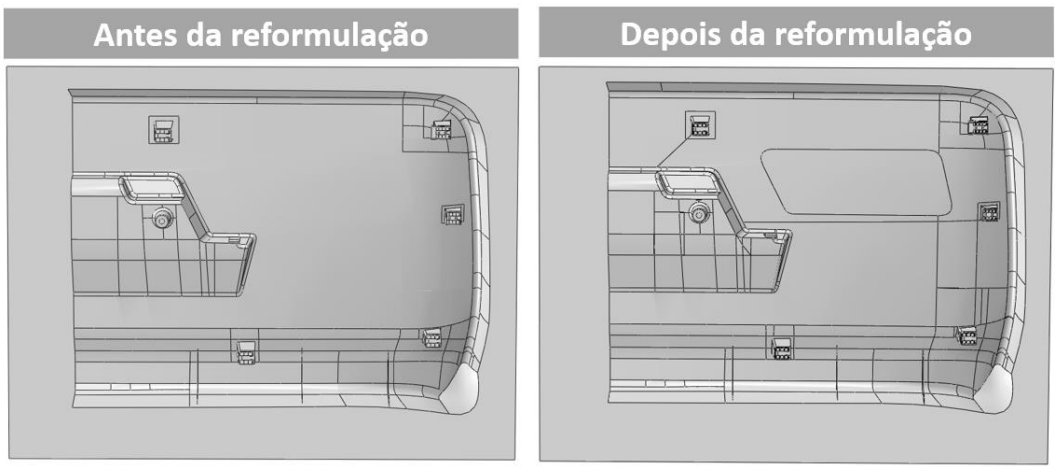


Figura 92 – Reformulação a geometria, devido ao *copyAuxMdbModel*

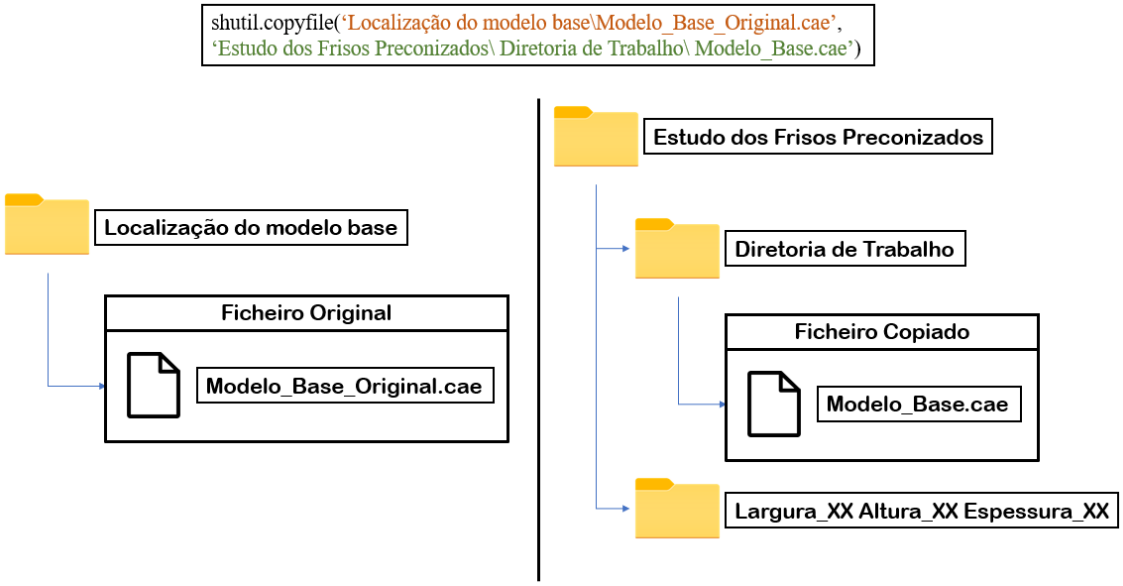


Figura 93 – Demonstração do método *shutil.copyfile*

Por si só o Abaqus® não possui a capacidade de copiar um ficheiro do computador, e enviá-lo para uma diretoria especifica, para isso foi necessário recorrer às funcionalidades do Python de manipulação de ficheiros dentro do sistema operativo, mais propriamente, à livraria *shutil.* Com a *shutil*, ganha-se acesso ao comando shutil.copyfile(), que recebe dois argumentos, uma string que representa a diretoria do ficheiro que se quer copiar, e uma segunda string que é a diretoria para onde se quer que este seja copiado [63]. Com o ficheiro copiado abre-se então este novo ficheiro para ser alterado. A abertura de um ficheiro no Abaqus® seria feita através de: *File* -> *Open File* -> Localização do ficheiro, o que dentro do código se traduz para o comando *openMDB()*, sendo o único argumento que este recebe, uma string que representa a localização do ficheiro a abrir [65].

Na Figura 94, tem-se então o novo código da função *Open_base_model*, onde em vez de se importar o ficheiro original, a função recebe como argumento a variável path, que corresponde á localização onde se quer guardar a cópia, e concatena-se o nome que se quer que a cópia passe a ter, e a sua extensão, que neste caso é *"Modelo_Base.cae".* Na linha 3, a sequência total é então guardada na nova variável *newpath*, e esta é utilizada no método *shutil.copyfile* como sendo a localização onde se vai guardar a cópia do ficheiro base. Dependendo do teste que se quer realizar o ficheiro a copiar é diferente, e, portanto, é necessário dizer à função qual é o teste que se quer realizar, através da variável test, que pode ser igual a "*Estatico*" ou "*Ball Drop*" (Figura 94 Linha 4 e 9). Mais tarde, na função principal do programa, o modelo é aberto através do método *openMdb*, utilizando-se a variável newpath como argumento.

```
# Open the base model depending on the test
def Open_base_model(path, test):
    newpath = path + '\Modelo_Base.cae'
    if test == 'Estatico':
        shutil.copyfile('C:\Users\Gabriel Ramos\Desktop\ ' \
            'Teste completo\Modelos\Estatico\ ' \
            'Modelo_prototipo_base.cae',
             newpath)
    elif test == 'Ball Drop':
        shutil.copyfile('C:\Users\Gabriel Ramos\Desktop\ ' \
            'Teste completo\Modelos\Ball Drop\ ' \
            'Modelo_prototipo_base.cae',
             newpath)

    return newpath
```

Figura 94 - Cópia e abertura do ficheiro base através da livraria *shutil*

3.5.3.4 *Função do desenho da geometria do hexágono*

No desenho da geometria total dos frisos, não é necessário desenhar as arestas uma a uma. Para se simplificar a formulação da geometria, desenha-se previamente apenas um dos hexágonos, e guarda-se em memória o seu *sketch*. Quando for necessário desenhar a geometria completa, vai-se buscar o *sketch* guardado, e movê-lo para a localização desejada, como demonstrado na Figura 95.

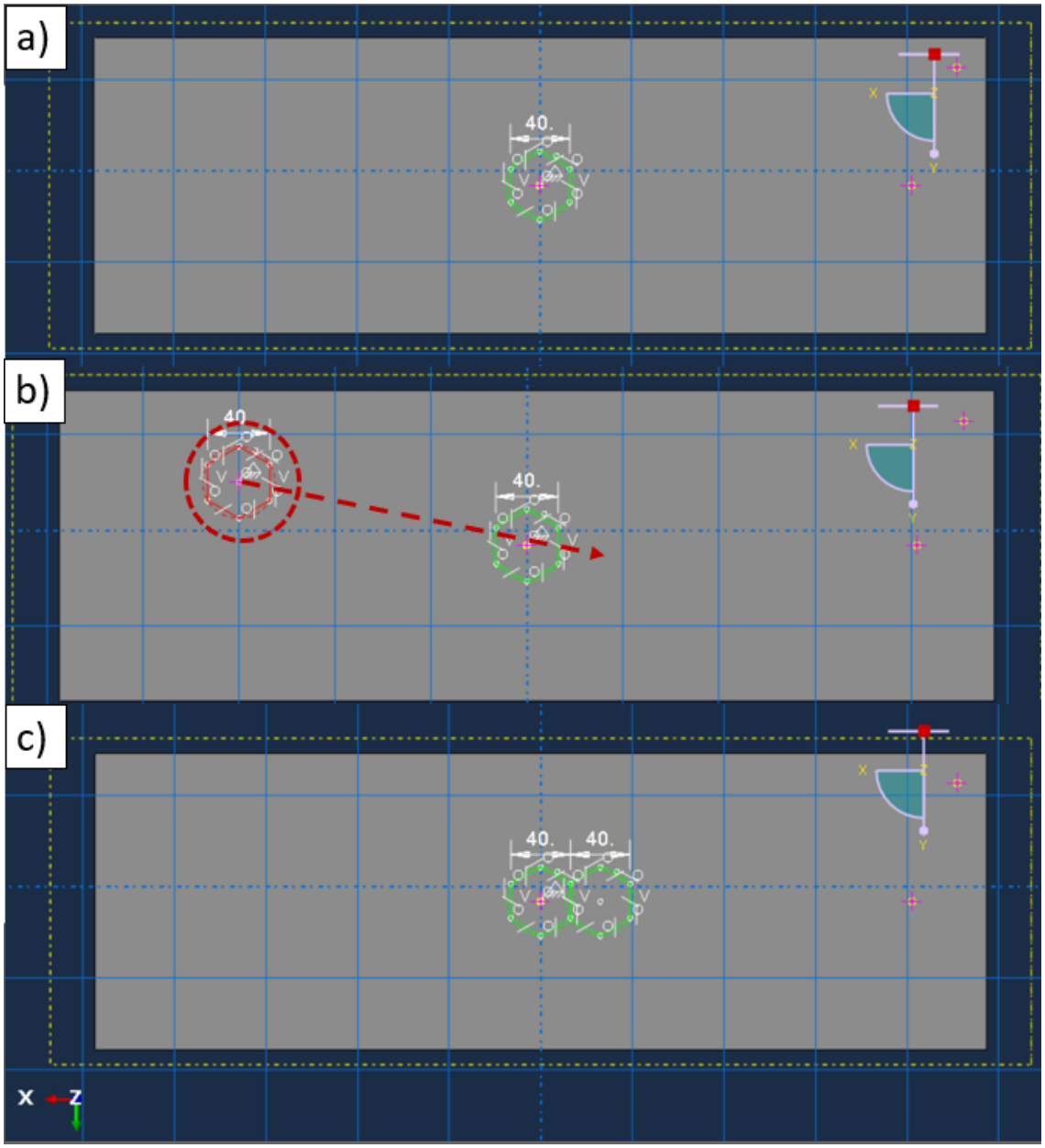


Figura 95 – Translação do hexágono base

O próprio desenho do hexágono é obtido, desenhando-se seis arestas unidas umas às outras, e só depois é que são impostas as restrições de coincidência, verticalidade e igualdade entre arestas, de maneira obter-se um hexágono equilátero. No final do desenho é então imposta a condição de largura do hexágono, através da atribuição de uma dimensão à geometria. Com a geometria desenhada, grava-se então o sketch e atribui-se um nome, tendo-se neste caso atribuído o nome de *"Hex"* (Figura 96).

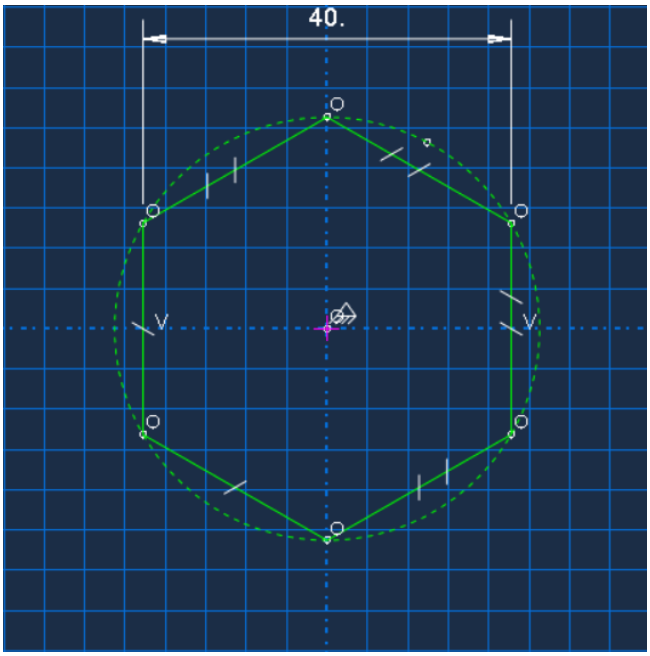


Figura 96 – Hexágono de base

Traduzindo-se o processamento da geometria do hexágono para Python, primeiro é necessário iniciar a janela "*Sketch*" onde se vai desenhar. Isto é feito através do objeto *ConstrainedSketch* presente na Figura 99 Linha 8, aqui tem de se então introduzir o nome inicial do perfil e o tamanho da folha, tendo se utilizado os valores de *"Hex"* e 200, respetivamente, que são os valores base utilizados pelo Abaqus®, quando se inicia um *Sketch* [64]. Do *sketch* extraem-se então os métodos *geometry* e *vertices*, que são repositórios onde são armazenados cada um dos respetivos parâmetros.

Antes de se descrever a componente de desenho do algoritmo, é necessário ter uma ideia de como funcionam os repositórios do programa. Um repositório dentro do Abaqus®, funciona como uma lista onde são armazenados todos os objetos de um determinado tipo de dados. Dois exemplos disto são os repositórios *geometry* e o *vertices*, que, respetivamente, guardam as arestas e os vértices de um desenho [66]. Como se encontra demonstrado na Figura 97, cada item de um repositório possui um número que o identifica, designado por *key*, e é através desse número que se pode referir à geometria através do código Python. Ora um problema que esta estrutura de dados tem, é o facto de, caso sejam removidas ou adicionadas novas geometrias, as *keys* que estão alocadas são alteradas. Um exemplo disto pode ser observado na Figura 98, onde devido à remoção de um dos itens, as chaves das arestas que a sucediam, reverteram para o valor anterior como se a primeira geometria nunca tivesse sido criada. Para além das alterações da geometria, a atribuição de *keys* de um modelo, pode ser diferente consoante a versão do Abaqus® que se está a utilizar [67]. Os métodos do objeto repositório são os mesmos que são utilizados nos dicionários do Python (Tabela 6) [67]:

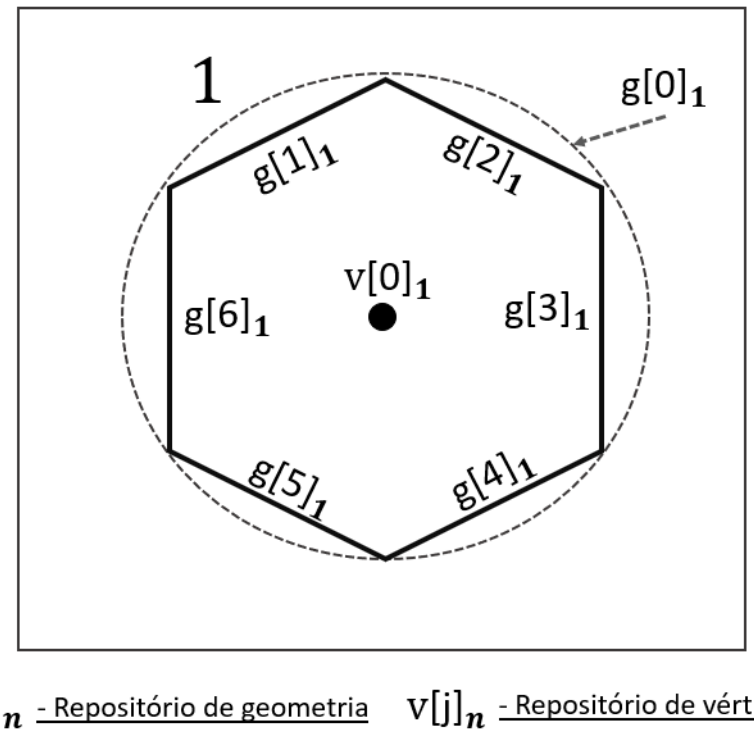


Figura 97 - Demonstração gráfica de um repositório e do hexágono gerado pelo programa

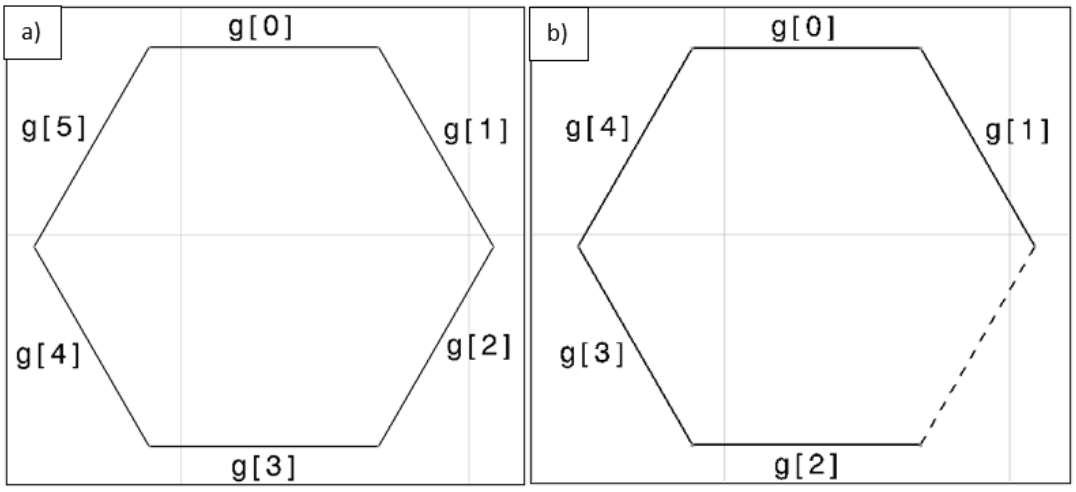


Figura 98 -Demonstração da remoção de um item de um repositório a) *Keys* iniciais b) *Keys* finais.

Tabela 6 – Métodos de um repositório

Método	Descrição
.has_key()	Devolve o valor booleano “True” se a key mencionada existir dentro do repositório;
.items()	Devolve uma lista com todas as chaves do repositório que não têm armazenado um valor nulo;
.keys()	Devolve um objeto de visualização que mostra todas as keys existentes num repositório, pela ordem que foram criadas;
.values()	Devolve um objeto de visualização que mostra todos os valores existentes num repositório, pela ordem que foram criadas;

Com a informação atrás referida, traduziu-se então todo o processo representado na Figura 96 para o código Python da Figura 99.

```
# Draw the hexagon profile
def Hex(hex_size):

    # Sketch Name
    sketch_name = 'Hex'

    # Create Hex Sketch
    s = mymodel.ConstrainedSketch(name=sketch_name,sheetSize=200.0)
    g, v = s.geometry, s.vertices
    s.CircleByCenterPerimeter(center=(0.0, 0.0), point1=(11.25, 21.25))
    s.Line(point1=(-18.75, 15.0519932234904),
        point2=(-18.75, -15.0519932236057))
    s.VerticalConstraint(entity=g[3], addUndoState=False)
    s.CoincidentConstraint(entity1=v[2], entity2=g[2],
        addUndoState=False)
    s.CoincidentConstraint(entity1=v[3], entity2=g[2],
        addUndoState=False)
    s.Line(point1=(-18.75, -15.0519932236057),
        point2=(0.0, -24.0442300770892))
    s.CoincidentConstraint(entity1=v[4], entity2=g[2],addUndoState=False)
    s.Line(point1=(0.0, -24.0442300770892),
        point2=(17.7218085984473, -16.25))
    s.CoincidentConstraint(entity1=v[5], entity2=g[2],addUndoState=False)
    s.Line(point1=(17.7218085984473, -16.25),
        point2=(17.7218085984473, 16.25))
    s.VerticalConstraint(entity=g[6], addUndoState=False)
    s.CoincidentConstraint(entity1=v[6], entity2=g[2],addUndoState=False)
    s.Line(point1=(17.7218085984473, 16.25),
        point2=(-1.25, 24.0117158903731))
    s.CoincidentConstraint(entity1=v[7], entity2=g[2],addUndoState=False)
    s.Line(point1=(-1.25, 24.0117158903731),
        point2=(-18.75, 15.0519932234904))
    s.setAsConstruction(objectList=(g[2], ))
    s.EqualLengthConstraint(entity1=g[3], entity2=g[8])
    s.EqualLengthConstraint(entity1=g[8], entity2=g[7],addUndoState=False)
    s.EqualLengthConstraint(entity1=g[7], entity2=g[6],addUndoState=False)
    s.EqualLengthConstraint(entity1=g[6], entity2=g[5],ddUndoState=False)
    s.EqualLengthConstraint(entity1=g[5], entity2=g[4],addUndoState=False)

    # Set the hexagon size
    s.DistanceDimension(entity1=g[3], entity2=g[6],
        textPoint=(0.0668716430664063,
        31.3793067932129), value=hex_size)

    s.Spot(point=(0.0, 0.0))
    s.FixedConstraint(entity=v[8])
    s.CoincidentConstraint(entity1=v[8], entity2=v[0])

    return sketch_name
```

Figura 99 - Algoritmo de desenho do hexágono base

Dentro da função "*Hex*", da linha 8 à 47 são então desenhadas cada uma das arestas do hexágono através de diversos métodos, que representam os diversos comandos de desenho do Abaqus®. Na Tabela 7 tem-se então a descrição de cada um dos que são utilizados na função "*Hex*".

Tabela 7 – Métodos usados na função "Hex"

Método	Descrição
CircleByCenterPerimeter()	Este método constrói um círculo usando um ponto ao centro e um ponto no perímetro da circunferência [68];
Line()	Constrói uma linha entre dois pontos [68];
Spot()	Insere um ponto numa localização especificada [68];
setAsConstruction()	Torna uma linha numa linha de construção [69];
VerticalConstraint()	Cria uma restrição vertical a uma linha [69];
CoincidentConstraint()	Cria uma restrição de coincidência entre geometrias [69];
EqualLenghtConstraint()	Impõem que duas ou mais linhas possuam a mesma largura [69];
FixedConstraint()	Impõem que uma geometria não se possa mover no espaço [69];
DistanceDimension()	Especifica a distância entre duas geometrias [69];

É importante de salientar que no método *DistanceDimension* da linha 41 não é atribuído um valor em concreto para a distância entre as duas geometrias *g[3]* e *g[6]*, é em vez disso introduzida uma variável com o nome *hex_size*, variável tal que é introduzida como um argumento da função *Hex* (Linha 2) podendo-se assim desenhar

o hexágono com valores de largura diferentes, desde que cada vez que se chame a função seja introduzido um valor diferente para o seu argumento (Figura 100).

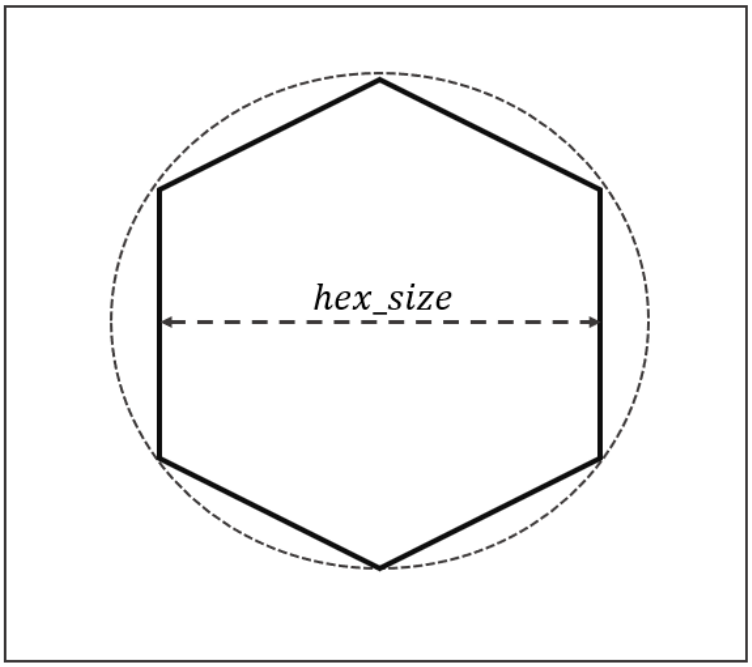


Figura 100 – Largura do hexágono base (*hex_size*)

3.5.3.5 *Pré-requisitos da função do desenho da geometria do reforço de frisos*

Antes de se iniciar a função responsável pelo desenho da geometria, para que ela consiga desempenhar a sua função, o modelo que vai ter a sua geometria alterada necessita de possuir determinadas propriedades para que a os frisos sejam desenhados com sucesso:

1. Tem de ser introduzido um Reference Point com o nome "*Sketch_Center_Point*", na face onde vão ser desenhados os frisos;
2. Tem de ser introduzido um *Coordinate System* auxiliar, com o nome "*Drawing_CSYS*", na face onde vão ser desenhados os frisos;
3. Tem de ser introduzido um *Datum Plane*, com o nome "*Sketch_Plane*", paralelo à face onde vão ser introduzidos os frisos;
4. Tem de ser introduzida uma face, com o nome "*Sketch_Plane_Bottom*", paralelo à face onde vão ser introduzidos os frisos, mas no sentido contrário ao "*Sketch_Plane*";

Dentro da função de desenho, o *reference point* "*Sketch_Center_Point*" demarca o ponto de origem do referencial da grelha que é utilizada no desenho da geometria, e é a partir deste ponto de referência que o programa calcula as posições dos diferentes hexágonos. No contexto do caso de estudo da peça presente, este ponto de referência

deve ser colocado sobre o eixo de simetria da peça, no local demonstrado na Figura 101.

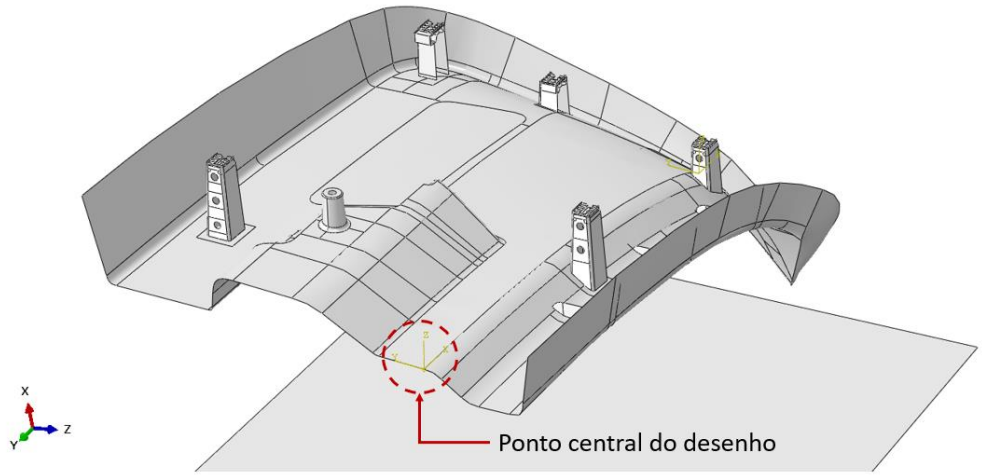


Figura 101 – Ponto central do desenho (*Sketch_Center_Point*)

O *coordinate system* "*Drawing_CSYS*" é utilizado para orientar a grelha utilizada no desenho da geometria, sendo o seu eixo y utilizado para demarcar a direção vertical da grelha, enquanto que a direção horizontal é delimitada através do eixo y do sistema de coordenadas e a localização do ponto utilizado como centro da grelha (Figura 102).

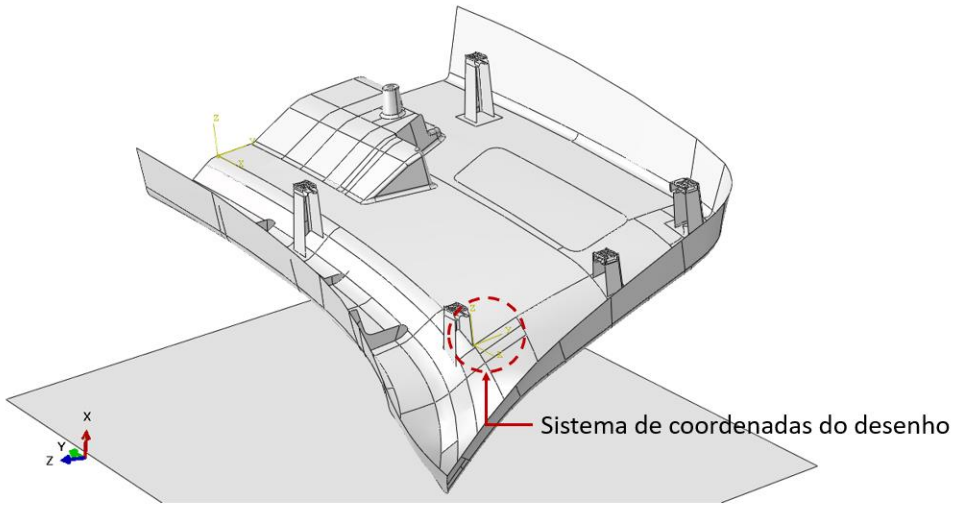


Figura 102 – Sistema de coordenadas do desenho (*Drawing_CSYS*)

O “*Sketch_Plane*” e “*Sketch_Plane_Bottom*” funcionam em conjunto, e servem como limites para a extrusão dos frisos. O “*Sketch_Plane*” é o local onde vão ser desenhados os frisos, e o “*Sketch_Plane_Bottom*” é o local até onde estes vão ser extrudidos. A razão pelo qual se utilizam estas duas faces é porque, em determinadas peças, se a face onde vão ser introduzidos os frisos tiver uma geometria relativamente complexa, o Abaqus® não deixa extrudir até essa face. De maneira a ultrapassar este problema, e tornar o *script* aplicável a qualquer tipo de peça, tomou-se partido de uma funcionalidade intrínseca do Abaqus®. Quando uma peça é extrudida até uma dada localização, e durante o seu percurso esta interseta uma segunda face, apesar desta não ser o seu limite, o Abaqus® realiza uma partição automática da face que está a extrudir, utilizando a segunda face como meio de partição, originando assim duas faces em vez de uma (Figura 103). Com isto em mente, o plano para extrudir os frisos é então o de os desenhar num plano acima do local onde se quer que estes sejam colocados (“*Sketch_Plane*”), e extrudir até uma superfície plana onde seja possível extrudi-los (“*Sketch_Plane_Bottom*”), cuja face onde os frisos devem ser colocados se encontre no trajeto da extrusão de frisos, de modo a realizar as partições (Figura 104).

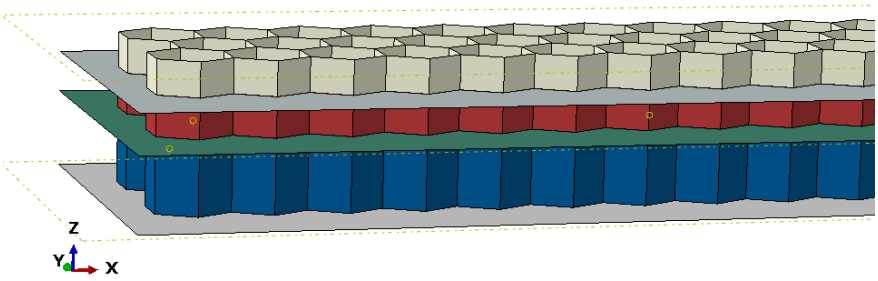


Figura 103 – Exemplo de partição automática

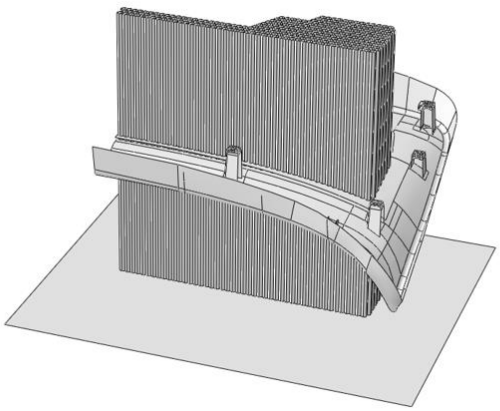


Figura 104 - Demonstração da metodologia de extrusão

3.5.3.6 Algoritmo de desenho (Python)

Como já foi referido no Capítulo 3.5.2, para se preencher a área onde vão ser aplicados os frisos (Figura 105), é empregue uma metodologia diagonal, no qual os frisos são desenhados diagonalmente, ao longo de uma linha horizontal que serve de base, estando esta dividida em duas partes, uma a que se denominou de zona A e outra de zona B. A zona A representa a área, cujos primeiros frisos da linha diagonal estão sobre o eixo de simetria da peça (Linha A), enquanto que a zona B corresponde aos frisos cujo primeiro friso se encontra segundo a linha horizontal (Linha B) (Figura 106). Esta distinção entre zonas é muito importante, pois, como os frisos da zona A não possuem uma linha horizontal para se poder iniciar o desenho, esta zona necessita de ter um algoritmo de cálculo das coordenadas de cada friso, diferente dos da zona B.

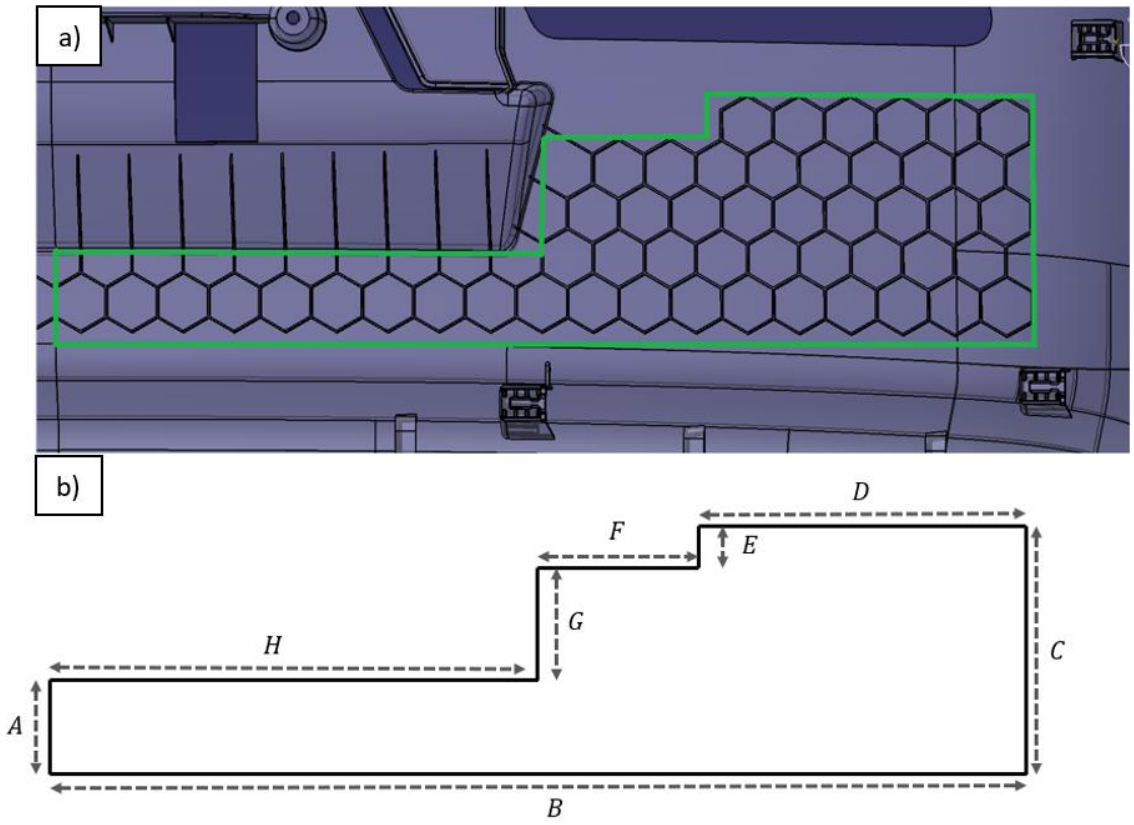


Figura 105 - Representação da zona de reforço a) Na peça original b) No algoritmo de desenho

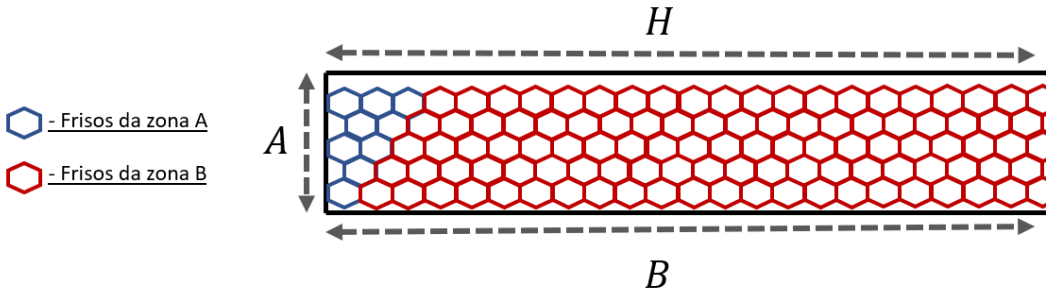


Figura 106 - Zonas do algoritmo de desenho

3.5.3.6.1 Algoritmo de desenho da Zona A – Metodologia teórica

Começando-se pelo algoritmo da zona A, estes são desenhados utilizando a linha A como base para os frisos iniciais e, portanto, a primeira coisa que deve ser feita, é calcular qual é o número máximo de frisos que se consegue colocar ao longo da sua extensão, tendo em atenção que a geometria dos frisos é variável ao longo dos ensaios. Para isto é necessário descobrir qual é a altura ocupada por uma dada geometria de frisos, e a altura que se tem disponível para os colocar. Com estas duas informações pode-se chegar ao número de frisos, arredondando por defeito (inteiro mais baixo) o valor obtido da equação (18).

$$vertical\_hex\_number\_A = \frac{Altura\ disponível}{Altura\ da\ geometria} \tag{18}$$

onde:

- $vertical\_hex\_number\_A$ – "Número de hexágonos máximo segundo a linha A"
- $Altura\ disponível$ – "Altura disponível para a colocação dos frisos" (mm)
- $Altura\ da\ geometria$ – "Altura da geometria de referência do friso" (mm)

No que toca à altura ocupada por uma dada geometria de frisos, como se pode ver na Figura 106, esta não é igual á altura do hexágono, pois, como entre dois hexágonos verticais existe um outro intercalado, a altura da geometria deve contabilizar este espaçamento entre os verticais. Sendo assim, a altura ocupada por uma dada geometria deve ser igual à soma entre a altura de um hexágono e a largura de um dos seus lados (equação (19)), como representado na Figura 107.

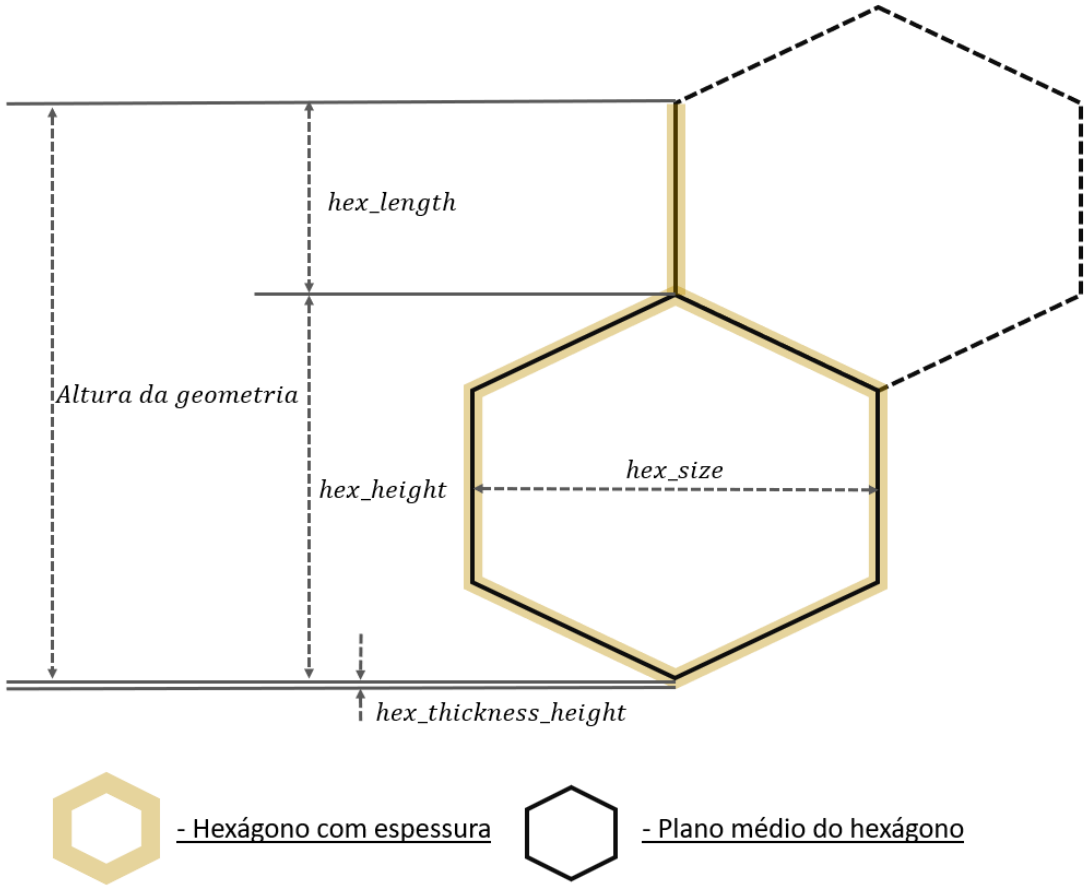


Figura 107 - Altura da geometria

$$Altura\ da\ geometria = hex\_height + hex\_length \quad (19)$$

onde:

- hex_height –"Altura do hexágono" (mm)
- hex_length –"Largura das arestas do hexágono" (mm)

No que diz respeito à altura disponível para se colocarem os frisos verticais, assumir que esta é igual à largura da linha A não é correto, pois tem de se ter em consideração que, tanto a espessura, como a largura dos frisos, vai influenciar a extensão que se deve considerar. Observando a Figura 108, onde está representada uma disposição onde cabem três frisos verticalmente, pode-se presenciar que, parte da largura da linha A vai ser ocupada pelas espessuras do primeiro e do último hexágono, tendo assim de se remover estes valores à largura da linha A. Para além disto tem também que se ter em atenção que, para o último hexágono da coluna, não é necessário desenhar a largura do hexágono intercalar, contudo este é contabilizado na altura ocupada pela geometria. Sendo assim tem de se adicionar à altura da linha A, o valor da largura do hexágono. Com estes dois critérios, para se chegar à largura disponível para o desenho dos frisos, deve-se adicionar o valor respetivo à largura dos frisos, à

largura da linha A, e remover-se a altura ocupada pelas espessuras, do primeiro e do último hexágono (equação (20)).

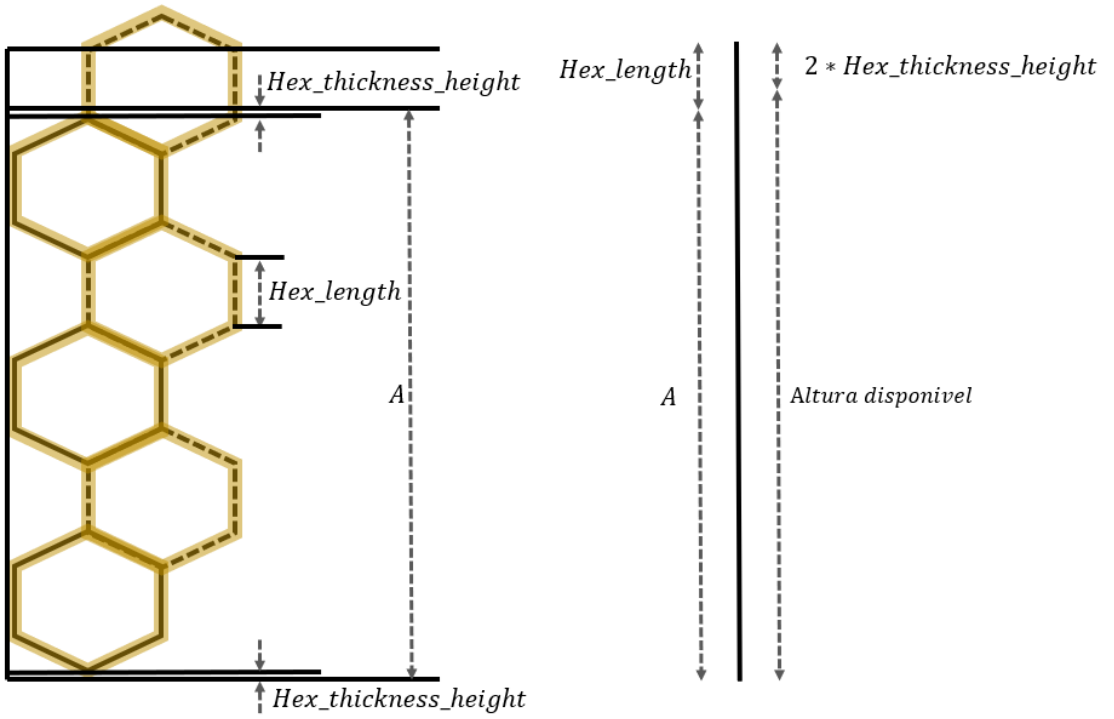


Figura 108 - Altura disponível

$$\text{Altura disponível} = A + hex\_length - 2 * hex\_thickness\_height \quad (20)$$

Onde:

- A —"Largura da linha A" (mm)
- $Hex\_thickness\_height$ —"Altura da espessura do hexágono" (mm)

Para o cálculo da altura disponível, resta apenas parametrizar o cálculo do espaço ocupado pelas espessuras do primeiro e último hexágono. Observando em pormenor a extremidade de um hexágono, é possível verificar que existe uma relação trigonométrica, entre a altura da extremidade, e a espessura do próprio hexágono, devido a formarem um triângulo retângulo, com um angulo interno de 30º, onde a altura da espessura corresponde à hipotenusa, e metade da espessura a um cateto (Figura 109) (equação (21)).

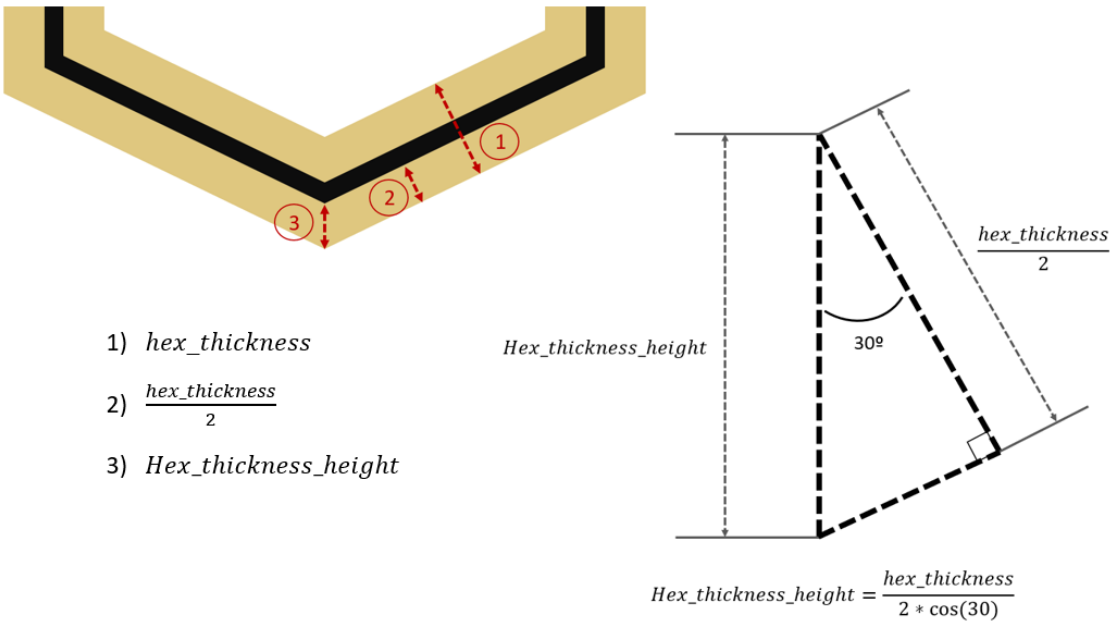


Figura 109 - Altura ocupada pela espessura do hexágono

$$hex\_thickness\_height = \frac{hex\_thickness}{2 * \cos(30^{o})} \tag{21}$$

Onde:

- $hex\_thickness$ –"Espessura dos frisos" (mm)

Com todos os parâmetros contabilizados, combinaram-se as equações (18) , (19) , (20) e (21), para se obter a equação (22), que relaciona o número de hexágonos que é possível colocar verticalmente na zona A, em função dos parâmetros dos hexágonos.

$$vertical\_hex\_number\_A = \frac{A + hex\_length - \frac{hex\_thickness}{\cos(30^{o})}}{hex\_height + hex\_length} \tag{22}$$

Possuindo o número de hexágonos que devem ser colocados ao longo da linha A, desenvolveu-se então uma metodologia de desenho para estes frisos. No desenvolvimento teve-se então o cuidado de o desenhar, de modo que os hexágonos que são desnecessários para a geometria final não sejam produzidos e que o processo de extrusão seja feito em duas etapas diferentes (Figura 110), devido às razões mencionadas no Capítulo 3.5.2.3.

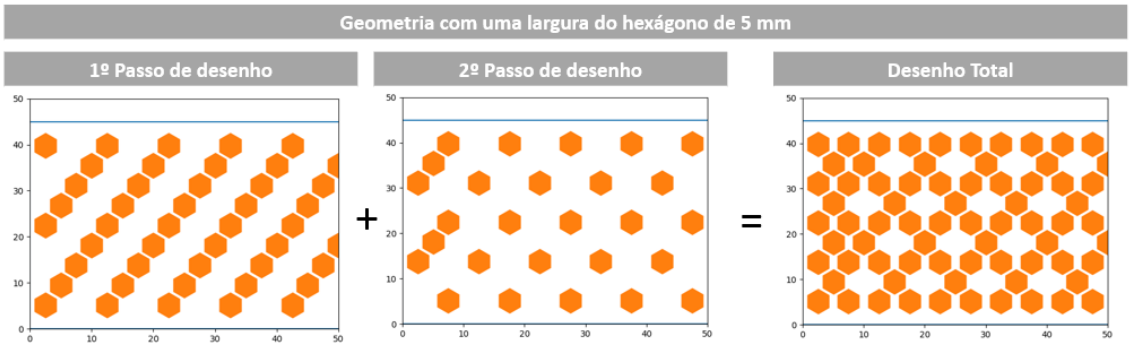


Figura 110 - Demonstração da metodologia de desenho da linha A

Como o número de hexágonos que pode ser desenhado na zona A varia de geometria para geometria, e os limites da zona de desenho vão alterando da mesma forma, optou-se por se iniciar o desenho, pelo único ponto fixo da zona, o ponto intermédio da zona A. É neste ponto intermédio, que se inicia então o desenho, e é a partir dele que se vão desenhando os hexágonos até aos respetivos limites da zona A (Figura 111).

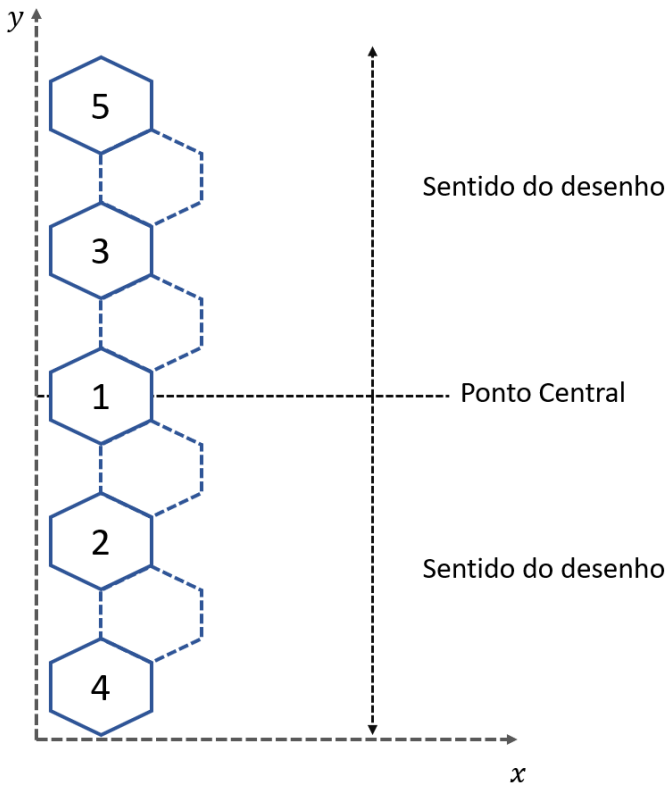


Figura 111 - Sentido do desenho (Zona A)

Definido o ponto de início do desenho, existe agora a questão de existirem dois tipos de geometrias diferentes, que necessitam de funções de cálculo diferentes para a determinação das coordenadas dos centros de cada um dos hexágonos. Estes dois tipos, de geometrias provêm do facto de o número de hexágonos que pode ser colocado segundo a linha vertical da zona A, poder ser par ou ímpar. Como se pode presenciar na Figura 112 e Figura 113, apesar de se poder determinar a posição segundo o eixo horizontal da mesma forma para as duas geometrias, o mesmo não pode ser dito para a posição vertical. Ao tomar-se o ponto de partida como o ponto central, para se chegar ao valor da coordenada y, no caso par, é necessário subtrair (ou somar) ao valor do ponto intermédio, o valor de meia largura de um hexágono e meia altura, para os primeiros dois hexágonos, sendo necessário subsequentemente de adicionar ou remover o valor de uma geometria (*geometry_height*)(equação (23)). Para o ímpar basta apenas subtrair-se ou somar-se o valor de uma geometria (equação (24)).

$$hex\_y\_length = \frac{A}{2} \pm (pair\_index\_A - 0.5) * geometry\_height \quad Se, hex\_vertical\_number\_A\ for\ Par \tag{23}$$

$$hex\_y\_length = \frac{A}{2} \pm (pair\_index\_A - 1.0) * geometry\_height \quad Se, hex\_vertical\_number\_A\ for\ Ímpar \tag{24}$$

onde,

- hex_y_length –"Coordenada vertical do hexágono" (mm)
- $pair\_index\_A$ –"Índice do respetivo par de hexágonos"

Quanto às coordenadas horizontais, estas são iguais para todos os hexágonos do eixo de simetria, mas continuam a estar dependentes da geometria, pois o espaçamento, para além de contabilizar com a largura do hexágono, tem de também ter em atenção a sua espessura, para que esta não saia dos limites. Ou seja, a coordenada horizontal é calculada da seguinte forma (equação (25)):

$$hex\_x\_length = \frac{hex\_size + hex\_thickness}{2} \tag{25}$$

onde,

- hex_x_length –"Coordenada horizontal do hexágono" (mm)
- hex_size –"Largura entre duas arestas paralelas do hexágono" (mm)

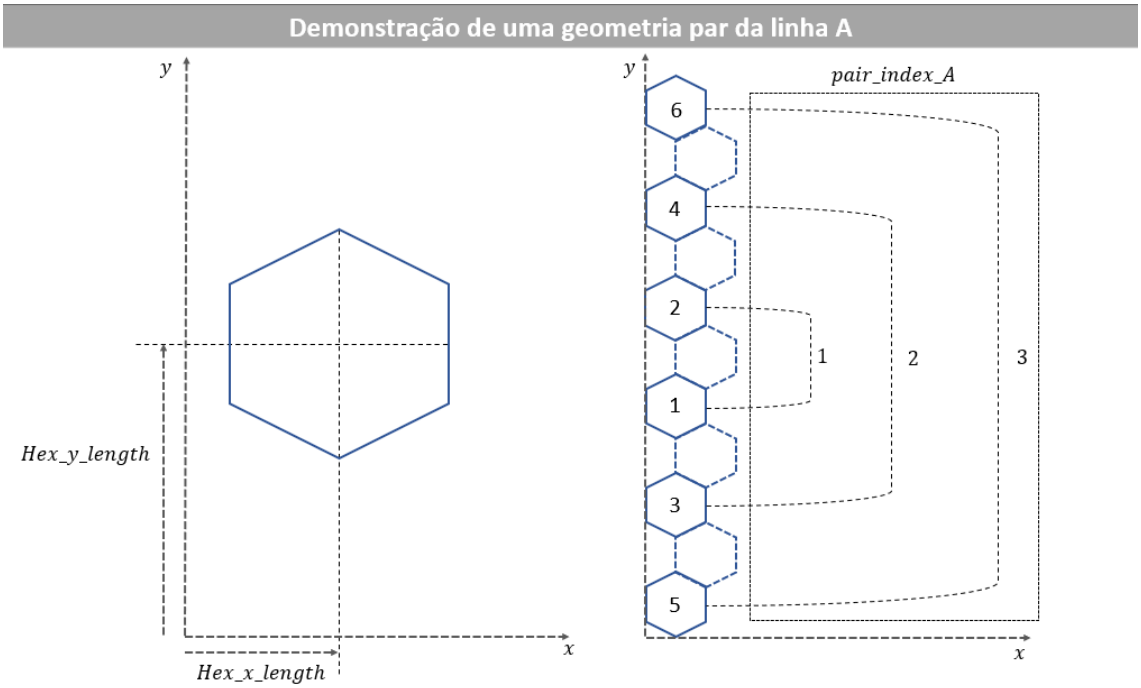


Figura 112 - Demonstração do algoritmo da zona A (Par)

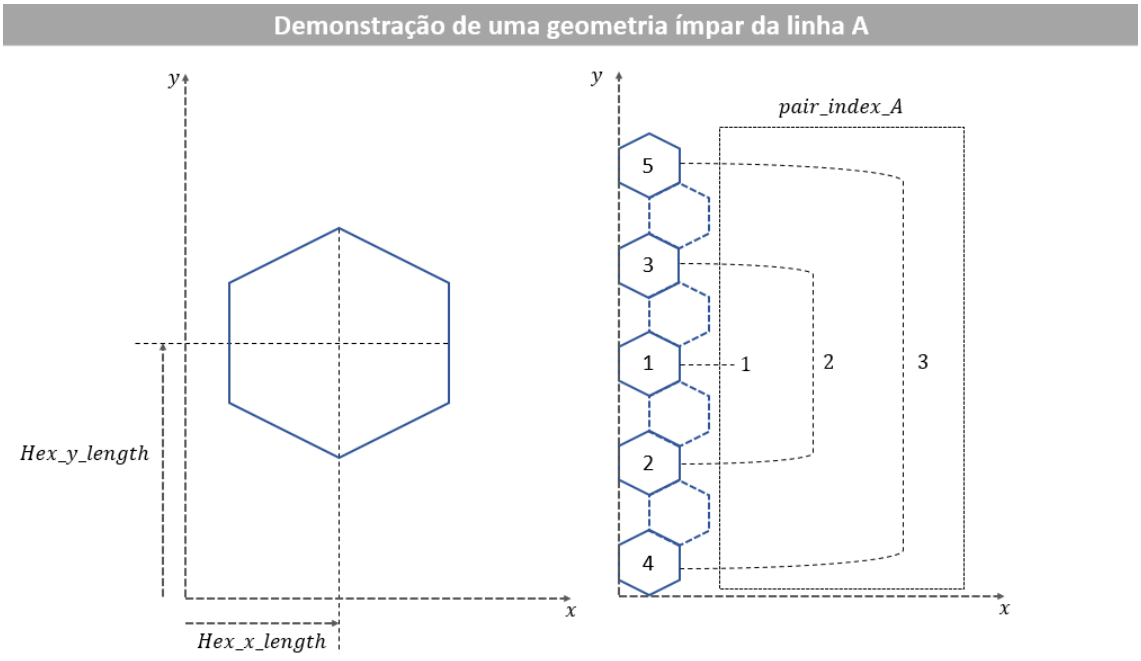


Figura 113 - Demonstração do algoritmo da zona A (Ímpar)

Para além do cálculo das coordenadas de cada hexágono, como já foi dito, é necessário que estes sejam desenhados de forma intercalada, de maneira que o Abaqus® os consiga extrudir. Para isto, criou-se uma variável chamada de *pair_index_A*, que representa a numeração de cada um dos pares de hexágonos, que são formados por um hexágono, e o seu respetivo simétrico, pela linha horizontal que passa no ponto central. Através deste número, para além de se tornar possível o cálculo das coordenadas de cada par de hexágonos, facilita-se também a aplicação das condições que determinam quais os hexágonos que têm de ser desenhados em cada passagem do desenho. Como se pode ver na Figura 114, no caso da geometria par, na primeira passagem do algoritmo, é correto afirmar que cada vez que o índice de um par de hexágonos é impar, o hexágono que é desenhado é o inferior, do respetivo par, enquanto que na segunda passagem, estes são desenhados apenas quando o índice é par, nas outras restantes condições desenha-se o hexágono superior do respetivo par.

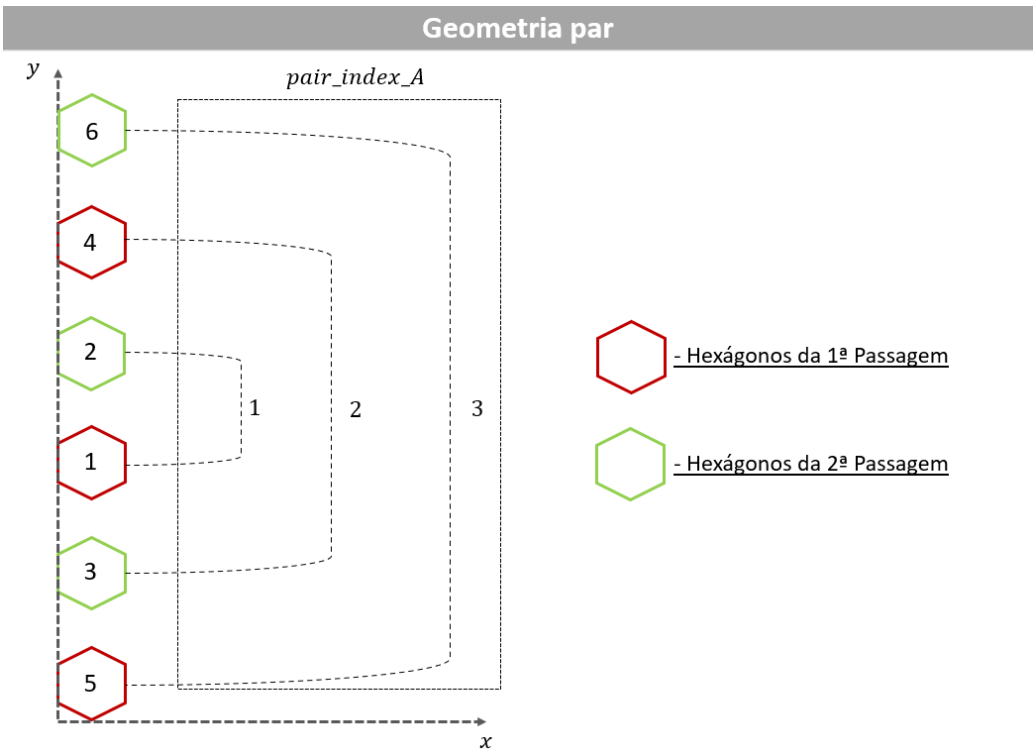


Figura 114 - Demonstração da sequência intercalar da geometria par

No que diz respeito à geometria ímpar, para se construir um desenho intercalar, apenas basta afirmar que na primeira passagem o valor inicial do *pair_index_A* é um, e o incremento desta variável é feito de duas em duas unidades, levando a que sejam construídos apenas os hexágonos de índices ímpares, sem que seja necessário recorrer a uma função condicional dentro do *loop* do programa. Na segunda passagem apenas se considera que o valor inicial é dois e o resto do processo é o mesmo (Figura 115).

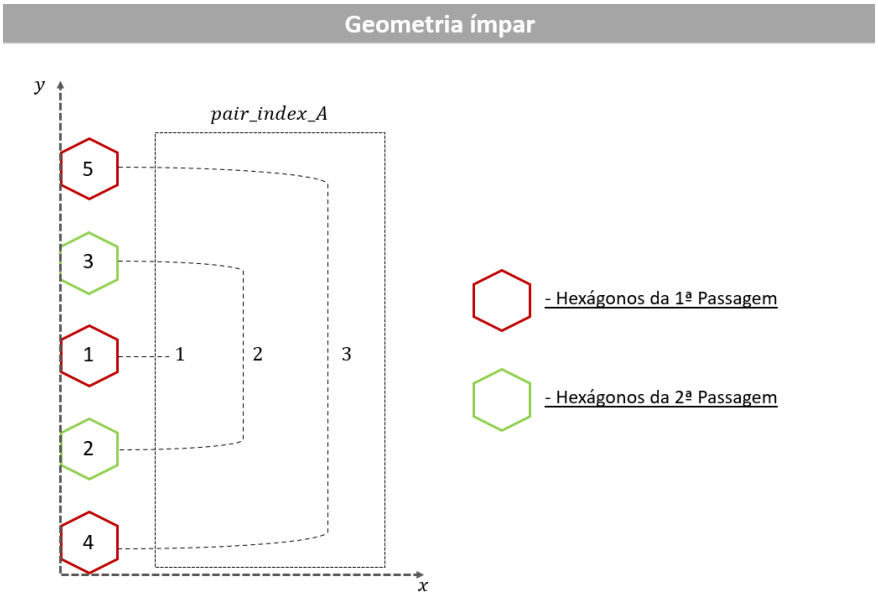


Figura 115 - Demonstração da sequência intercalar da geometria ímpar

Com os hexágonos do eixo de simetria desenhados, faltam agora as diagonais. Ao contrário dos verticais o processo de cálculo destes hexágonos é igual para todas as geometrias, e, portanto, dentro do código este vai ser gravado numa função à parte, de modo a poder ser chamado para qualquer parte do programa. Tal como anteriormente, o primeiro passo deste processo, é calcular quantos hexágonos é que são necessários desenhar, e, portanto, o programa tem de calcular qual é o espaço disponível para o desenho, e quanto espaço é ocupado por cada um dos hexágonos (equação (26)).

$$hex\_diagonal\_number = \frac{Largura\ disponível}{hex\_size} \tag{26}$$

onde,

- $hex\_diagonal\_number$ —"Número de hexágonos a desenhar na diagonal"
- $Largura\ disponível$ —"Largura disponível para desenhar os hexágonos diagonais" (mm)

A largura disponível, no contexto deste problema é obtida através do que se designou de altura disponível. Observando a representação gráfica deste problema na Figura 116, denota-se que a altura disponível é obtida de forma semelhante à altura disponível dos hexágonos verticais, contudo neste caso em específico, esta é obtida removendo-se o valor de apenas uma das alturas ocupadas pela espessura, o valor da altura de um hexágono e o valor da altura do hexágono vertical que dá início à linha diagonal (equação (27)). A altura do hexágono vertical não é necessário calcular, porque este valor já é proveniente do cálculo dos hexágonos verticais (*hex_y_length*).

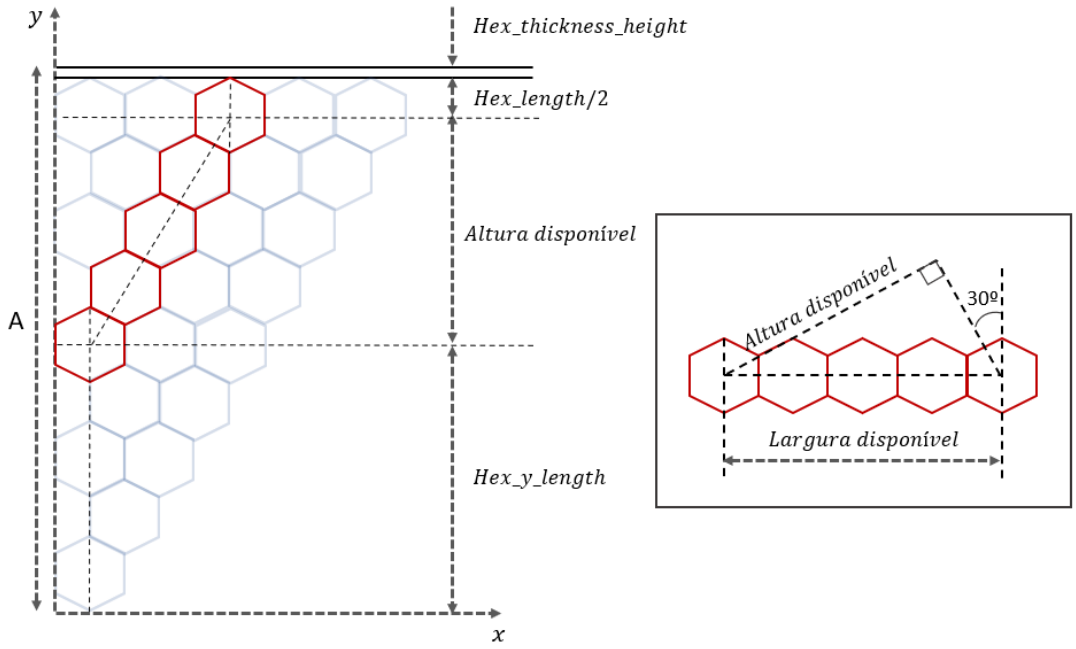


Figura 116 - Demonstração do cálculo do número de hexágonos diagonais

$$Altura\ disponível = A - hex\_y\_length - hex\_thickness\_height - \frac{hex\_length}{2} \quad (27)$$

onde:

- $Altura\ disponível$ – "Altura disponível para desenhar os hexágonos" (mm)

Por sua vez este valor é utilizado para calcular a largura disponível, que através do triângulo representado na Figura 116, é obtido através da seguinte relação trigonométrica (equação (28)):

$$Largura\ disponível = \frac{Altura\ disponível}{\cos(30^o)} \quad (28)$$

Substituindo tudo na equação (26) , obtém-se a equação (29).

$$hex\_diagonal\_line = \frac{A - hex\_y\_length - hex\_thickness\_height - \frac{hex\_length}{x}}{hex\_size * \cos(30^o)} \quad (29)$$

Com o número de hexágonos calculado, resta calcular as coordenadas de cada um deles. Como os hexágonos encontram-se lado a lado, e seguem uma trajetória que faz

um ângulo de 60º com a horizontal, a variação que cada um sofre em cada uma das suas componentes é igual às equações (30) e (31) (Figura 117):

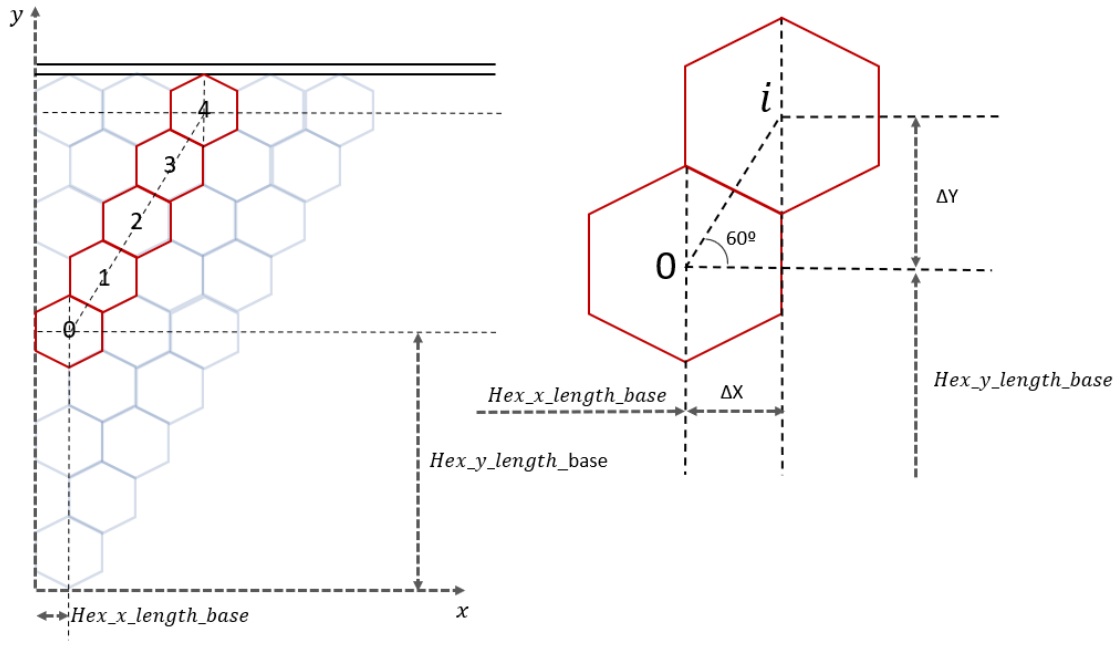


Figura 117 - Demonstração do cálculo das coordenadas dos hexágonos diagonais

$$hex\_x\_length = hex\_x\_length\_base + i * \Delta x \quad (30)$$

$$hex\_y\_length = hex\_y\_length\_base + i * \Delta y \quad (31)$$

onde,

$$\Delta x = hex\_size * \cos(60) \quad (32)$$

$$\Delta y = hex\_size * sen(60^{o}) \quad (33)$$

- $hex\_x\_length\_base$ – "Coordenada horizontal do hexágono vertical que dá origem à sequência diagonal" (mm)
- $hex\_y\_length\_base$ – "Coordenada vertical do hexágono vertical que dá origem à sequência diagonal" (mm)
- Δx – "Variação da coordenada horizontal em cada ciclo" (mm)
- Δy – "Variação da coordenada vertical em cada ciclo" (mm)
- i – "Índice do friso"

Ao contrário dos hexágonos verticais, nem todos os diagonais são necessários para desenvolver a estrutura completa dos frisos. Para se diminuir a complexidade da formulação da estrutura, desenvolveu-se uma metodologia para suprimir os desnecessários. Como se pode ver na Figura 118, uma maneira de se abordar este problema é incrementando o índice do hexágono diagonal, de dois em dois valores, contudo, em certos casos específicos, utilizar esta abordagem por si só pode gerar problemas. Na zona A, os problemas que podem surgir são:

1. O hexágono de índice 1 tem sempre de ser desenhado, caso contrário uma das faces necessárias para a estrutura não será desenhada;
2. O último hexágono da sequência tem sempre de ser desenhado, independentemente do valor do seu índice, visto que este nunca terá hexágonos suficientes à sua volta para gerar as faces todas;

Contornar o primeiro problema é relativamente simples, basta considerar o primeiro valor do índice do hexágono (*i*), como sendo igual a 1, e incrementando o seu valor, de dois em dois. O segundo problema, já é mais complexo, para que o último hexágono seja sempre desenhado, o programa quando detetar que o índice corresponde ao penúltimo da lista, em vez de incrementar o valor do índice em dois valores, incrementa em um. Para que o programa consiga detetar o penúltimo valor, a seguinte metodologia foi implementada dentro do código (equação (34) e (35)):

$$i = i + 2 \qquad Se, i \neq hex\_diagonal\_number - 1 \tag{34}$$

$$i = i + 1 \qquad Se, i = hex\_diagonal\_number - 1 \tag{35}$$

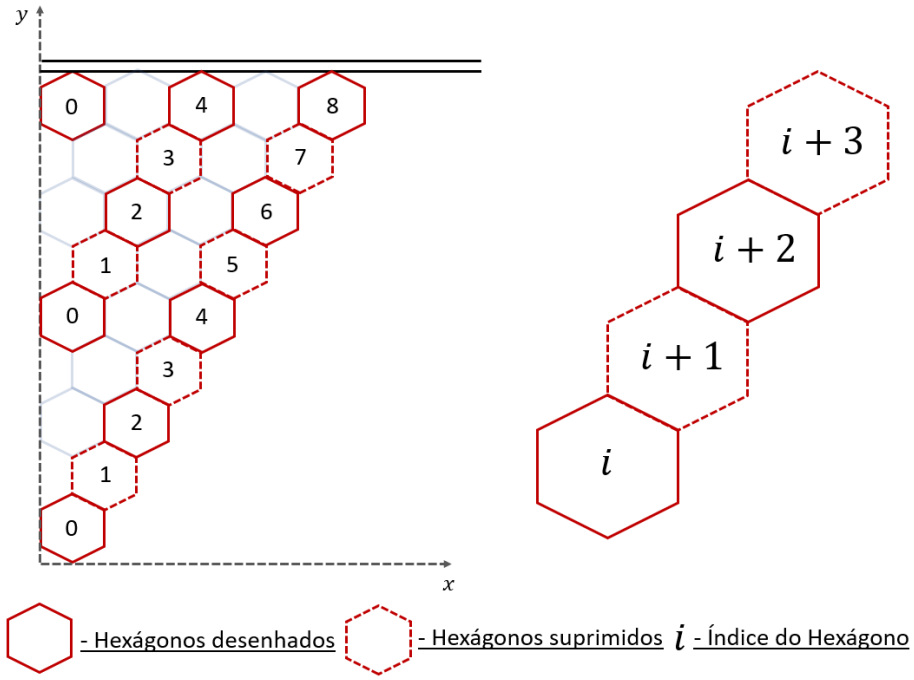


Figura 118 - Demonstração da supressão de hexágonos desnecessários (sem correções)

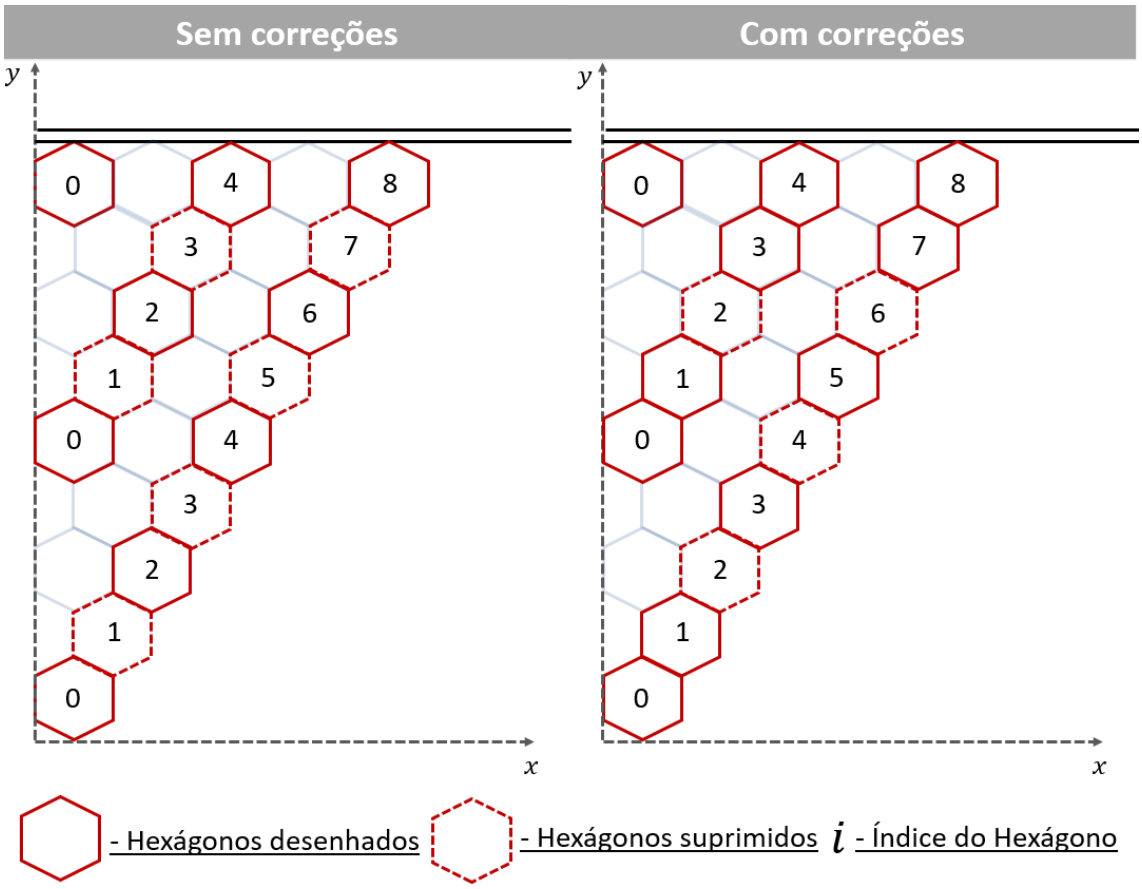


Figura 119 - Demonstração da supressão de hexágonos desnecessários (Com correções)

O processo de se incrementar o índice de dois em dois, permite suprimir todos os hexágonos desnecessários, mas não pode ser aplicado em todas as linhas diagonais, tal como o processo de extrusão este tem de ser feito intervaladamente entre as linhas. Ou seja, enquanto que em certas linhas o incremento do índice é feito de dois em dois valores, na linha seguinte o processo terá de ser feito com incrementos de um (Figura 120). Como o processo de se realizar todos hexágonos de uma diagonal coincide com o intercalar da extrusão, simplesmente pode-se impor às condições das equações (34) e (35), que na primeira etapa de desenho, o incremento será sempre feito com o valor de um, e na segunda etapa o incremento será o dobro, exceto nas condições mencionadas no parágrafo anterior, resultando assim nas condições das equações (36) e (37).

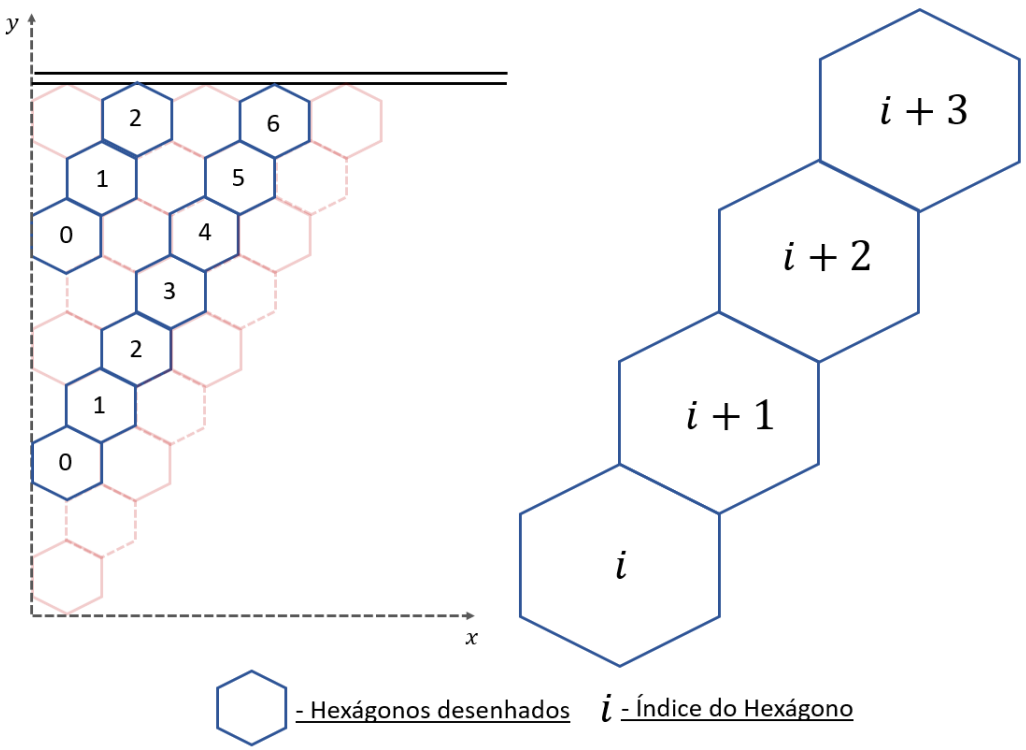


Figura 120 - Demonstração de linhas diagonais com incremente de um valor

$$i = i + 2 \qquad Se, i \neq hex\_diagonal\_number\_1 \tag{36}$$

$$i = i + 1 \qquad Se, i = hex\_diagonal\_number - 1, ou\ first\_\text{int} = 1 \tag{37}$$

onde,

- $first\_int$ – "Etapa do desenho"

3.5.3.6.2 Algoritmo de desenho da Zona A – Código Python

Com a metodologia teórica tratada é então necessário passá-la para código Python. A primeira coisa que deve ser feita é impor os pré-requisitos necessários para que o Abaqus® consiga identificar todas as geometrias, pontos de referência e planos necessários para o desenho da geometria (Figura 121). Estes requisitos são os mesmos que foram identificados no Capítulo 3.5.3.5. Dentro destes, o ponto que é utilizado para centrar o desenho ("*Sketch_Center_Point*") tem o seu nome armazenado na variável *center_point* (Figura 121 Linha 6). No caso do sistema de coordenadas e do plano de desenho, como estes se tratam de *features*, não é possível acedê-los diretamente através do nome que lhes foi atribuído, é necessária a *key* que os identifica. De forma a contornar este problema nas linhas 15 e 16, o programa procura qual é a chave que está associada ao nome de cada uma das *features* e, mais tarde, utiliza essa *key* para aceder á geometria e guardá-la numa variável. Com os pré-requisitos tratados e armazenados, o programa chama a função *sketch_start* (Figura 121 Linhas 22 e 23) que inicia o sketch para desenhar os frisos, recebendo como argumentos o plano onde estes devem ser desenhados (*drawing_plane*), o eixo do sistema de coordenadas que orienta o desenho (*align_edge*), a parte onde se deve fazer o desenho (*p*), o nome do desenho (*profile_name*) e o ponto que serve de origem para a grelha de desenho (*center_point*).

```
1 def Create_Hex_Assembly(hex_size, hex_lenght, hex_height,
2     hex_thickness, sketch_name,
3     A,B,C,D,E,F,G,H):
4 
5     # Pre-requesite: Center Point Name
6     center_point = 'Sketch_Center_Point'
7 
8     # Pre-requesite: CSYS Datum that gives the drawing orientation
9     p = mymodel.parts['Base']
10    d3 =  p.datums
11    align_edge_id = p.features['Drawing_CSYS'].id
12    align_edge = d3[align_edge_id].axis2
13 
14    # Pre-requesite: Drawing plane datum
15    drawing_plane_id = p.features['Sketch_Plane'].id
16    drawing_plane = d3[drawing_plane_id]
17 
18    # Profile Name
19    profile_name = '__profile__'
20 
21    # Initiate the Sketch
22    s, g, v = sketch_start(drawing_plane, align_edge, p,
23        profile_name, center_point)
24 
25    # Bollean that checks its the first sketch step
26    sketch_step_1 = True
```

Figura 121 - Função de desenho da geometria dos frisos – Pré-requisitos (*Create_Hex_Assembly*)

Dentro da função *sketch_start* da Figura 122, de maneira a poder-se converter os dados dos pontos de referência do modelo 3D para o desenho bidimensional teve de se criar uma matriz de transformação, através do método *MakeSketchTransform* do objeto *Transform* (Figura 122 Linhas 15-19) [70]. Isto permite criar uma matriz 4x3 que permite transformar coordenadas do sketch em coordenadas do *assembly* ou da *part*, permitindo assim criar um ambiente de desenho, com restrições impostas por características do modelo 3D. Neste caso em específico as restrições que são introduzidas, são o plano de desenho, o eixo de orientação, e as coordenadas do ponto de origem do desenho, que foram obtidas através da repartição do *tuble* gerado pelo método *getCoordinates* quando aplicado no ponto de origem *center_point* (Figura 122 Linhas 6-12). Com a transformação de coordenadas concluída, é então criado o *sketch* com as restrições mencionadas (Figura 122 Linhas 22-25), e tanto o sketch, como os seus membros (*geometry* e *vertices*) são devolvidos à função *Create_hex_assembly*, de modo a poderem ser acedidos por ela.

```
# This function initiates the sketch
def sketch_start(drawing_plane, align_edge, p,
    profile_name, center_point):

    # Get center Point Coordinates
    d3 =  p.datums
    sketch_center_point_id = p.features[center_point].id
    sketch_center_point_coordinates = p.getCoordinates(
        d3[sketch_center_point_id])
    center_x_coordinate = sketch_center_point_coordinates[0]
    center_y_coordinate = sketch_center_point_coordinates[1]
    center_z_coordinate = sketch_center_point_coordinates[2]

    # Create the transformation matrix
    t = p.MakeSketchTransform(sketchPlane= drawing_plane,
        sketchUpEdge= align_edge,
        sketchPlaneSide=SIDE1, sketchOrientation=RIGHT,
        origin=(center_x_coordinate,
        center_y_coordinate, center_z_coordinate))

    # Start Sketch
    s = mymodel.ConstrainedSketch(name=profile_name,
            sheetSize=2460.97, gridSpacing=61.52, transform=t)
    g, v = s.geometry, s.vertices
    p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)

    return( s, g, v )
```

Figura 122 – Inicialização de um sketch (*sketch_start*)

Com o sketch devidamente iniciado passa-se então para o algoritmo de desenho, que começa com a zona A. Como por sua vez esta possui funções de calculo diferentes, conforme a geometria gerada possui um número par ou impar de frisos segundo o eixo de simetria, o algoritmo da zona A está dividido em duas partes, uma para o cálculo de uma geometria par (Figura 123) e outra para a geometria impar(Figura 124).

```
    # Sketching the hexagons through line A
    for first_int in range(1,3):

        # Calculate the number of hexagons that fit in line A
        hex_thickness_height = hex_thickness/(2*np.cos(np.radians(30)))
        available_height = A + hex_lenght - 2*hex_thickness_height
        geometry_height = hex_height + hex_lenght
        hex_vertical_number_A = int(
            math.floor(available_height/geometry_height))

        # Alters the base value of the pair index
        pair_index_A = 1
        if (hex_vertical_number_A%2 != 0) and (first_int == 2):
            pair_index_A = 2

        # Calculate the hexagon coordinates
        while (2*pair_index_A-1) <= hex_vertical_number_A:

            # Calculate the hexagon x coordinate
            hex_x_lenght = hex_size/2 + hex_thickness/2

            # Calculations if the number of hexagons is even
            if hex_vertical_number_A%2 == 0:
                # Conditions to intercalate the sketch
                if (first_int == 1 and pair_index_A%2 != 0) \
                    or (first_int == 2 and (pair_index_A)%2 == 0):
                    # Bottom hexagon y coordinate
                    hex_y_lenght = A/2-(pair_index_A-0.5)*geometry_height
                else:
                    # Top hexagon y coordinate
                    hex_y_lenght = A/2+(pair_index_A-0.5)*geometry_height

                # Sketch the hexagon that's along the vertical line
                draw_hex(hex_x_lenght, hex_y_lenght, s, sketch_name, g, v)

                # Calculate the number of hexagons that fit diagonaly
                hex_diagonal_number = int(math.floor((\
                    (A-hex_y_lenght)/np.cos(np.radians(30)) \
                    -(hex_height/2 + hex_thickness_height)/ \
                    np.cos(np.radians(30)))/hex_size))

                # Sketch the hexagons that fit diagonaly
                draw_diagonal_hex_A(hex_size, hex_y_lenght, hex_x_lenght,
                    first_int,hex_diagonal_number, s,
                    sketch_name, g, v)

            else:
```

Figura 123 – Função de desenho da geometria dos frisos da zona A (Geometria Par)

```
47              else:
48                  # Calculations if the number of hexagones is odd
49                  # Calculate the hexagons y coordinates
50                  const = pair_index_A-1
51                  hex_y_lenght_1 = A/2-(const)*geometry_height
52                  hex_y_lenght_2 = A/2+(const)*geometry_height
53
54                  # Sketch the hexagon that's along the vertical line
55                  draw_hex(hex_x_lenght, hex_y_lenght_1, s,
56                      sketch_name, g, v)
57
58                  # Calculate the number of hexagons that fit diagonaly
59                  hex_diagonal_number=int(
60                      math.floor(((A-hex_y_lenght_1)/ \
61                      np.cos(np.radians(30))-(hex_height/2 + \
62                      hex_thickness_height)/ \
63                      np.cos(np.radians(30)))/hex_size))
64
65                  # Sketch the hexagons that fit diagonaly
66                  draw_diagonal_hex_A(hex_size, hex_y_lenght_1,
67                      hex_x_lenght, first_int,
68                      hex_diagonal_number, s,
69                      sketch_name, g, v)
70
71                  # Sketch the second hexagon
72                  if hex_y_lenght_1 != hex_y_lenght_2:
73
74                      # Sketch the hexagon that's along the vertical line
75                      draw_hex(hex_x_lenght, hex_y_lenght_2, s,
76                          sketch_name, g, v)
77
78                      # Calculate the number of hexagons that fit diagonaly
79                      hex_diagonal_number=int(
80                          math.floor(((A-hex_y_lenght_2)/ \
81                          np.cos(np.radians(30))-(hex_height/2+ \
82                          hex_thickness_height)/ \
83                          np.cos(np.radians(30)))/hex_size))
84
85                      # Sketch the hexagons that fit diagonaly
86                      draw_diagonal_hex_A(hex_size, hex_y_lenght_2,
87                          hex_x_lenght, first_int,
88                          hex_diagonal_number, s,
89                          sketch_name, g, v)
90
91                  pair_index_A += 1
92              pair_index_A += 1
```

Figura 124 – Função de desenho da geometria dos frisos da zona A (Geometria Ímpar)

Antes de se iniciar o algoritmo de cálculo para cada uma das geometrias, existe uma parte do programa que é comum às duas. Nesta porção de código comum, são então impostos os parâmetros dos dois ciclos existentes dentro do algoritmo, um que rege o número de passo que são utilizados na extrusão (Figura 123 Linha 2), e outro que verifica a cada passagem se o número de hexágonos que foi desenhado já ultrapassou a quantidade que deve de ser desenhada (Figura 123 linha 17). Para além destes dois tem-se também o cálculo do número hexágonos que devem ser desenhados segundo o eixo de simetria, como demonstrados nas equações (19), (20), (21) e (22) (Figura 123 Linhas 6-9), a alteração do valor inicial do índice de par (*pair_index_A*) (Figura 123 Linhas 12-14) e o seu incremento a cada ciclo (Figura 124 Linhas 91-92), que permite que o desenho seja feito de forma intercalada, e por fim o cálculo da coordenada horizontal, que é comum a todos os hexágonos, e segue a formula da equação (25).

Dentro dos componentes demonstrados na Figura 123 pode-se presenciar na linha 17 que o critério que leva à verificação do número de hexágonos já desenhados, é se:

$$2 * pair\_index\_A - 1 \leq hex\_vertical\_number\_A \tag{38}$$

Ora, se o *pair_index_A* representa o número de um respetivo par de hexágonos, faria sentido afirmar que o número de hexágonos desenhados até ao momento seria igual a duas vezes o valor do índice. Contudo, como o algoritmo trata de duas geometrias distintas, uma par e outra ímpar, isto apenas seria verdade para a geometria par. Na geometria ímpar, como existe um hexágono que não tem um par respetivo, deve ser descontado um valor ao dobro do número de índices (Figura 125). Como as duas metodologias possuem duas fórmulas de cálculo diferentes, então a função condicional da Figura 123 linha 17, deveria de ter duas condições, uma para cada geometria, onde as metodologias de cálculo seriam as duas mencionadas. Porém, como na geometria par, são sempre desenhados dois hexágonos, pode-se utilizar o critério da geometria ímpar na par. Por exemplo, caso o número de hexágonos a gerar fosse igual a seis, ao chegar-se ao ponto em que o valor de *pair_index_A* fosse igual a três, utilizando a equação (38), o número de hexágonos seria igual a cinco, que, como é inferior ao número de hexágonos a desenhar pode prosseguir com o desenho dos hexágonos do respetivo par (Hexágonos 5 e 6), com o acréscimo de *pair_index_A* para quatro, o número de hexágonos passaria agora para oito, deixando assim de satisfazer a condição necessária para desempenhar o *loop*. Assim, apesar de o valor calculado para o número de hexágonos não ser fidedigno da realidade, o *loop* para na mesma etapa, permitindo desenhar o número de hexágonos correto. Com isto escusa-se de se utilizar duas funções condicionais na Figura 123 linha 17, e a velocidade de resolução é aumentada.

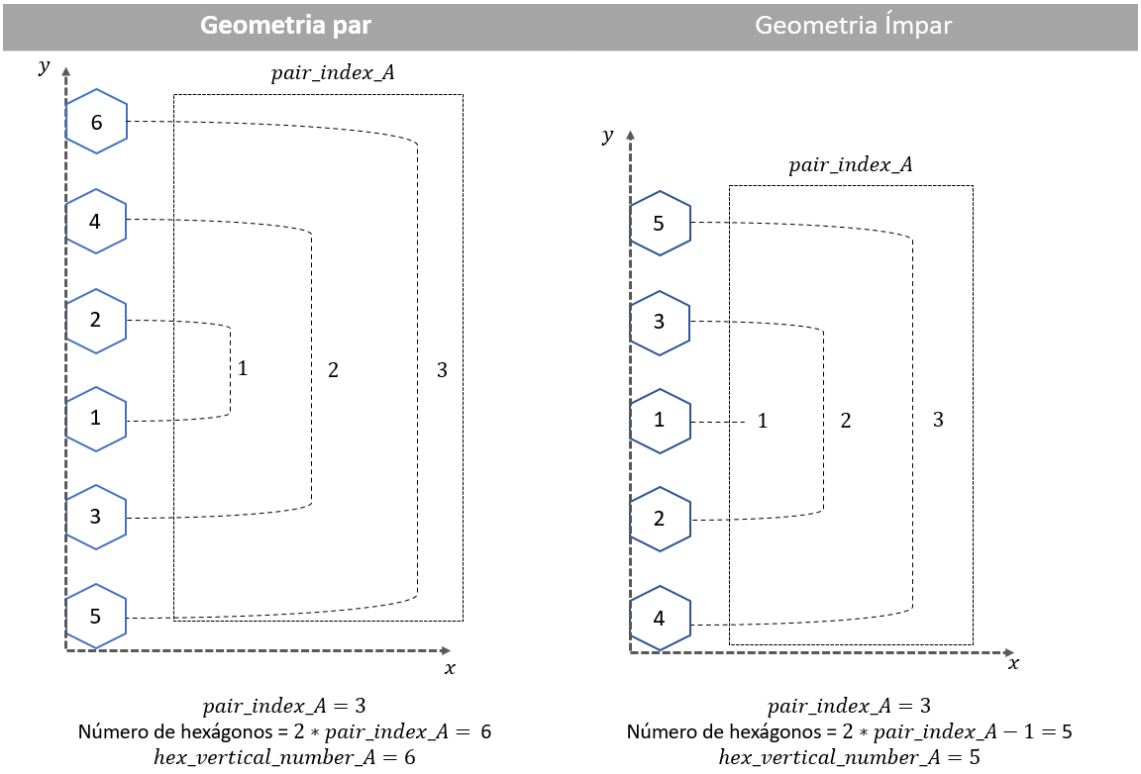


Figura 125 - Demonstração do cálculo do número de hexágonos desenhados

Dentro do algoritmo da zona A tem-se também as distinções entre algoritmo da geometria par e algoritmo da geometria ímpar. Para a geometria par, o programa divide-se em três etapas:

a) Cálculo da coordenada y do hexágono da geometria par – nesta parte são calculadas as coordenadas do hexágono superior ou inferior de um respetivo par, recorrendo à equação (23) e as condições que levam a cada uma (Figura 123 Linhas 23-31);
b) Função responsável pelo desenho do hexágono nas coordenadas calculadas – esta função recebe as coordenadas de um determinado hexágono, e através das funcionalidades do API (*Application Programming Interface*) do Abaqus® cria um hexágono nas respetivas coordenadas, do *sketch* que foi aberto anteriormente (Figura 123 Linhas 34);
c) Cálculo do número de hexágonos diagonais e a função responsável pelo seu desenho – nesta etapa são então calculados o número de hexágonos que se devem produzir diagonalmente (equação (29)), e depois tanto este valor, como as coordenadas do hexágono base são atribuídas sobre a forma de argumentos a uma função que trata de desenhar os hexágonos diagonais, tomando a precauções necessárias (Figura 123 Linhas 37-45);

Para o cálculo da geometria ímpar, como a abordagem teórica do algoritmo é diferente, o programa também é organizado de forma diferente:

a) Cálculo das coordenadas y de um par de hexágonos na geometria ímpar – tal como na geometria par, nesta primeira secção são calculadas as componentes verticais das coordenadas dos hexágonos a desenhar. A diferença que este tem quando comparado com o anterior, é que em vez de se calcular as propriedades de apenas de um dos hexágonos de um respetivo par, aqui podem ser calculados e desenhados os dois ao mesmo tempo, pois como a geometria é ímpar, hexágonos de um mesmo par, vão corresponder a uma mesma passagem do algoritmo, a equação usada nesta parte do programa é a equação (24) deduzida anteriormente (Figura 124 Linhas 50-52);
b) Desenho do hexágono inferior, e a sua respetiva linha diagonal – nesta parte do programa as propriedades do hexágono vertical inferior, são então enviadas para a função *draw_hex*, que se responsabiliza por transmitir a informação para o Abaqus®, e desenhar um hexágono na respetiva posição, depois são calculados e desenhados os hexágonos da respetiva linha diagonal (Figura 124 Linhas 55-69);
c) Desenho do hexágono inferior, e a sua respetiva linha diagonal – com o hexágono inferior desenhado, é necessário desenhar o superior do par. O processo de desenho é exatamente o mesmo que na Figura 124 linhas 55-69, mas agora existe uma condição de que o hexágono superior só pode ser desenhado, se as ordenadas dos dois membros do par forem diferentes. Esta condição está aqui, para permitir que, quando for necessário desenhar o hexágono central, como este não tem um hexágono que lhe sirva de par, e o programa foi feito para desenhar dois hexágonos de uma vez, este segundo desenho não acontece (Figura 124 Linhas 72-89). O hexágono central é facilmente identificável, pois como o seu índice de par corresponde ao valor 1, introduzindo isto na equação de cálculo da coordenada vertical resulta em:

$$pair\_index\_A = 1 \tag{39}$$

$$const = pair\_index\_A - 1 = 0 \tag{40}$$

$$hex\_y\_length\_1 = \frac{A}{2} - const * geometry\_height = \frac{A}{2} \tag{41}$$

$$hex\_y\_length\_2 = \frac{A}{2} + const * geometry\_height = \frac{A}{2} \tag{42}$$

onde:

- $hex\_y\_length\_1$ – "Coordenada vertical do hexágono inferior de um respetivo par" (mm)
- $hex\_x\_length\_2$ – "Coordenada vertical do hexágono superior de um respetivo par" (mm)

Para o desenho das linhas diagonais, ao longo do algoritmo da zona A é chamada a função draw_*diagonal_hex_A*. Esta função recebe as propriedades da linha diagonal, como o número de hexágonos a desenhar, o tamanho do hexágono, as coordenadas do hexágono de base e as características do sketch do Abaqus® que deve ser alterado, como argumentos, e trata de desenhar a respetiva linha, respeitando os critérios necessários para que a geometria formulada seja o mais simples possível (Figura 126).

```
# This function draws the diagonal hexagons that start at
# line A of the hexagone area
def draw_diagonal_hex_A(hex_size, hex_y_lenght_base,
    hex_x_lenght_base, first_int, hex_diagonal_number, s,
    sketch_name, g, v):

    i = 1
    while i <= hex_diagonal_number:

    # Calculation of the horizontal and
    # vertical coordinates of the hexagon
        hex_x_lenght = hex_x_lenght_base + \
            i * hex_size * np.cos(np.radians(60))

        hex_y_lenght = hex_y_lenght_base + \
            i * hex_size * np.sin(np.radians(60))

    # Sketch the hexagon on the calculated coordinates
        draw_hex (hex_x_lenght, hex_y_lenght, s, sketch_name, g, v)

    # This part of the functions makes it so that hexagons
    # that aren't necessary for the drawing are skipped
    # resulting in a simpler computacional model
        if (i == hex_diagonal_number-1) or (first_int == 1):
            i += 1
        else:
            i += 2
```

Figura 126 – Cálculo da posição e desenho dos hexágonos diagonais (*draw_diagonal_hex_A*)

A função draw_diagonal_hex_A, tal como todas as outras funções está então dividida em diversas partes:

a) Imposição do valor inicial – Nestas duas linhas é imposto o valor inicial do índice como sendo igual a 1, para se evitar que o primeiro hexágono seja suprimido (Figura 126 Linhas 7-8);
b) Cálculo das coordenadas do hexágono e desenho – aqui são calculadas as coordenadas dos hexágonos verticais através das equações (30) e (31), deduzidas anteriormente, sendo depois desenhados os hexágonos no Abaqus® (Figura 126 Linhas 12-19);
c) Argumento condicional para suprimir hexágonos – este argumento, corresponde ao argumento lógico apresentado em (36) e (37), que permite que os hexágonos desnecessários para a estrutura sejam suprimidos (Figura 126 Linhas 24-27);

Para além da *draw_diagonal_hex_A* outra função que é muito utilizada ao longo do programa, é a função *draw_hex*. *draw_hex* permite transmitir toda a informação que é obtida através dos cálculos das propriedades do hexágono, para uma janela de *sketch*, de forma a desenhar o respetivo hexágono na posição correta da peça. De uma maneira geral, o que a função faz é o seguinte:

1. Recebe as coordenadas onde é necessário colocar um hexágono, e em que instancia de *sketch* é que se deve desenhar (Figura 127 Linha 3);
2. Vai buscar o *sketch* do hexágono que está guardado em memória e trá-lo para a janela de *sketch* recebida (Capítulo 3.5.3.4) (Figura 127 Linha 6);
3. Faz uma translação do *sketch* do hexágono, para as coordenadas que recebeu como argumento (Figura 127 Linha 9-17);

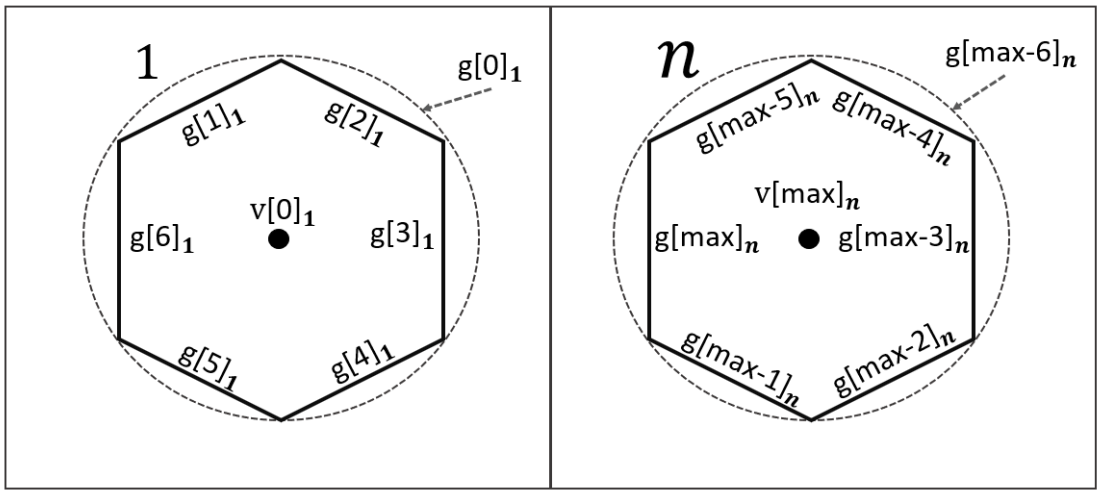
```
# This function receives a set of coordinates
# and draws a hexagon on that same point
def draw_hex (hex_x_lenght, hex_y_lenght, s, sketch_name, g, v):

    # Retrive the hexagon sketch that was saved
    s.retrieveSketch(sketch=mymodel.sketches[sketch_name])

    # Move the hexagon to the respective coordinates
    s.move(vector=(hex_x_lenght, hex_y_lenght),
        objectList=(g[max(g.keys())-6],
        g[max(g.keys())-5],
        g[max(g.keys())-4],
        g[max(g.keys())-3],
        g[max(g.keys())-2],
        g[max(g.keys())-1],
        g[max(g.keys())],
        v[max(v.keys())]))
```

Figura 127 – Desenho de um hexágono da geometria (*draw_hex*)

Para que a função consiga transladar o *sketch* do hexágono para a posição correta, ele tem de conseguir descobrir quais as arestas e vértices do sketch que fazem parte do *sketch* do hexágono que se está a desenhar, no momento em que a função é chamada. Como as *keys* do repositório de geometrias e de vértices são geradas por ordem crescente de valor (Capítulo 3.5.3.4), as *keys* correspondentes ao último hexágono criado, são as *keys* com maior valor absoluto de cada repositório. Como o *sketch* do hexágono tem as *keys* organizadas na forma representada na Figura 128, as *keys* que a função necessita de ir buscar são as últimas 7 do repositório de geometrias e a última do repositório dos vértices. Uma outra questão que tem de ser tida em conta, é o facto de os repositórios não se encontrarem ordenados, o que leva a que a última *key* do repositório, não seja necessariamente a *key* de maior valor (Figura 129). Para tal o programa procura então a *key* com o maior valor absoluto, através da função *max(g.keys())*.



$g[i]_n$ - Repositório de geometria $v[j]_n$ - Repositório de vértices max - Valor máximo do respetivo repositório

Figura 128 – Sketch do hexágono base com as respetivas keys

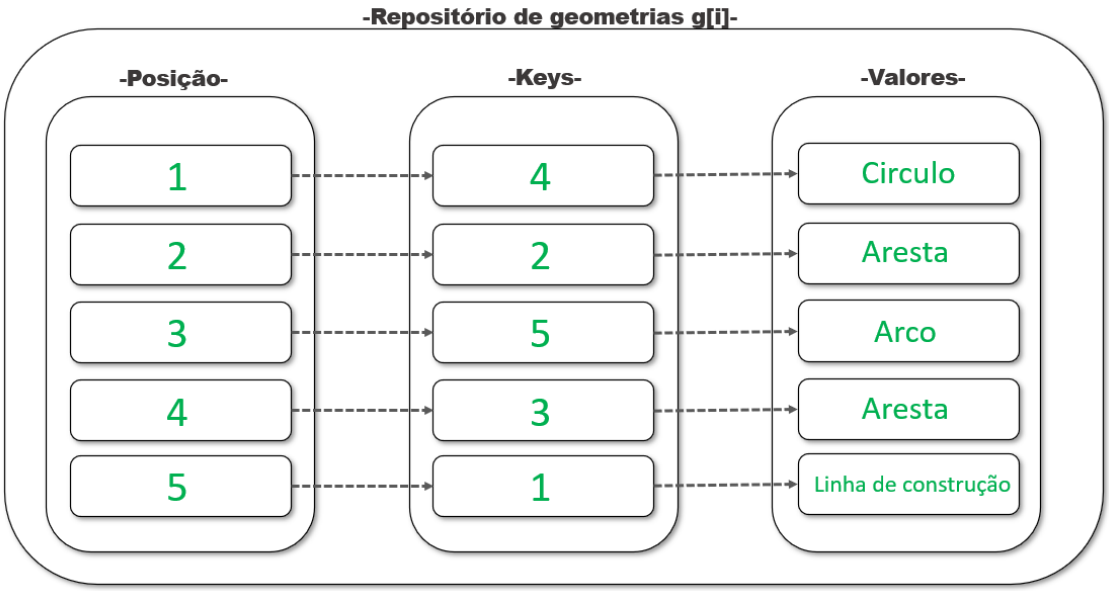


Figura 129 – Representação gráfica da ordenação de um repositório

Todas estas funções formam o algoritmo de desenho para os hexágonos que se encontram na zona A. Este algoritmo pode ser representado pelo fluxograma da Figura 130.

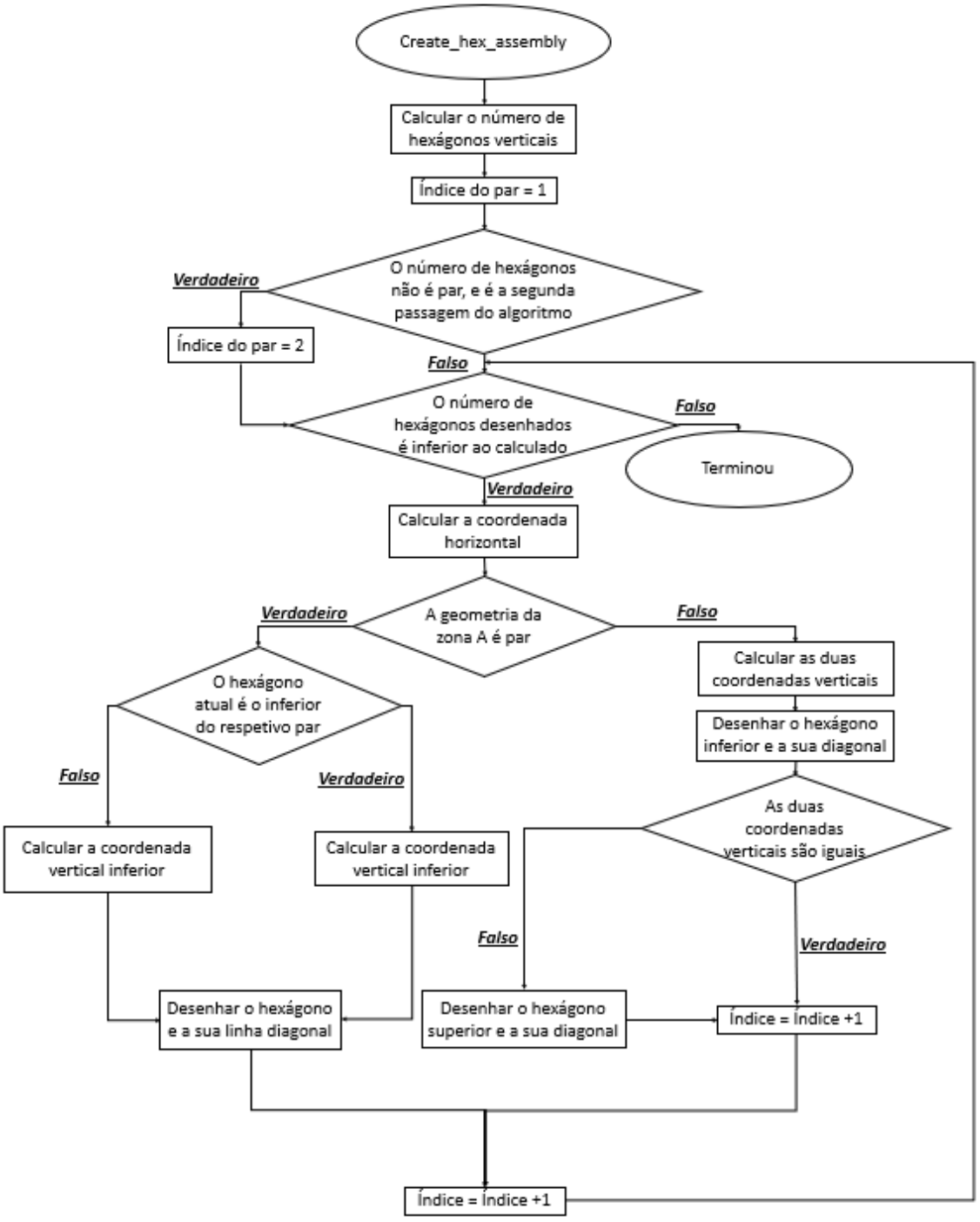


Figura 130 - Algoritmo de desenho da zona A (Fluxograma)

3.5.3.6.3 Algoritmo de desenho da Zona B – Metodologia teórica

No que toca a zona B, o algoritmo de desenho é muito semelhante ao da zona A, sendo que a maior diferença, é que em vez de se desenhar os hexágonos de base ao longo da linha A, estes são produzidos ao longo da linha B (Figura 131).

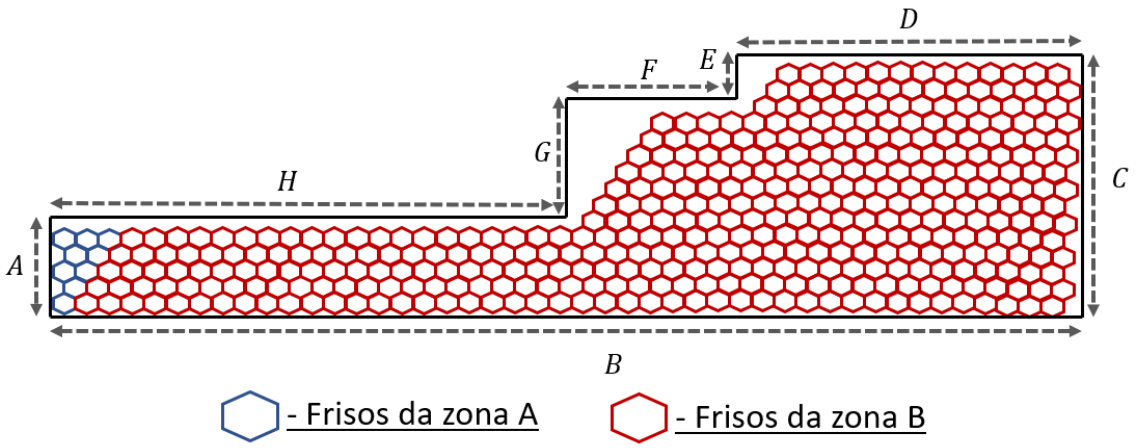


Figura 131 - Ilustração da zona B

Tendo em conta a área representada, o primeiro passo de cálculo, passa por se descobrir quantos hexágonos são possíveis desenhar ao longo da linha B. Tal como na zona A, este valor é calculado aproximando ao valor inteiro mais baixo, o resultado da equação (43).

$$horizontal\_hex\_number\_B = \frac{Largura\ disponível}{Largura\ da\ geometria} \tag{43}$$

onde:

- $horizontal\_hex\_number\_B$ – "Número de hexágonos segundo a linha B"
- $Largura\ disponível$ – "Altura disponível para a colocação dos frisos" (mm)
- $Largura\ da\ geometria$ – "Altura da geometria de referência do friso" (mm)

Neste caso em concreto a largura disponível é obtida removendo duas vezes o valor de meia espessura, à largura da linha B. A remoção destes dois parâmetros advém de ser necessário ter espaço para garantir que as espessuras do primeiro, e do último hexágono da linha, não saiam da zona de desenho dos frisos (Figura 132). A largura da geometria, corresponde apenas à largura de um hexágono, que como já foi visto no algoritmo da zona A, corresponde à variável *hex_size*. Substituindo-se os valores mencionados na equação (43), obtém-se a equação (44).

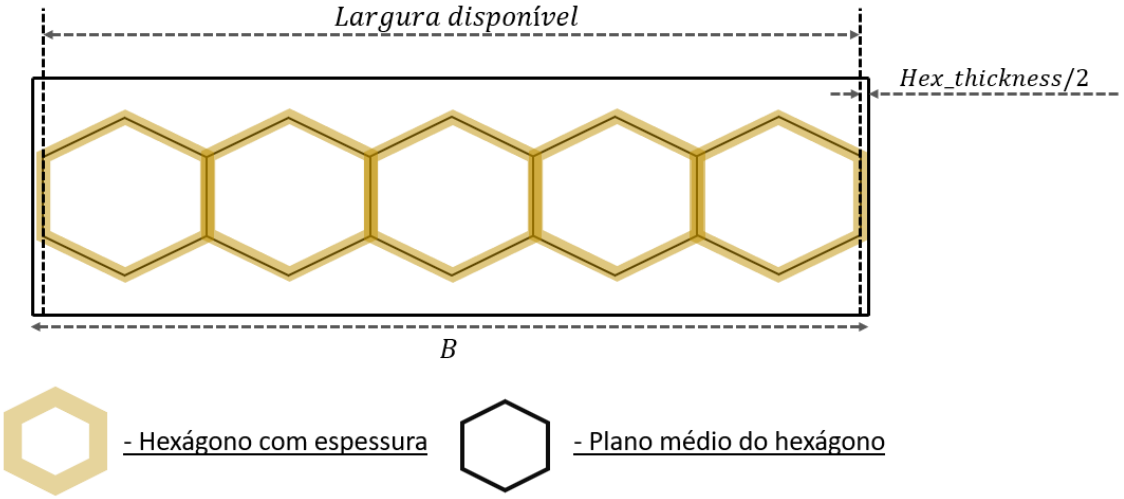


Figura 132 - Ilustração da largura disponível

$$horizontal\_hex\_number\_B = \frac{B - hex\_thickness}{hex\_size} \quad (44)$$

Quanto às coordenadas de cada um destes hexágonos, a coordenada horizontal, é obtida adicionando-se sucessivamente o valor da largura da geometria às condições do primeiro hexágono da fila (equação (45) e equação (46)) (Figura 133), e a vertical é igual à altura do último hexágono inferior da zona A, que pode ser obtido através da equação (47).

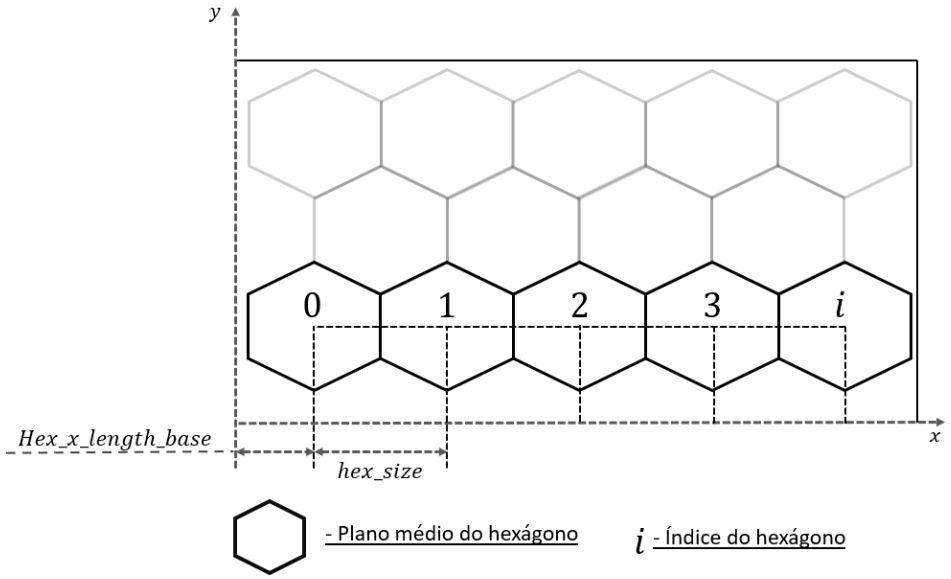


Figura 133 – Ilustração do cálculo da coordenada horizontal

A coordenada horizontal é obtida através de:

$$hex\_x\_length = hex\_x\_length\_base + i * hex\_size \quad (45)$$

onde,

$$hex\_x\_length\_base = \frac{hex\_size + hex\_thickness}{2} \quad (46)$$

A coordenada vertical como corresponde à altura do último hexágono inferior da zona A, este pode ser obtido da seguinte forma:

$$hex\_y\_length = \frac{A}{2} - (hex\_length + hex\_height) * (\frac{vertical\_hex\_number\_A - 1}{2}) \quad (47)$$

A equação representada pode ser utilizada, independentemente da geometria da zona A ser par ou ímpar, pois a componente $(\frac{vertical\_hex\_number\_A-1}{2})$ permite que o cálculo tenha em consideração o valor adicional de meia largura da geometria quando a geometria é par e não o tem em consideração quando é ímpar. Na Figura 134 está demonstrado uma representação visual deste efeito.

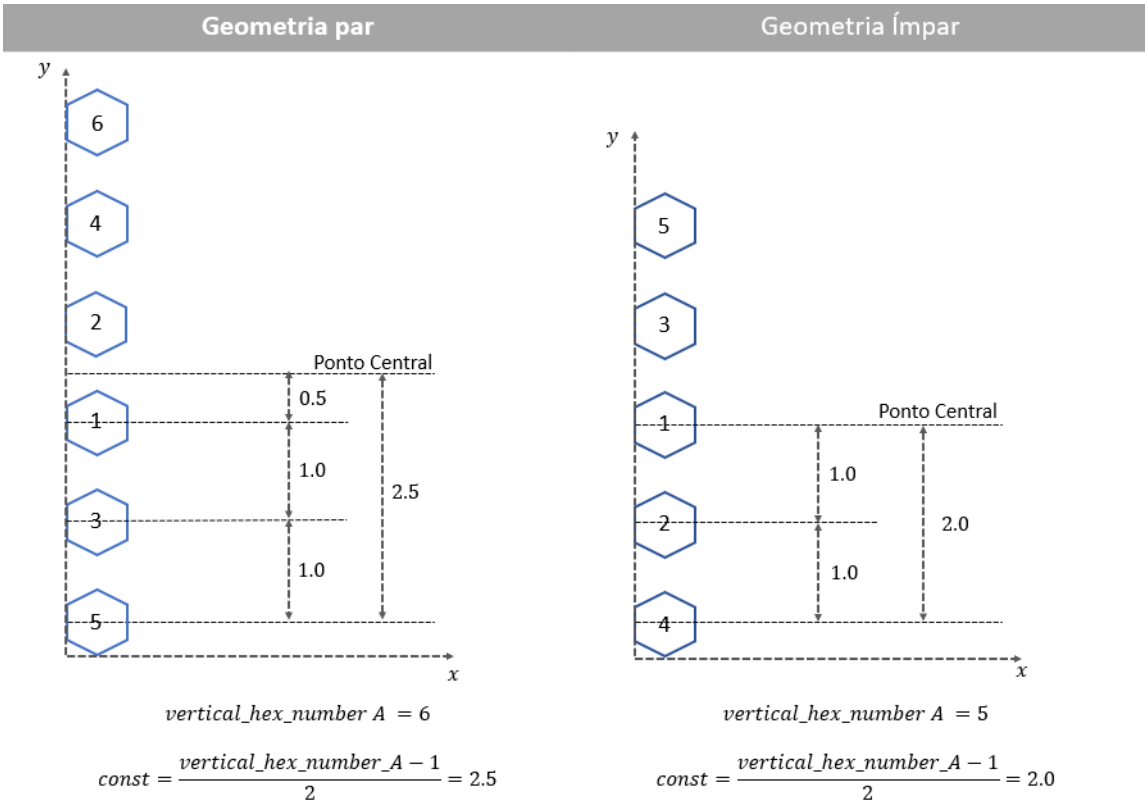


Figura 134 – Ilustração do cálculo do último hexágono da zona A

Tal como no algoritmo da zona A, para que o Abaqus® consiga extrudir o desenho da geometria, este tem de ser desenhado em duas partes, em que em cada uma as linhas diagonais são feitas de forma intercalada. Para se conseguir este efeito na zona B, simplesmente foi imposto que o índice do hexágono é incrementado de dois em dois valores, e que o valor inicial seria diferente conforme os parâmetros dos hexágonos (Figura 135).

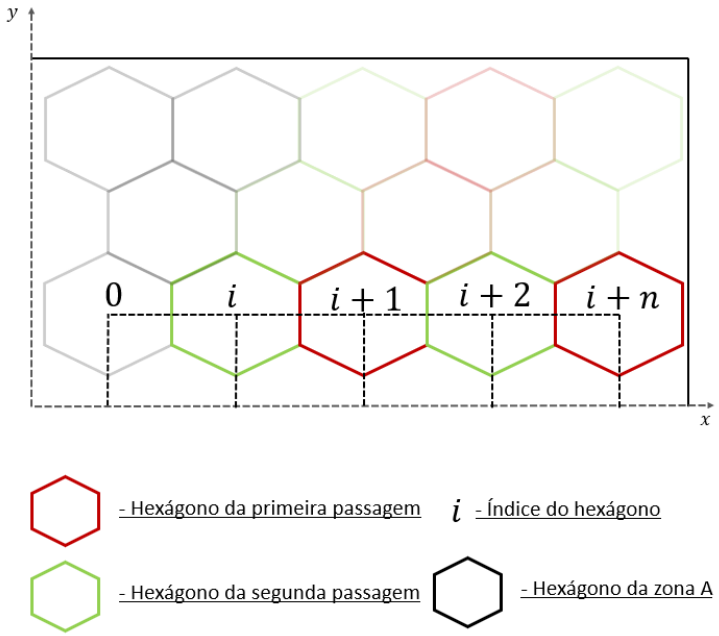


Figura 135 - Desenho intercalar dos hexágonos segundo a linha B

No que toca ao valor inicial, este não possui um valor fixo para cada passagem (ex: 1ª Passagem –> i = 1 e 2ª Passagem –> i = 2), o valor inicial do índice do hexágono, tem de estar em conformidade com a forma como a zona A está a ser desenhada. Este efeito pode ser visto na Figura 136 e na Figura 137, no qual, dependendo da geometria da zona A, do número de hexágonos, e a passagem que se pretende desempenhar, o valor do índice inicial será igual a 1 ou a 2. Tendo-se estudado com mais detalhe esta questão, chegou-se à conclusão que para descrever-se este acontecimento, deve-se de utilizar a seguinte condição lógica (equação (48)):

$$
\begin{aligned}
& i = 1, \\
& Se\ (\mathrm{Pr}imeira\ \ passagem)\ e\ (hex\_vertical\_number\_A\%4 = 0)\ ou \\
& (hex\_vertical\_number\_A\%2 = 0)\ e\ (Segunda\ \ passagem)\ e \\
& (hex\_vertical\_number\_A\%4 \neq 0)\ ou \\
& (hex\_vertical\_number\_A\%2 \neq 0)\ e\ (\mathrm{Pr}imeira\ \ passagem)\ e \\
& ((hex\_vertical\_number\_A-1)\%4 \neq 0)\ ou \\
& (hex\_vertical\_number\_A\%2 = 0)\ e\ (Segunda\ Passagem)\ e \\
& ((hex\_vertical\_number\_A-1)\%4 = 0)
\end{aligned}
\tag{48}
$$

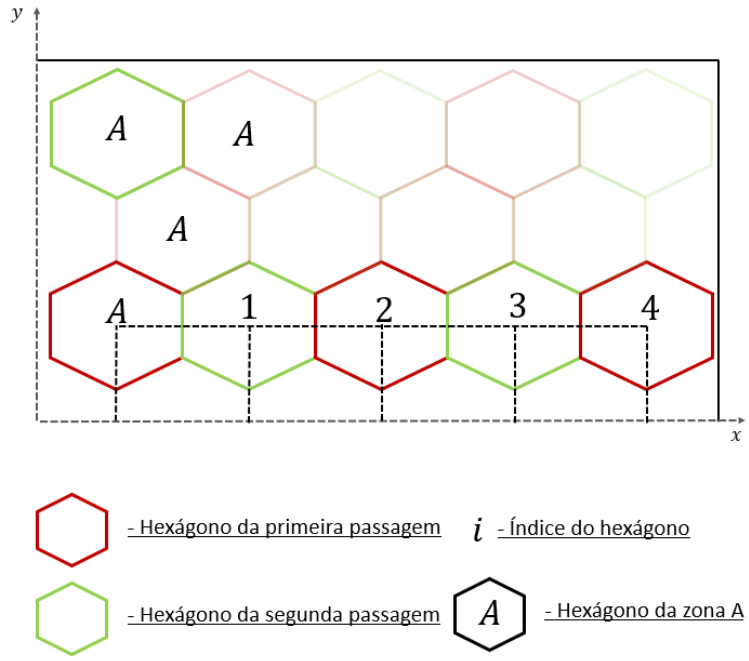


Figura 136 – Demonstração do valor inicial do índice (Geometria par da zona A)

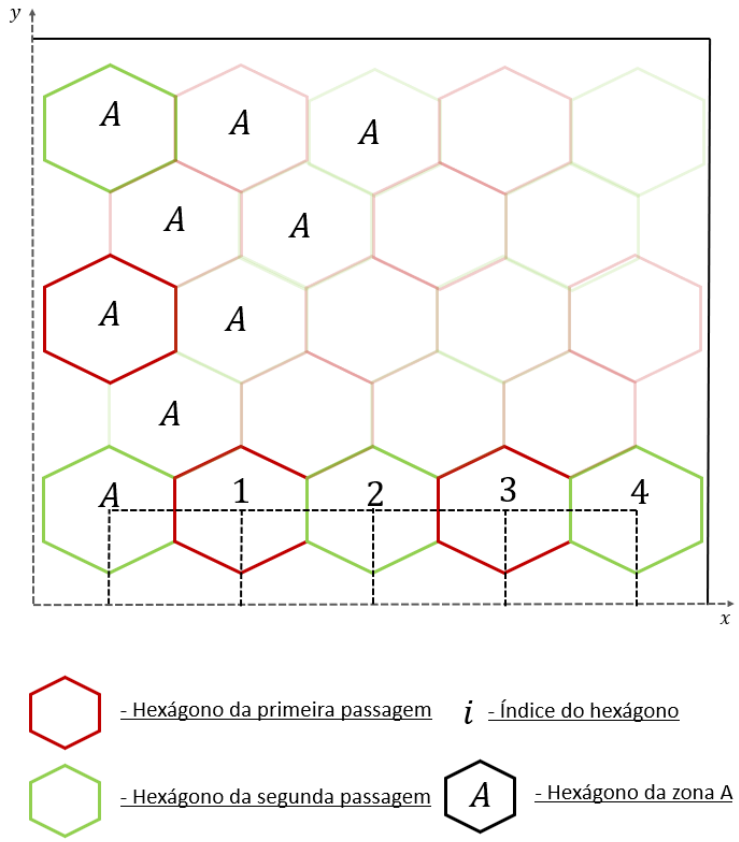


Figura 137 – Demonstração do valor inicial do índice (Geometria par da zona B)

Para o cálculo da linha diagonal de cada um dos hexágonos base, enquanto que na zona A se simplificou o problema, ao considerar-se que a linha H seria a linha limite até ao qual os hexágonos podem ser desenhados, no caso da zona B, dependendo do local onde o hexágono base for desenhado, o limite poderá ser qualquer uma das linhas horizontais que delimitam a área dos frisos (Figura 138). Para se tratar deste problema, o algoritmo da zona B possui então uma função, cujo objetivo é o de descobrir qual deve ser a linha limite que deve ser utilizada para desenhar a linha diagonal.

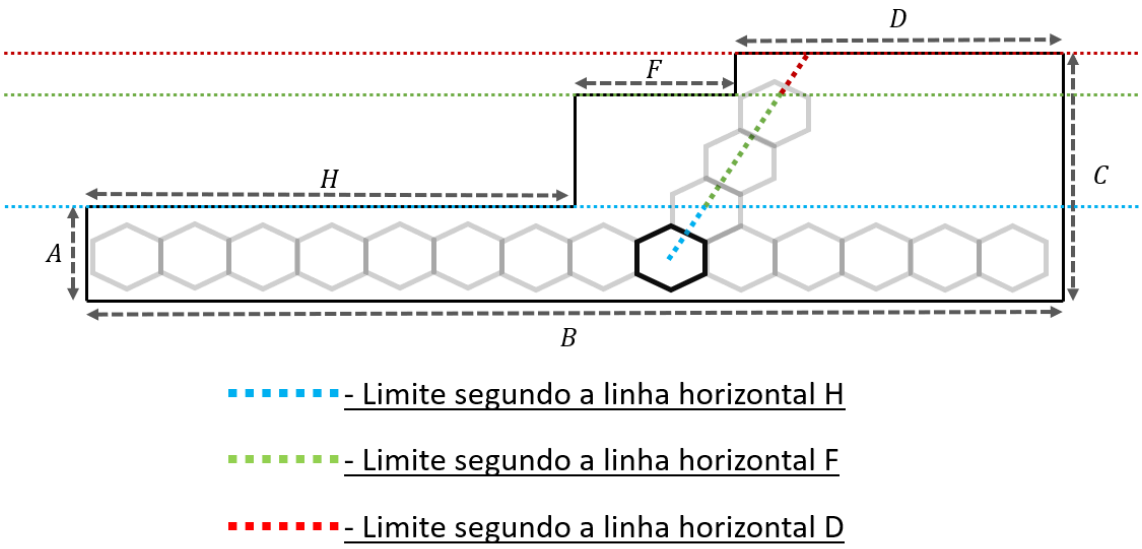


Figura 138 - Limites das linhas diagonais da zona B

Para que a função responsável pela deteção de limites seja capaz de desempenhar a sua função, esta tem de receber duas informações sobre cada um dos limites (Figura 139):

1. $vertical\_limit$ – "Distância entre a linha horizontal que serve de limite, e o eixo dos x do referencial a ser usado";
2. $horizontal\_limit$ – "Distância entre o ponto da extremidade direita, da linha horizontal, e o eixo dos y do referencial a ser usado";

Com estas duas variáveis, e as coordenadas do hexágono de base, a função faz uma análise trigonométrica, no qual, para cada um dos limites disponíveis, vai calcular qual é o espaço ocupado pela linha diagonal, se esta fosse desenhada até o respetivo limite, e depois confirma se os hexágonos da linha diagonal podem ser inseridos dentro do espaço existente. Caso não seja possível desenhar a linha para o respetivo limite, a função repete o cálculo para o limite seguinte, e assim sucessivamente até encontrar uma linha limite para o qual consiga desenhar.

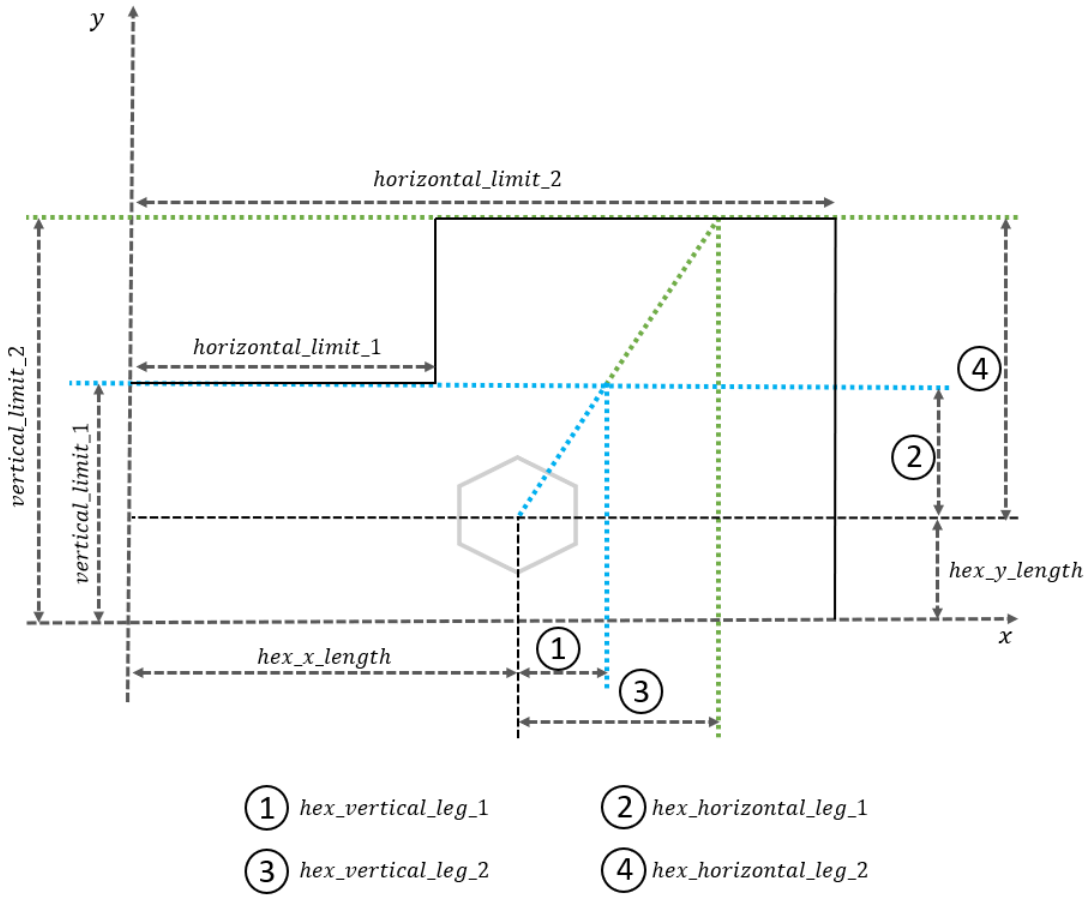


Figura 139 – Ilustração das variáveis de cálculo para a deteção de limites

O processo de cálculo do limite segue então a seguinte formula:

1. Para o primeiro limite, a função calcula qual é a distância vertical entre o hexágono de base e a linha limite (equação (49));

$$hex\_vertical\_leg\_1 = vertical\_limit\_1 - hex\_y\_length \quad (49)$$

onde,

- $hex\_vertical\_leg\_1$ – "Distância vertical entre o hexágono de base e a primeira linha limite" (mm)
- $vertical\_limit\_1$– "Distância entre a linha horizontal limite 1, e o eixo dos x do referencial a ser usado" (mm)

2. Calcula-se a largura horizontal ocupada pela linha diagonal, através de relações trigonométricas (equação (50));

$$hex\_horizontal\_leg\_1 = \frac{hex\_vertical\_leg\_1}{\tan(60^o)} \quad (50)$$

onde,

- $hex\_horizontal\_leg\_1$ – "Distância entre o ponto da extremidade direita, da linha horizontal, e o hexágono de base" (mm)

3. Verifica-se se existe espaço suficiente para desenhar a linha diagonal, dentro do limite a ser analisado, através da condição lógica da equação (51) ;

$$hex\_horizontal\_leg\_1 \leq horizontal\_limit\_1 - hex\_x\_length \quad (51)$$

4. Se a condição anterior se verificar, procede-se com o cálculo do número de hexágonos da linha diagonal, segundo a equação (52) , equação tal, que é obtida da mesma forma que a equação (29) do Algoritmo da zona A. Se não se verificar, repetem-se os cálculos anteriores, mas agora para o limite seguinte, até se encontrar um dos limites em que seja possível desenhar. Na área da figura deste caso de estudo, existem 4 possíveis limites para este cálculo (Figura 140);

$$hex\_diagonal\_number = \frac{hex\_vertical\_leg\_1 - hex\_thickness\_height - \frac{hex\_length}{2}}{hex\_size * \cos(30^o)} \quad (52)$$

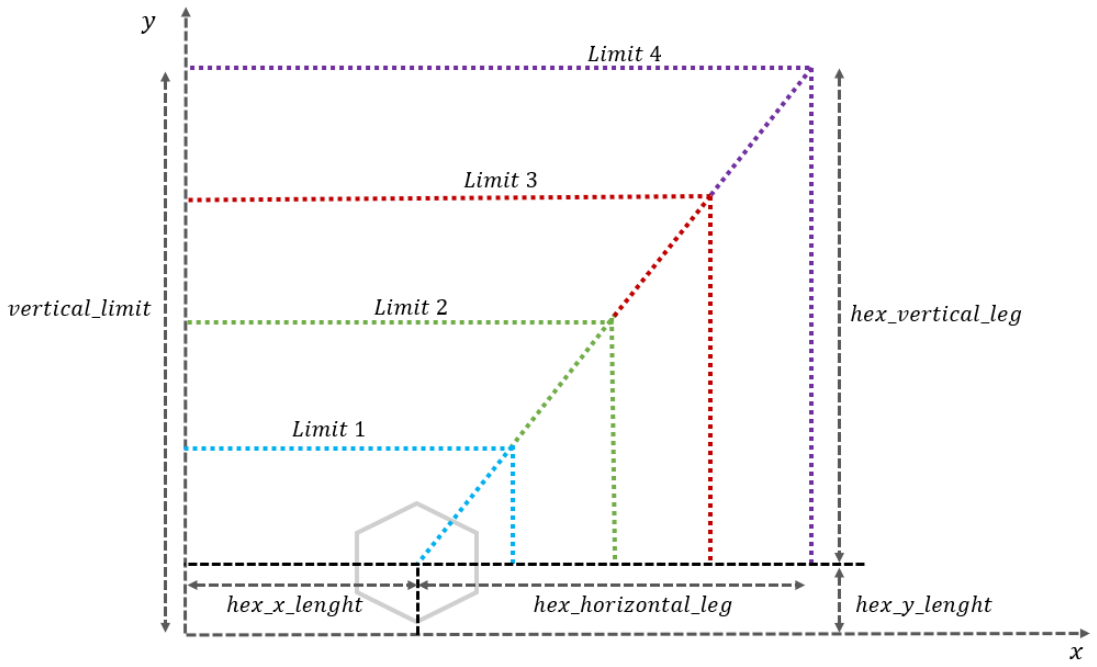


Figura 140 – Ilustração dos limites de desenho da zona dos frisos

5. Caso nenhum dos limites horizontais seja o limite correspondente à linha, então a linha vertical C é tomada como limite, e o cálculo do número de hexágonos que devem de ser desenhados é regido pela equação (53).

$$hex\_diagonal\_number = \frac{hex\_vertical\_leg\_4}{hex\_size * \cos(30^o)} \quad (53)$$

onde,

$$hex\_vertical\_leg\_4 = \frac{horizontal\_limit\_3 - hex\_x\_length}{\tan(60^o)} \quad (54)$$

- $hex\_vertical\_leg\_4$ – "Distância vertical entre o hexágono de base e a quarta linha limite" (mm)
- $horizontal\_limit\_3$ – "Distância entre a linha horizontal limite 3, e o eixo dos x do referencial a ser usado" (mm)

Para que no desenho da linha diagonal, o hexágono não passe por cima do extremo da linha limite, como está representado na Figura 141, as linhas limites não podem ter as mesmas larguras que as linhas da zona dos frisos que as representam. Para cada uma delas tem de ser introduzida uma folga para permitir que isto não aconteça.

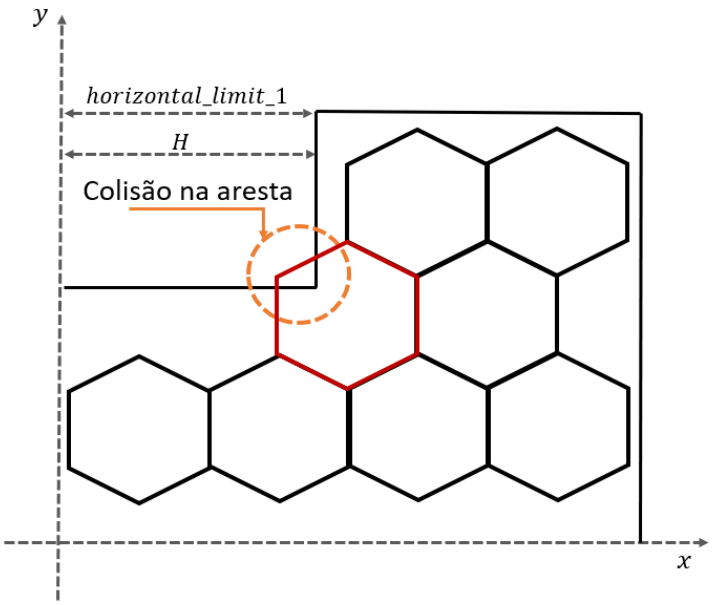


Figura 141 – Colisão nos ângulos retos

Para cada um dos limites tem de se aplicar então a respetiva folga, e isto é conseguido através da equação (55), equação (56) e equação (57). Estas equações são obtidas através das análises trigonométricas, que estão presentes na Figura 142 e Figura 143.

$$horizontal\_limit\_1 = H + \frac{hex\_length + hex\_thickness}{2} \tag{55}$$

$$horizontal\_limit\_2 = horizontal\_limit\_1 + F \tag{56}$$

$$horizontal\_limit\_3 = B - \frac{hex\_size + hex\_thickness}{2} \tag{57}$$

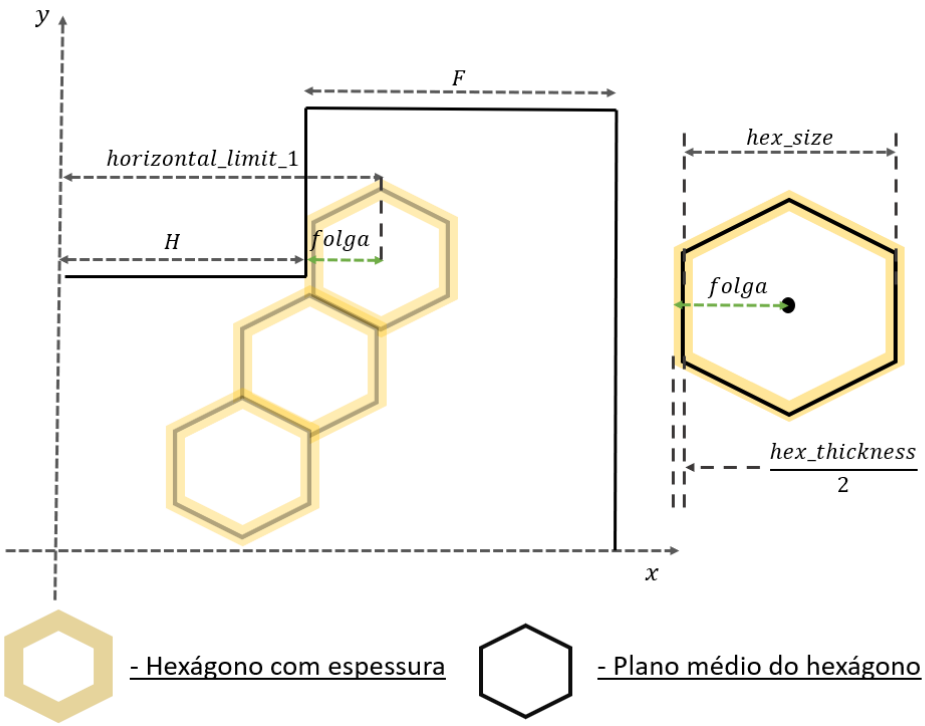


Figura 142 – Demostração gráfica da folga do limite horizontal 1 e 2

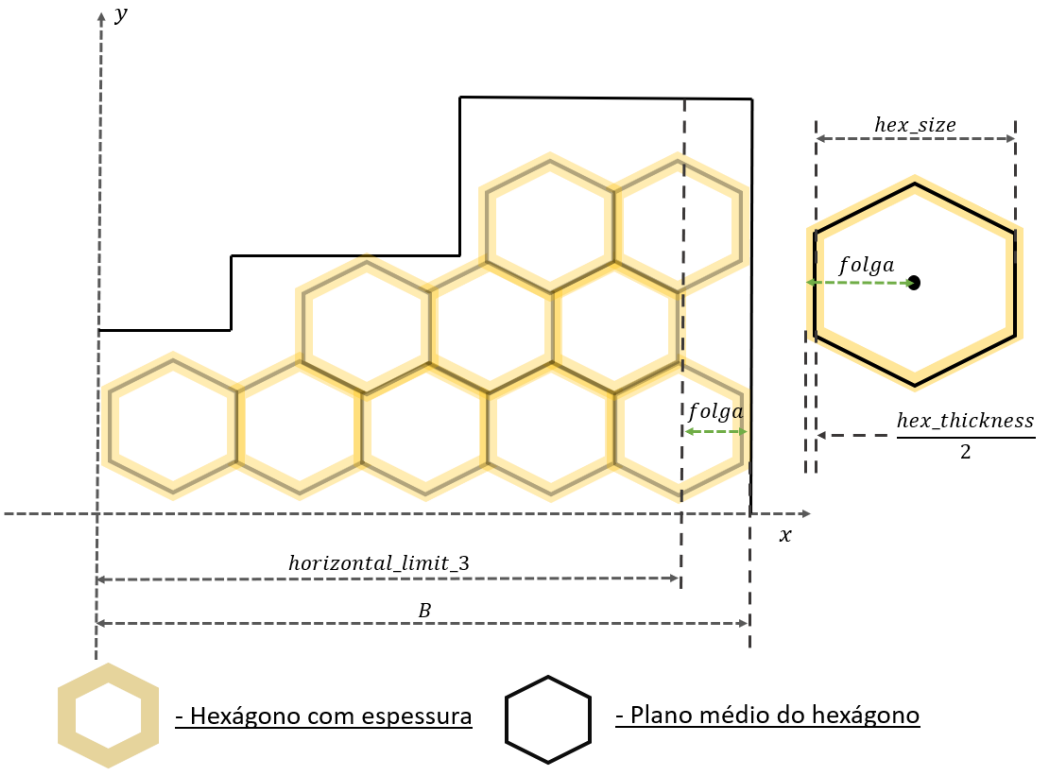


Figura 143 – Demostração gráfica da folga do limite horizontal 1 e 2

Tal como no algoritmo da zona A, nem todos os hexágonos das linhas diagonais necessitam de ser desenhados, e por isso pode-se simplificar o modelo através da supressão desses hexágonos. A abordagem da parte do algoritmo da zona B que trata deste processo, é semelhante ao da zona A, no sentido em que o valor do índice de cada hexágono é incrementado de dois em dois valores, em linhas diagonais intercaladas. Contudo estes dois diferem na parte referente aos casos em que a metodologia mencionada não chega para simplificar corretamente o modelo. No caso da zona B as situações que devem ser tidas em conta são as seguintes:

1. Numa linha diagonal em que ocorre a transição de um limite para o outro, a partir da altura do limite anterior o índice do hexágono tem de obrigatoriamente sofrer incrementos de um valor, e não de dois (Figura 144);

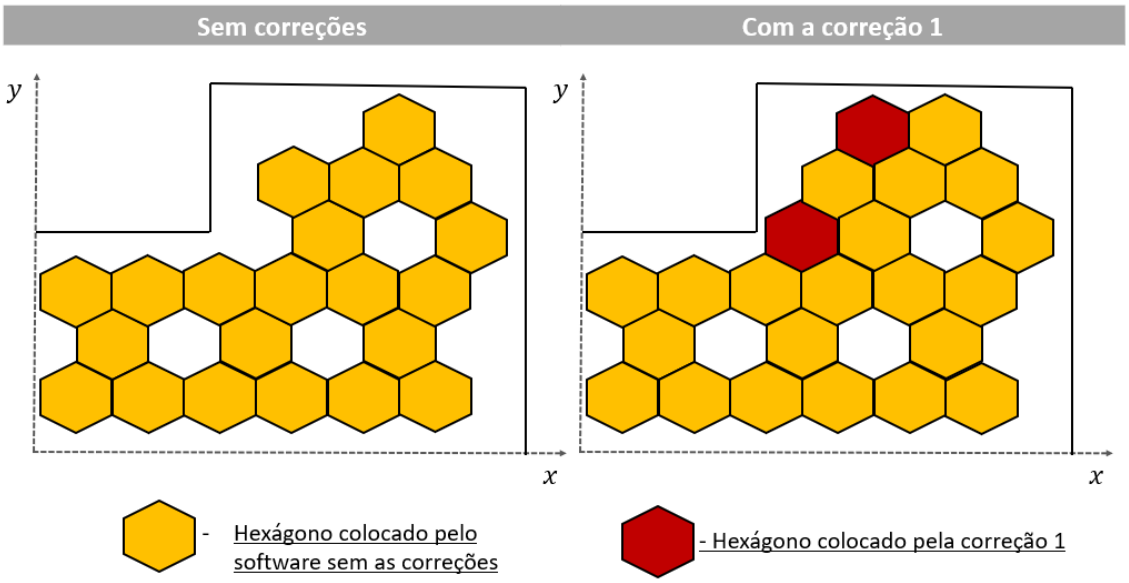


Figura 144 - Exemplo da primeira correção à supressão dos frisos

2. Em situações em que a o incremento de dois valores leva a que o último hexágono da linha diagonal seja suprimido, este tem de obrigatoriamente ser forçado a ser desenhado (Figura 145);

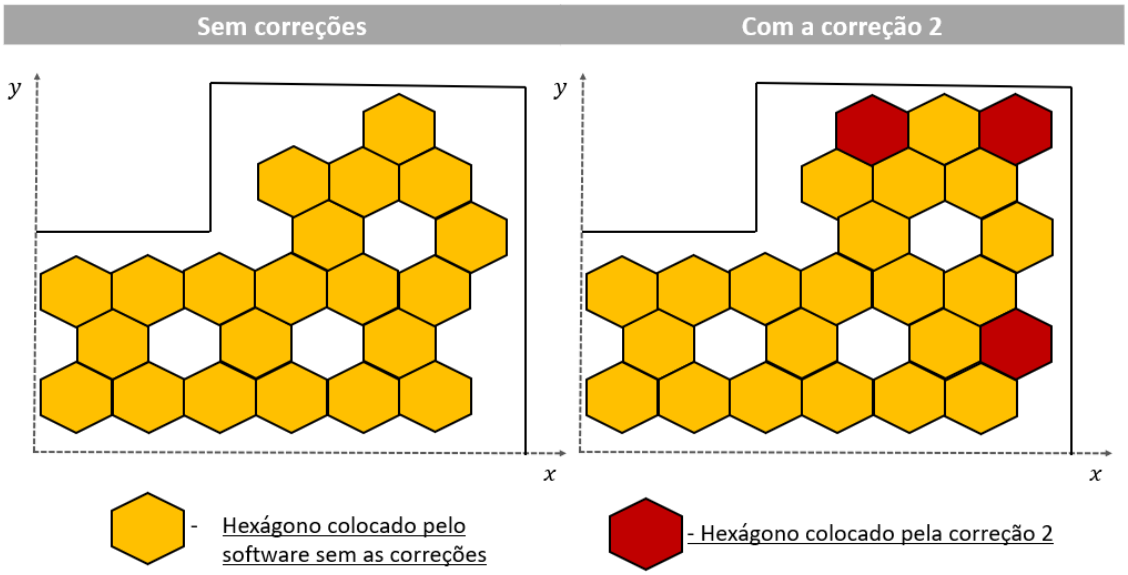


Figura 145 - Exemplo da segunda correção à supressão dos frisos

3. Quando o limite a ser utilizado se tratar da última linha limite do sistema (Linha vertical), o penúltimo hexágono tem de ser forçado a ser desenhado (Figura 146);

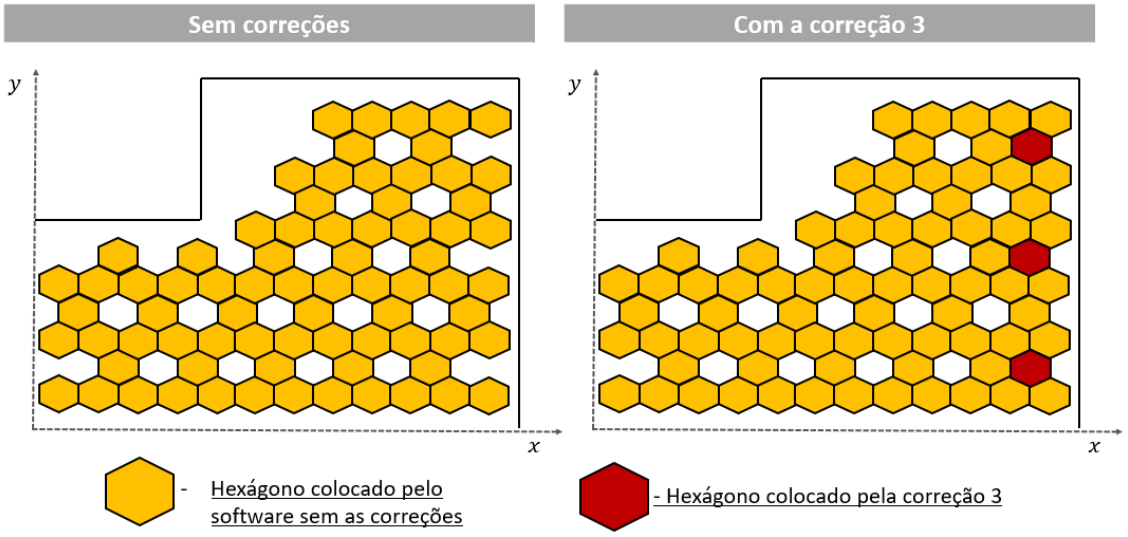


Figura 146 - Exemplo da terceira correção à supressão dos frisos

4. Nos ângulos retos da área de desenho, quando a correção 2 é aplicada, na última linha diagonal antes do ângulo reto, existe um hexágono em excesso (Figura 147). Nestas situações a correção 4 diz ao sistema para incrementar o valor do hexágono em duas unidades, em vez de uma.

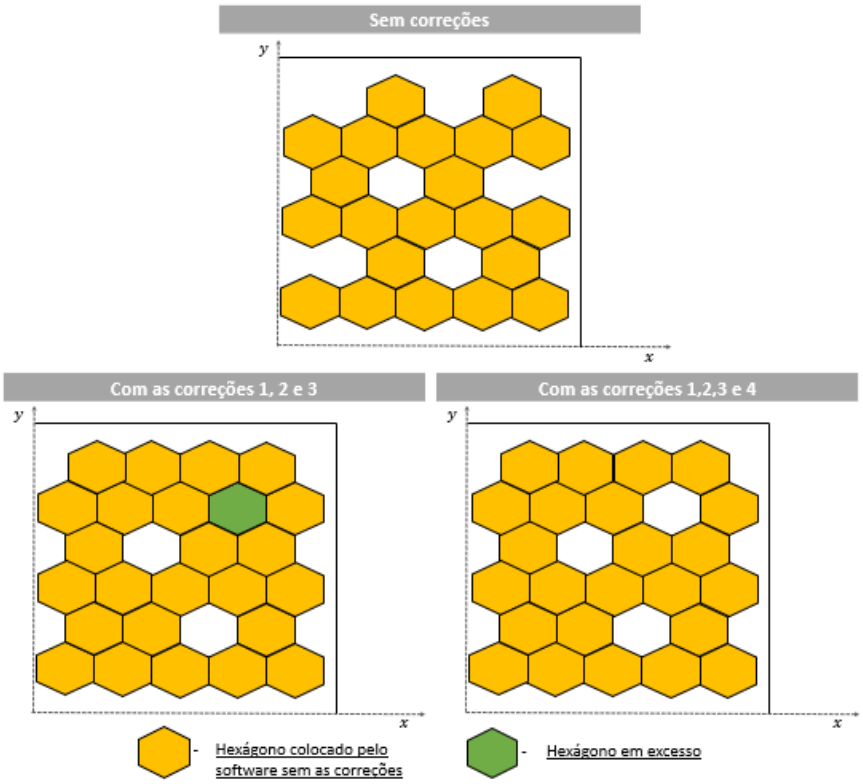


Figura 147 - Exemplo da quarta correção à supressão dos frisos

5. A quinta correção está associada à correção 1. Na correção 1 o programa deteta se uma dada linha diagonal, é a primeira a ser desenhada num dado limite do sistema, de forma a saber se deve aplicar a correspondente correção. Contudo, como o desenho da geometria tem de ser feita em duas etapas, em que as linhas diagonais são feitas de forma intercalada, o que acontece, é que na segunda linha diagonal do respetivo limite, como a linha anterior é feita numa etapa diferente, a correção 1 vai acusar a segunda linha como sendo a primeira linha do respetivo limite (Figura 148). Para dar a volta a este problema, quando a função responsável pela deteção de limites detetar que uma dada linha diagonal é a primeira de um dado limite, esta aplica uma metodologia de recursão [71], no qual a função chama-se a si própria, mas agora com as propriedades da linha anterior. Ao fazer-se isto, para além de analisar o limite que deve ser utilizado na linha atual, a função procura descobrir qual é o limite da linha anterior. Se os dois limites obtidos forem iguais, significa que a linha atual não é a primeira, mas sim a segunda, e, portanto, não se pode aplicar a correção 1.

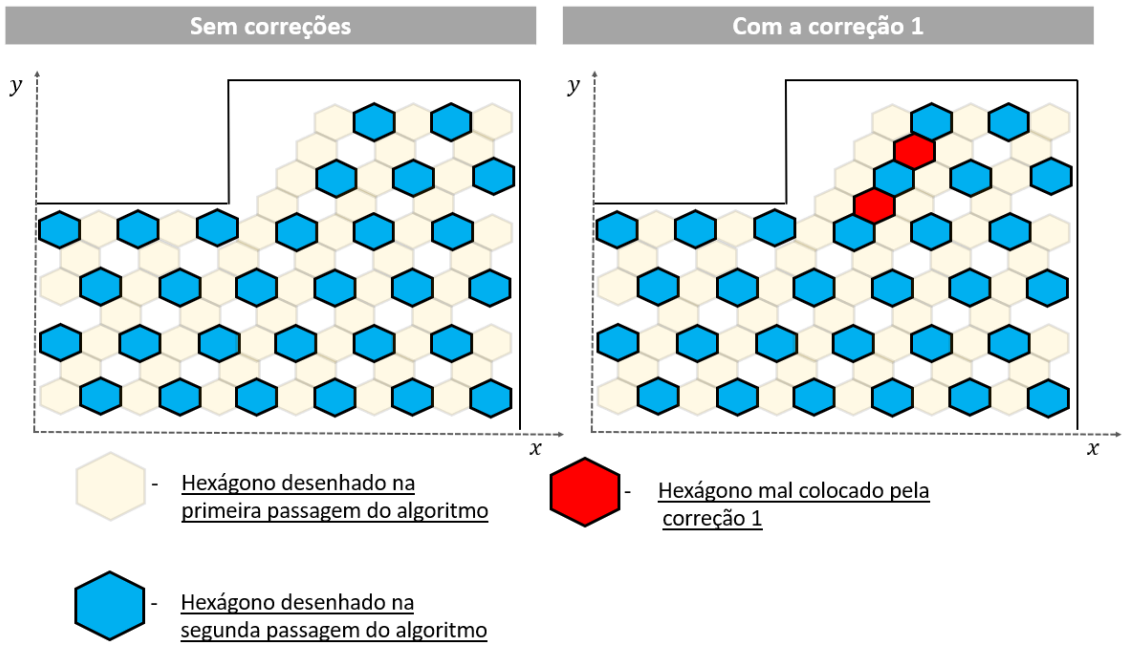


Figura 148 - Exemplo de uma falsa deteção da primeira linha de um limite

3.5.3.6.4 Algoritmo de desenho da Zona B – Código Python

A metodologia teórica tratada no capítulo anterior, foi então convertida para código Python tendo-se obtido as funções das representadas na Figura 149 à Figura 156. Na Figura 149 está demonstrada a parte do código da zona B que serve de base para todas as funções do algoritmo desta zona. Antes de se iniciar o desenho dos hexágonos, são determinados os valores dos limites da área de desenho que devem ser utilizados na deteção de limites (Linhas 1-16), é chamada a função que garante que a zona B seja congruente com a zona A (Linha 19) e são calculadas quantas filas diagonais devem ser desenhadas (Linha 25). Com esta informação disponível o programa inicia o desenho dos hexágonos, começando por calcular para cada hexágono que se insere ao longo da linha B as suas coordenadas horizontais e verticais (Linhas 22 e 30) e detetar para que limite é que a respetiva linha diagonal deve de ser desenhada (Linha 33-42). Com estes dois processos concluídos, o programa envia toda a informação para a *draw_diagonal_hex_B*, que se encarrega de desenhar todos os hexágonos, respeitando os critérios para a supressão dos desnecessários.

```
# Calculate the boundary limits
horizontal_limit_1 = H + (hex_lenght + hex_thickness)/2
horizontal_limit_2 = H + F + (hex_lenght + hex_thickness)/2
horizontal_limit_3 = B - hex_size/2 - hex_thickness/2
vertical_limit_1 = A
vertical_limit_2 = A + G
vertical_limit_3 = C

# Limit Switches, that check in what area the hexagon is being drawned
lim_2 = False
lim_2_counter = 0
lim_3 = False
lim_3_counter = 0
lim_4 = False
hex_diagonal_number_1 = 0
hex_diagonal_number_2 = 0

# Impose continuity between zone A and B
first_int_B = continuity_checker(first_int, hex_vertical_number_A)

# Calculate the hexagon vertical coordinate
hex_y_lenght = A/2-(hex_vertical_number_A-1)*((hex_lenght + hex_height)/2)

# Calculate the number of hexagons along line B
horizontal_hex_number_B = int(math.floor((B-hex_thickness)/hex_size))

for index_B in range (first_int_B, horizontal_hex_number_B, 2):

    # Calculate the hexagon horizontal coordinate
    hex_x_lenght = hex_size/2 + hex_thickness/2 + index_B*hex_size

    # This checks what boundarys of the hexagon area it needs to use
    lim_2, lim_3, lim_4, hex_diagonal_number,
    lim_2_counter, lim_3_counter,
    hex_diagonal_number_1, hex_diagonal_number_2 = limit_checker(
        lim_2, lim_3, lim_4, hex_y_lenght, hex_x_lenght,
        horizontal_limit_1, horizontal_limit_2, horizontal_limit_3,
        vertical_limit_1, vertical_limit_2, vertical_limit_3,
        lim_2_counter, lim_3_counter,
        hex_diagonal_number_1, hex_diagonal_number_2,
        hex_size, hex_thickness,
        hex_height, hex_thickness_height)

    # Draw the base hexagon and it's diagonal line
    draw_diagonal_hex_B(hex_diagonal_number, s, sketch_name,
        horizontal_limit_3, hex_size, lim_2, lim_3, lim_4,
        lim_2_counter, lim_3_counter, hex_x_lenght, hex_y_lenght,
        first_int, hex_diagonal_number_1, hex_diagonal_number_2, g, v)
```

Figura 149 – Algoritmo de desenho Zona B

Dentro da função principal mencionou-se a congruência entre a zona A e a zona B. Esta parte do programa é regida pela função *continuity_checker* (Figura 150), que dependendo da passagem que se está a desenhar, e o número de hexágonos desenhado verticalmente na zona A, altera as propriedades da zona B, de maneira a permitir que exista congruência nas passagens da zona A e da zona B. Os critérios condicionais utilizados dentro da função são os deduzidos no Capítulo 3.5.3.6.3.

```
1 # Check if there is continuity between Zone A and B
2 # If there is not, then it alters the parameters
3 # In order for the continuity to happen
4 def continuity_checker(first_int, hex_vertical_number_A):
5
6   if ((hex_vertical_number_A%2 == 0) and
7         (first_int == 1) and
8         (hex_vertical_number_A%4 == 0)) or \
9         ((hex_vertical_number_A%2 == 0) and
10        (first_int == 2) and
11        (hex_vertical_number_A%4 != 0)) or \
12        ((hex_vertical_number_A%2 != 0) and
13        (first_int == 1) and
14        ((hex_vertical_number_A-1)%4 != 0) or \
15        ((hex_vertical_number_A%2 != 0) and
16        (first_int == 2) and
17        ((hex_vertical_number_A-1)%4 == 0))):
18
19    first_int = 1
20  else:
21    first_int = 2
22
23  return first_int
```

Figura 150 – Função de congruência (*continuity_checker*)

A deteção do limite da linha diagonal é feita pela função *limit_checker* (Figura 151, Figura 152 e Figura 153). Tal como foi abordado na parte teórica, esta parte do algoritmo deteta qual é o limite que deve ser utilizado no desenho da linha diagonal através de cálculos trigonométricos. Para além disto, a função utiliza recursividade [71], para detetar se ocorreu uma transição de limite, na linha diagonal. Este processo consiste em chamar a função *limit_checker* dentro de si mesma (Linha 61 e 96), mas com os parâmetros da linha imediatamente anterior, para conferir se as duas linhas têm o mesmo limite. Se tiverem, então não houve uma troca de limite, e não é necessário recorrer a correções na supressão de hexágonos. Se forem diferentes, esta informação tem de ser passada para o algoritmo da zona B para fazer as devidas correções.

```python
# This function calculates what horizontal limit the program should use,
# while drawing the diagonal hexagons.
# It basicaly works as a boundary detection function,
# and does not allow the hexagon drawing to leave the hexagon area
def limit_checker(lim_2, lim_3, lim_4, hex_y_lenght, hex_x_lenght,
    horizontal_limit_1, horizontal_limit_2, horizontal_limit_3,
    vertical_limit_1, vertical_limit_2, vertical_limit_3,
    lim_2_counter, lim_3_counter,
    hex_diagonal_number_1, hex_diagonal_number_2,hex_size,
    hex_thickness, hex_height, hex_thickness_height):

    # Measurements of the first limit triangle
    hex_vertical_leg_1 = vertical_limit_1 - hex_y_lenght
    hex_horizontal_leg_1 = hex_vertical_leg_1/np.tan(np.radians(60))

    # Check if it should use line H as it's upper limit
    if hex_horizontal_leg_1 <= (horizontal_limit_1-hex_x_lenght):

        # Calculate the number of diagonal hexagons
        hex_diagonal_number = int(math.floor((
            (hex_vertical_leg_1)/\
            np.cos(np.radians(30))-\
            (hex_height/2+hex_thickness_height)/\
            np.cos(np.radians(30)))/hex_size))

        # Argument used for the correction 1 to know when to activate
        hex_diagonal_number_1 = hex_diagonal_number

        # Turns of limit 2, in case if the function enters recursion
        lim_2 = False

    else:
        # Measurements of the second limit triangle
        hex_vertical_leg_2 = vertical_limit_2 - hex_y_lenght
        hex_horizontal_leg_2 = hex_vertical_leg_2/np.tan(np.radians(60))

        # Check if it should use line F as it's upper limit
        if hex_horizontal_leg_2 <= (horizontal_limit_2-hex_x_lenght):

            # Calculate the number of diagonal hexagons
            hex_diagonal_number = int(math.floor((
                (hex_vertical_leg_2)/\
                np.cos(np.radians(30))-\
                (hex_height/2 + hex_thickness_height)/\
                np.cos(np.radians(30)))/hex_size))

            # Argument used for the correction 1
            hex_diagonal_number_2 = hex_diagonal_number
```

Figura 151 - Função de deteção de limites (*limit_checker*)(1/3)

```
            # Turns of limit 3, in case if the function enters recursion
            lim_2 = True
            lim_3 = False
            lim_2_counter += 1

            # Use recursion in order to check if there was
            # a transition between limits
            if lim_2_counter == 1:
                prev_lim = limit_checker(lim_2, lim_3, lim_4,
                    hex_y_lenght, (hex_x_lenght-hex_size),
                    horizontal_limit_1, horizontal_limit_2,
                    horizontal_limit_3, vertical_limit_1,
                    vertical_limit_2, vertical_limit_3,
                    lim_2_counter, lim_3_counter,
                    hex_diagonal_number_1, hex_diagonal_number_2,
                    hex_size, hex_thickness)[0]
            if prev_lim:
                lim_2_counter +=1
        else:
            # Measurements of the third limit triangle
            hex_vertical_leg_3 = vertical_limit_3 - hex_y_lenght
            hex_horizontal_leg_3 = hex_vertical_leg_3/\
                np.tan(np.radians(60))

            # Check if it should use line D as it's upper limit
            if hex_horizontal_leg_3 <= (horizontal_limit_3-hex_x_lenght):

                # Calculate the number of diagonal hexagons
                hex_diagonal_number = int(math.floor((
                    (hex_vertical_leg_3)/np.cos(np.radians(30))-\
                    (hex_height/2 + hex_thickness_height)/\
                    np.cos(np.radians(30)))/hex_size))

                # Turns of limit 2, in case if the function enters
                # it's recursive state
                lim_2 = False
                lim_3 = True
                lim_3_counter += 1

                # Use recursion in order to check if there was a
                # transition between limits
                if lim_3_counter == 1:
                    prev_lim = limit_checker(lim_2, lim_3, lim_4,
                    hex_y_lenght, (hex_x_lenght-hex_size),
                        horizontal_limit_1, horizontal_limit_2,
                        horizontal_limit_3, vertical_limit_1,
                        vertical_limit_2, vertical_limit_3,
                        lim_2_counter, lim_3_counter,
                        hex_diagonal_number_1, hex_diagonal_number_2,
                        hex_size, hex_thickness)[1]
```

Figura 152 - Função de deteção de limites (*limit_checker*)(2/3)

```
106                 if prev_lim:
107                     lim_3_counter +=1
108 
109             # Check if it should use line C as it's upper limit
110             else:
111                 # Measurements of the fourth limit triangle
112                 hex_horizontal_leg_4 = horizontal_limit_3 - hex_x_lenght
113                 hex_vertical_leg_4 = hex_horizontal_leg_4*\
114                     np.tan(np.radians(60))
115 
116                 # Calculate the number of diagonal hexagons
117                 hex_diagonal_number = int(math.floor((
118                     (hex_vertical_leg_4)/\
119                     np.cos(np.radians(30))-\
120                     (hex_height/2 + hex_thickness_height)/\
121                     np.cos(np.radians(30)))/hex_size))
122 
123                 # Turns of limit 3, in case if the function enters
124                 # it's recursive state
125                 lim_3 = False
126                 lim_4 = True
127 
128     return (lim_2, lim_3, lim_4, hex_diagonal_number,
129         lim_2_counter, lim_3_counter,
130         hex_diagonal_number_1, hex_diagonal_number_2)
```

Figura 153 - Função de deteção de limites (*limit_checker*)(3/3)

O desenho do hexágono base e a sua linha diagonal é desempenhado pela função *draw_diagonal_hex_B* (Figura 154). Esta função funciona da mesma forma que a equivalente da Zona A (*draw_diagonal_hex_A*), onde são calculadas as coordenadas de cada hexágono, e os seus parâmetros são enviados para a função draw_hex, para serem desenhados. Só que aqui, são chamadas duas funções extras, a *last_diagonal_check* (Figura 155) e a *forced_last_line* (Figura 156). Estas duas funções servem, respetivamente, para determinar se a linha que se está a desenhar é a última do respetivo limite, e se o último hexágono da linha diagonal vai ser forçado a ser desenhado. Estes dois parâmetros são necessários para conferir se é necessário proceder com a correção 5 do Capítulo 3.5.3.6.4. Para além disto, da Linha 30–42 encontram-se as condições lógicas que referem se um determinado hexágono deve ou não ser suprimido, através de informação obtida dentro desta função, do *limit_checker*, da *last_diagonal_check* e da *forced_last_line*.

```
# This function draws the diagonal hexagons of zone B
def draw_diagonal_hex_B (hex_diagonal_number, s, sketch_name,
    horizontal_limit_3, hex_size, lim_2, lim_3, lim_4,
    lim_2_counter, lim_3_counter, hex_x_lenght, hex_y_lenght,
    first_int, hex_diagonal_number_1, hex_diagonal_number_2, g, v):

    # Loop through the diagonal line
    index_B = 0
    while index_B <= (hex_diagonal_number):

        # Calculate the hexagon coordenates
        diagonal_hex_x_lenght = hex_x_lenght + \
            (index_B)*hex_size*np.cos(np.radians(60))

        diagonal_hex_y_lenght = hex_y_lenght + \
            (index_B)*hex_size*np.sin(np.radians(60))

        # Draw the hexagon on the calculated coordinates
        draw_hex(diagonal_hex_x_lenght, diagonal_hex_y_lenght,
            s, sketch_name, g, v)

        # Check if the current diagonal line is the last one of the limit
        last_diagonal_check = final_diagonal_check(horizontal_limit_3,
            diagonal_hex_x_lenght, hex_size, index_B, hex_diagonal_number)

        # Check if the last hexagon is forced to activate
        forced_last_line = activate_last_line(lim_3, hex_diagonal_number)

        # Logical conditions that check if a hexagon should be supressed
        if ((first_int == 2) and
            not(lim_2  and lim_2_counter == 1 and
            (index_B+1)>= hex_diagonal_number_1) and
            not(lim_3  and lim_3_counter == 1 and
            (index_B+1)>= hex_diagonal_number_2) and
            not(lim_4  and index_B==(hex_diagonal_number-2)) or \
            (forced_last_line and last_diagonal_check and not(lim_4))):

            index_B +=2
            if index_B == (hex_diagonal_number+1):
                index_B =  hex_diagonal_number
        else:
            index_B +=1
```

Figura 154 - Função de desenho dos hexágonos da zona B (*draw_diagonal_hex_B*)

```
# This function checks if a diagonal line is the last one of the limit
def final_diagonal_check(horizontal_limit,
    diagonal_hex_x_lenght, hex_size, index_B, hex_diagonal_number):

    if (horizontal_limit-diagonal_hex_x_lenght) < (2.2*hex_size) and \
        (index_B == (hex_diagonal_number-2)):

        # The value True means that it is the last line
        last_diagonal_check = True
    else:
        # The value False means that it isn't the last line
        last_diagonal_check = False

    return last_diagonal_check
```

Figura 155 - Função de deteção da última linha do limite

```
# This function checks if the last hexagon
# of the diagonal line is forced to activate
def activate_last_line(lim, hex_diagonal_number):

    if (lim and (hex_diagonal_number%2!=0)):
        # The value True means that the hexagon
        # was forced to activate
        forced_last_line = True
    else:
        # The value False means that the hexagon
        # was not forced to activate
        forced_last_line = False

    return forced_last_line
```

Figura 156 – Função de deteção da ativação do último hexágono da linha diagonal

Todo o processo de desenho do algoritmo da zona B encontra-se esquematizado nos fluxogramas da Figura 157 à Figura 161.

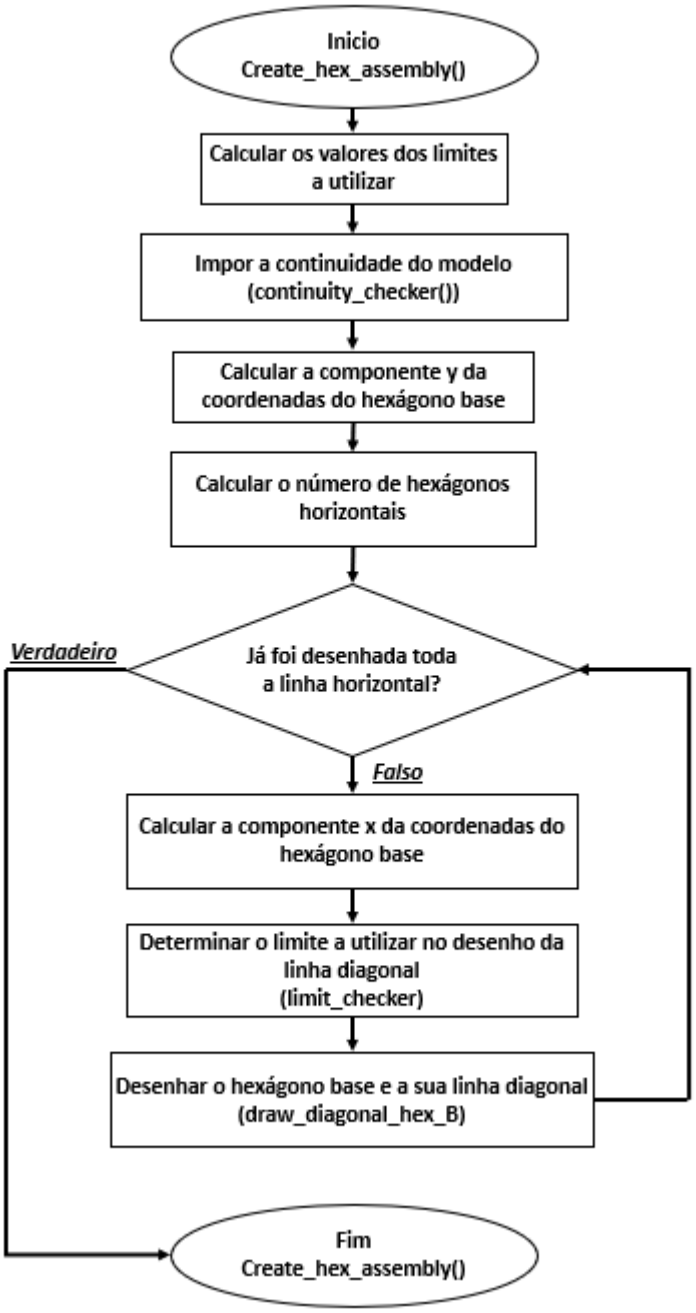


Figura 157 – Algoritmo de desenho da zona B (Fluxograma)

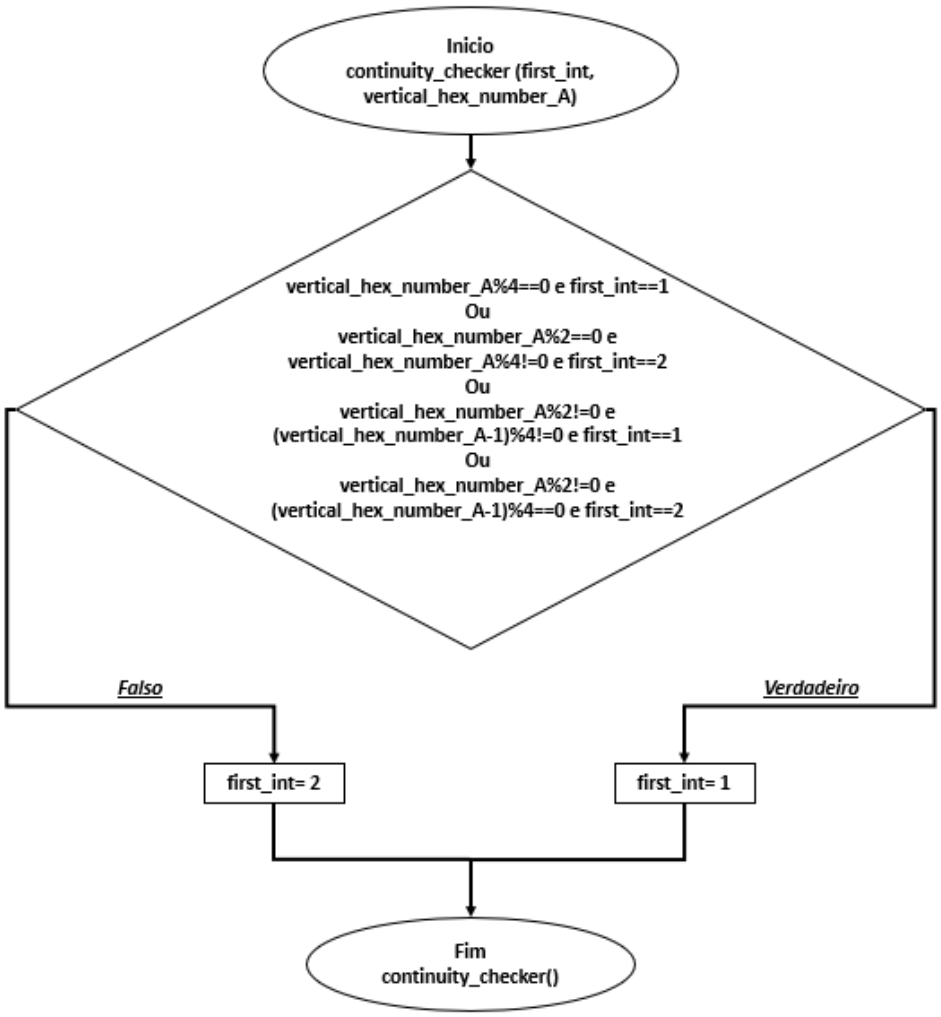


Figura 158 – Imposição da continuidade da geometria (Fluxograma)

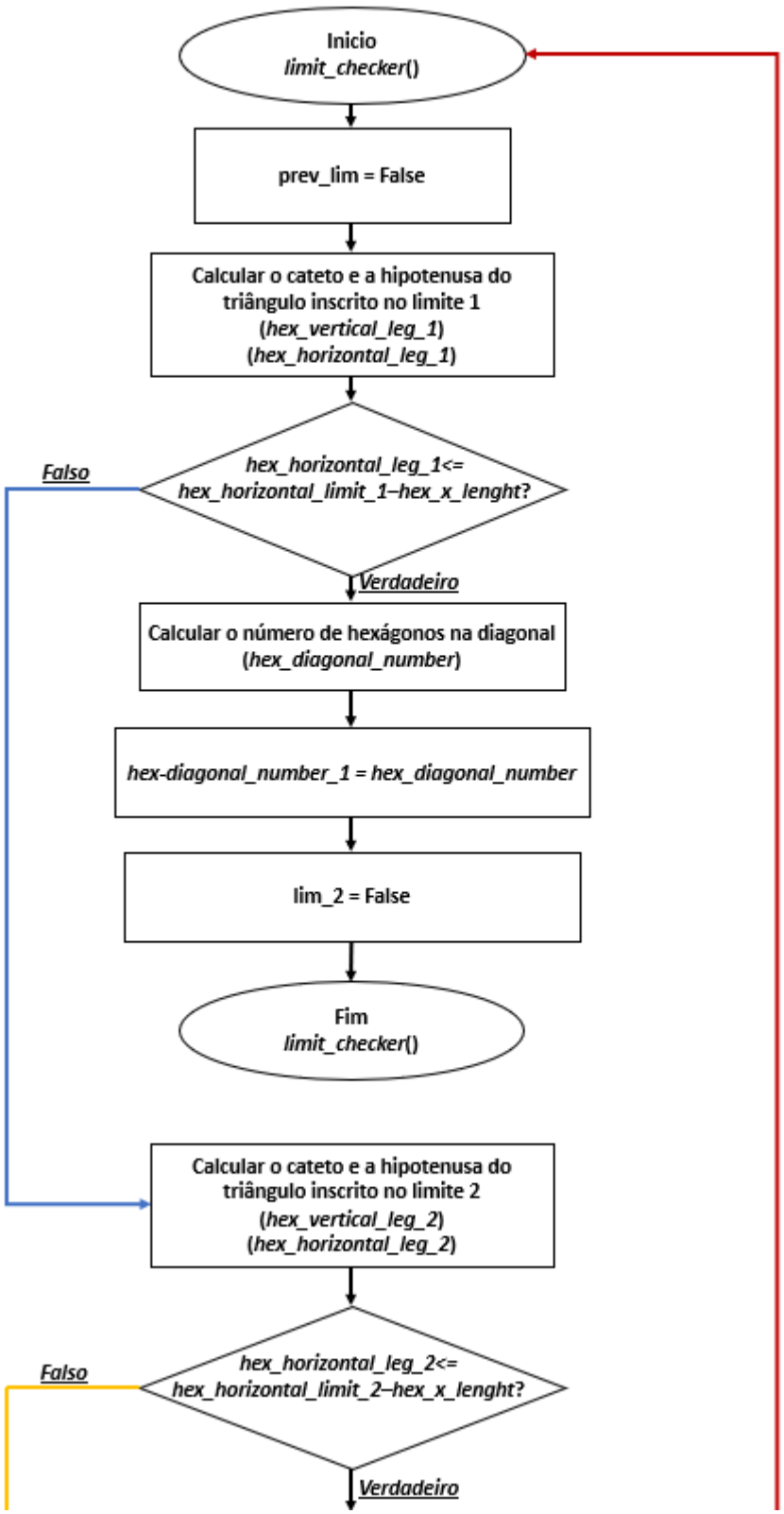


Figura 159 – Procura dos limites da linha diagonal (Fluxograma) (1/3)

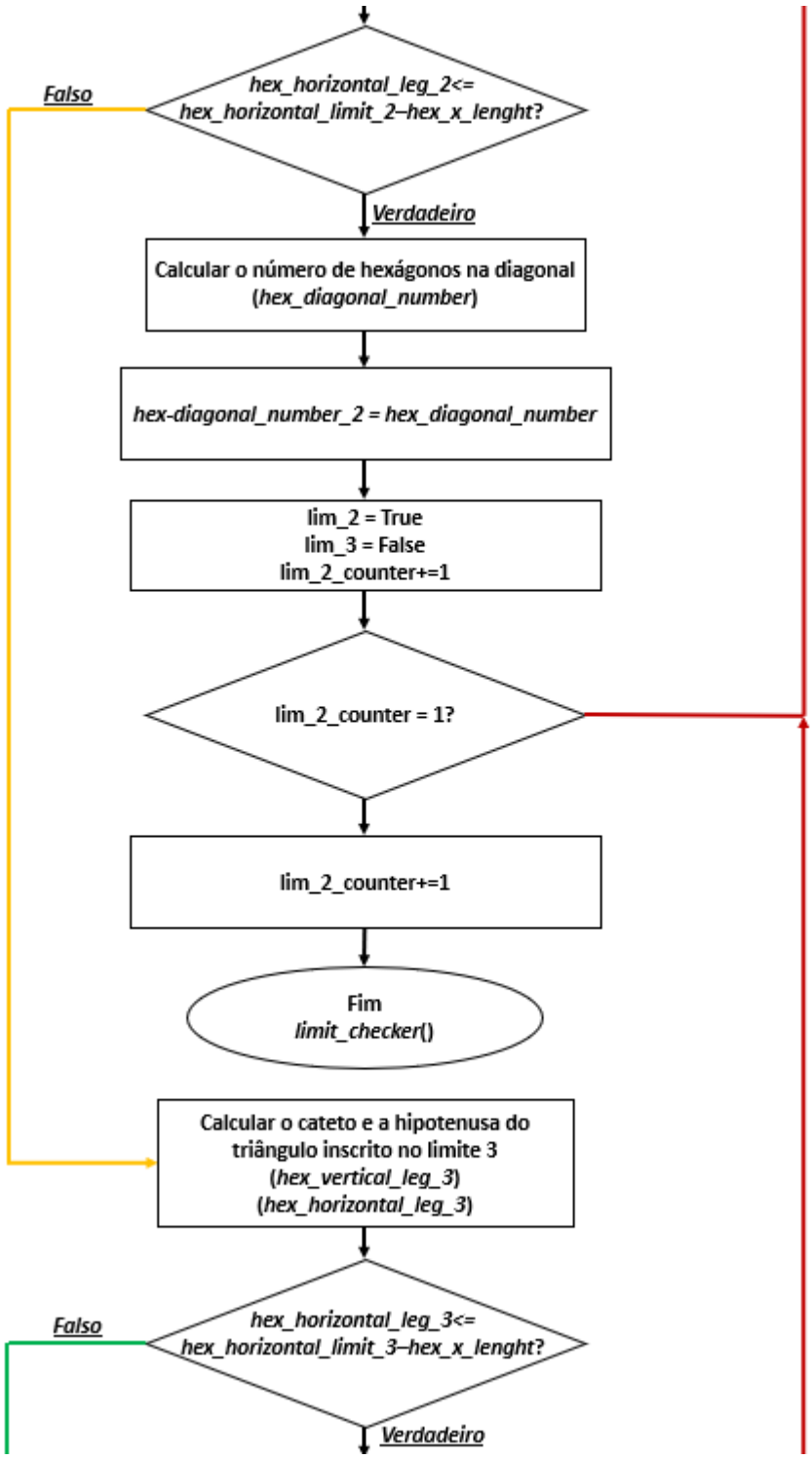


Figura 160 – Procura dos limites da linha diagonal (Fluxograma) (2/3)

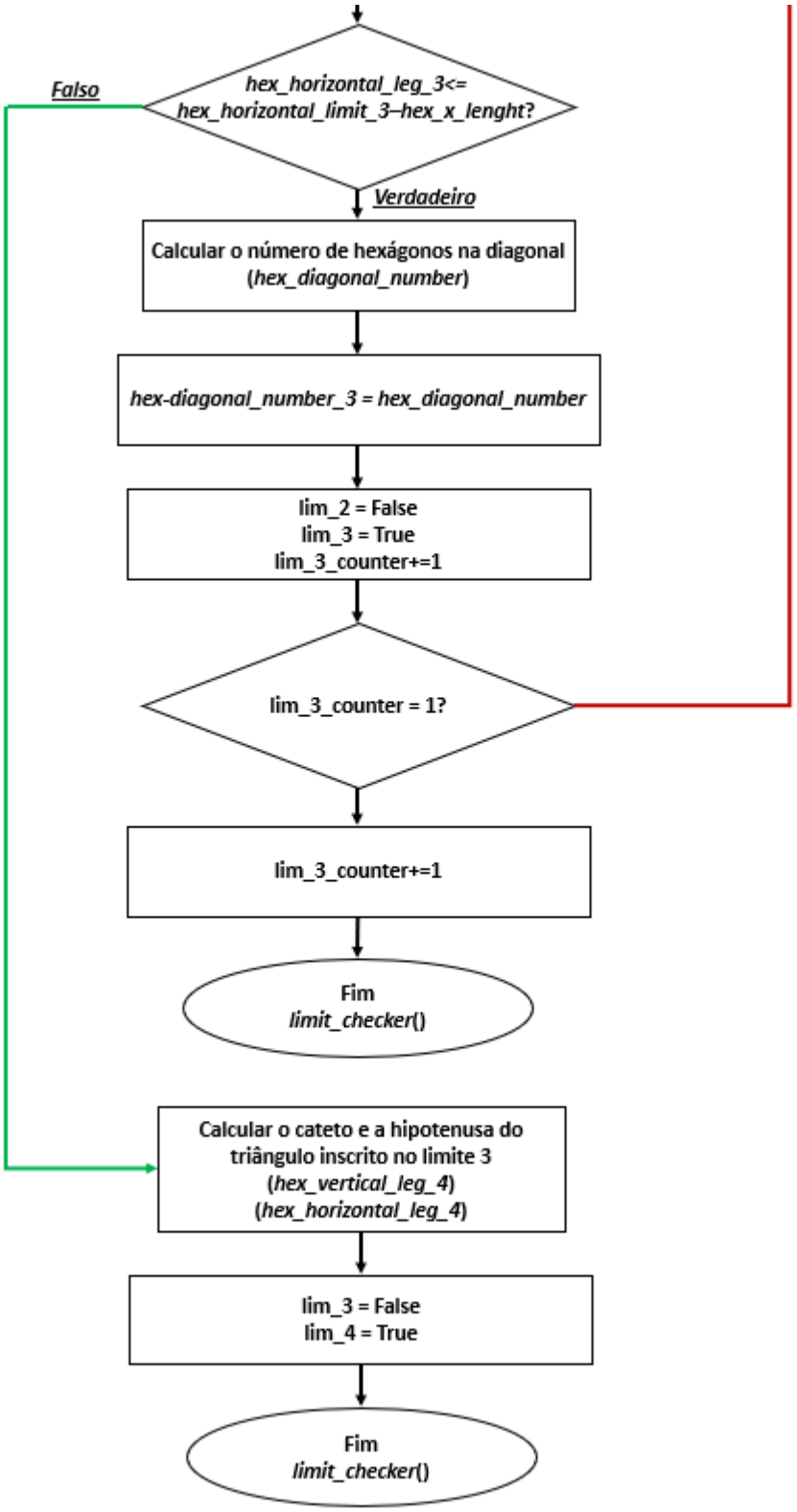


Figura 161 – Procura dos limites da linha diagonal (Fluxograma) (3/3)

3.5.3.6.5 Algoritmo de desenho – Etapas de desenho

Idealmente, como o desenho dos frisos tem de ser feito em duas etapas, o processo de extrusão deveria de ocorrer, após realizado o desenho de cada uma das fases da geometria, de maneira a que o processo seja o mais simples possível. Isto não é possível, devido à maneira como as keys dos repositórios do Abaqus® funcionam internamente. Normalmente quando se adiciona um item a um dado repositório, como este foi o último a ser colocado, a chave que o representa, é a de maior valor absoluto do respetivo repositório (Figura 162). Isto é verdade durante o desenho de um sketch, contudo quando se extrude o desenho, o repositório não varia da mesma forma. Quando um desenho é extrudido a numeração das keys varia de tal maneira, que deixa de ser possível prever a chave de uma dada geometria através da metodologia que se utilizou até agora (*g[max(keys)]*). Durante a resolução deste trabalho, não se conseguiu descobrir como é que se dá esta alteração dos repositórios, nem se encontrou na documentação do API do Abaqus®, ou do próprio software, algo sobre este assunto [69] [72]. Por causa disto, desenvolveu-se uma metodologia que permitisse dar a volta ao problema.

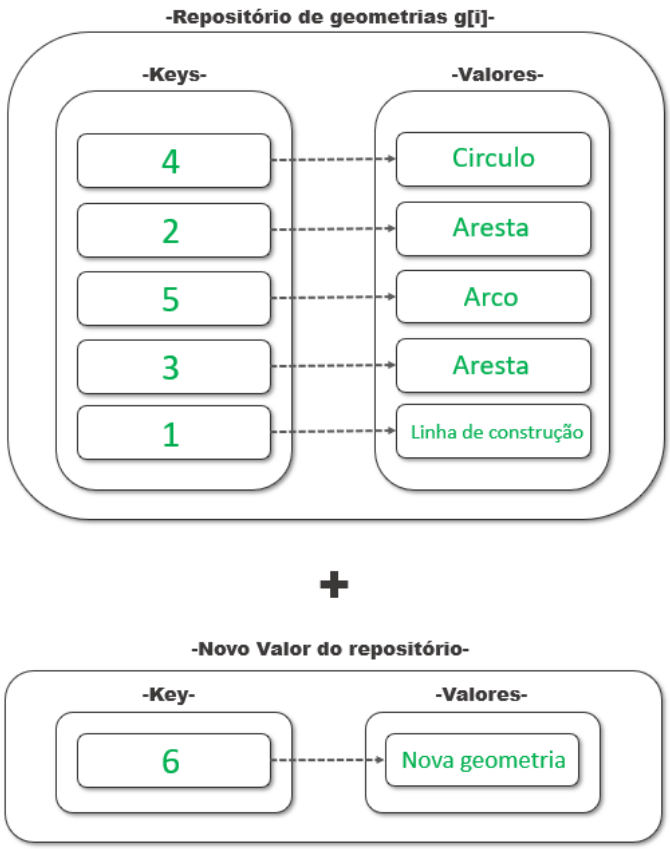


Figura 162 - Ilustração da adição de valores a um repositório

Durante o desenho da geometria dos frisos, em vez de se extrudir o desenho, no final de cada etapa, quando o primeiro desenho for terminado, este é guardado em memória, e subsequentemente tem toda a sua geometria apagada. É importante que seja a geometria a ser apagada e não a instância do *sketch*, pois, como foi mencionado no Capítulo 3.5.3.4, dentro de uma instância de *sketch*, se um dos valores dos repositórios for apagado, as keys revertem todas para o valor imediatamente anterior, e, portanto, se forem todas apagadas, estas voltam para o valor inicial do desenho, e pode-se continuar a utilizar a função *g[max(g.keys)]* para designar cada um dos elementos. Quando se apagar a geometria do primeiro desenho, começa-se a desenhar de imediato a segunda etapa, mas na mesma instância de *sketch*, visto que assim, cada elemento que é adicionado continua a ser o de maior valor absoluto do repositório (Figura 163). Terminando-se o desenho da segunda etapa, esta pode ser imediatamente extrudida, pois não é necessário desenhar mais nenhuma geometria. Com esta extrusão feita, abre-se uma nova instância de um *sketch*, e vai se buscar o desenho da primeira etapa que está guardado em memória. Como o *sketch* foi guardado previamente, na posição correta da peça, não é necessário movê-lo, e, portanto, deixa de ser necessário determinar as *keys* dos elementos. Assim, pode-se extrudir de imediato a geometria da primeira etapa. Este processo está esquematicamente representado no fluxograma da Figura 164.

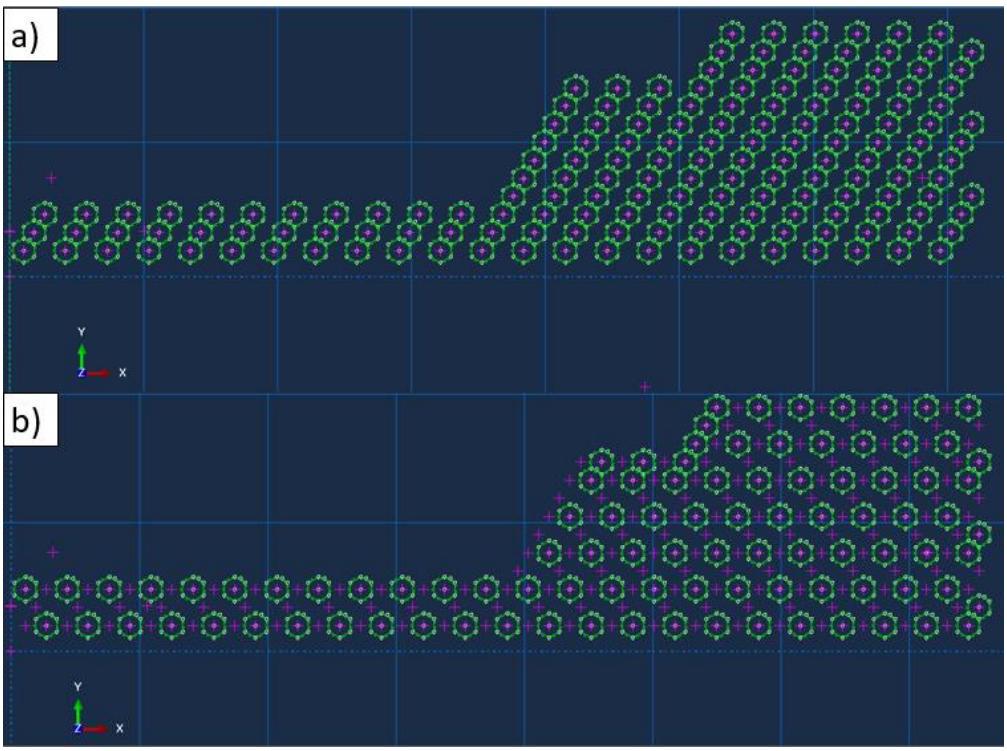


Figura 163 - Etapas de desenho no Abaqus® a) Etapa 1 b) Etapa 2

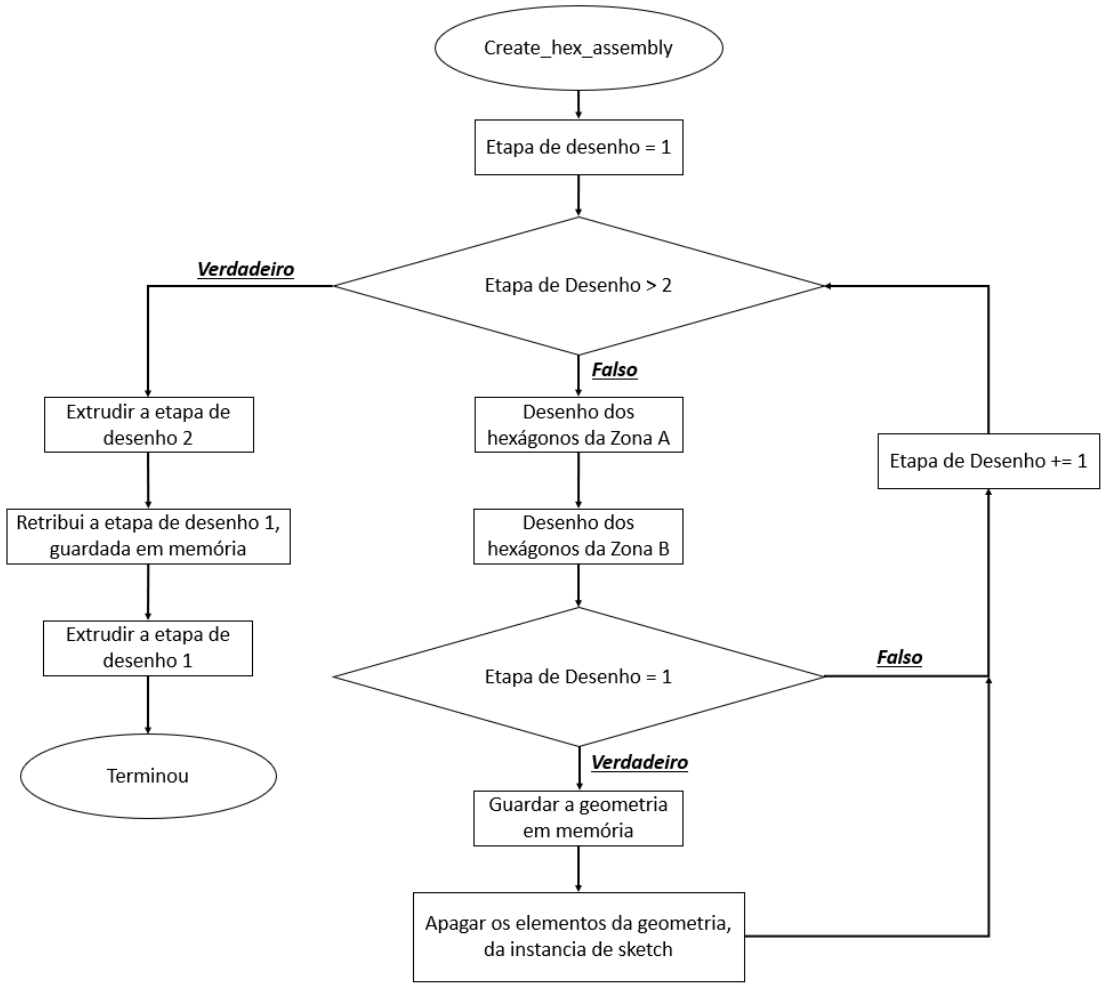


Figura 164 - Fluxograma da estratégia de extrusão por etapas

Inserindo todo este processo no algoritmo de desenho, escreveu-se o código da Figura 165. Nesta porção do algoritmo de desenho, caso a etapa a desenhar seja a primeira, o programa guarda a geometria em memória através das linhas 6 e 7, e só depois, cria uma lista com as *keys* que fazem parte da geometria, da etapa 1, (Linhas 10-14) e apaga todas as geometrias existentes nessa lista, em simultâneo (Linha 17). Na segunda etapa de desenho o programa não entra nesta secção porque o valor da variável que diz ao programa se é a primeira vez que ele está a correr esta secção do código passou para falsa (Linha 21) e, portanto, após a sua conclusão sai de dentro do loop de desenho e extrude a geometria de desenho que se encontra de momento ativa (Linha 26). Após isto, o programa inicia um novo desenho, e vai buscar a geometria que foi guardada em memória (Linha 28-32), para a poder extrudir, como se fez para a anterior (Linha 35).

```
1         # If it's the first sketch, it needs to be saved
2         # in memory, to be extruded later
3         if sketch_step_1:
4
5             # Save current sketch in memory
6             mymodel.ConstrainedSketch(name='Sketch_Step_1',
7                 objectToCopy=s)
8
9             # Get all the geometrys from the current sketch
10            base_hedge_keys = g.keys()
11            list_of_hedges_to_delete = []
12
13            for del_index in base_hedge_keys:
14                list_of_hedges_to_delete.append(g[del_index])
15
16            # Delete Current sketch from the sketch window
17            s.delete(objectList = list_of_hedges_to_delete)
18
19        # This boolean makes the program know that the first step
20        # has been completed
21        sketch_step_1 = False
22
23
24    # Extrude Sketch Step 2
25    Extrude_sketch(drawing_plane, align_edge, s, profile_name)
26
27    # Start Sketch Step 1
28    s, g, v = sketch_start(drawing_plane, align_edge,
29        p, profile_name, center_point)
30
31    # Get Sketch_Step_1 saved sketch
32    s.retrieveSketch(sketch=mymodel.sketches['Sketch_Step_1'])
33
34    # Extrude Sketch Step 1
35    Extrude_sketch(drawing_plane, align_edge, s, profile_name)
```

Figura 165 – Algoritmo de extrusão por etapas (Python)

Devido ao facto de a extrusão de um sketch, acontecer diversas vezes dentro do programa, esta parte do código possui a sua própria função (Figura 166), que recebe o plano onde se encontra o desenho, a aresta que o alinha, a instância e o nome do sketch, e extrude até à face de alvo, que o utilizador definiu nos pré-requisitos (Capitulo 3.5.3.5) (Figura 166 Linhas 12-15). Após todo este processo a função elimina o desenho, para não ficar a ocupar espaço na memória (Figura 166 Linha 18).

```
# Extrude a given sketch
def Extrude_sketch(drawing_plane, align_edge,
    sketch_instance, profile_name):

    # Internal variable
    p = mymodel.parts['Base']

    # Get the face to extrude to
    bottom_extrude_face = p.sets['Sketch_Plane_Bottom'].faces[0]

    # Extrude the sketch
    p.ShellExtrude(sketchPlane = drawing_plane,
        sketchUpEdge = align_edge, upToFace = bottom_extrude_face,
        sketchPlaneSide=SIDE1, sketchOrientation=RIGHT,
        sketch=sketch_instance, flipExtrudeDirection=ON)

    # Delete the sketch
    del mymodel.sketches[profile_name]
```

Figura 166 - Função responsável pela extrusão de um sketch (*Extrude_sketch*)

3.5.3.6.6 Exemplos de geometrias geradas pelo algoritmo de desenho

Com o presente algoritmo de desenho, é possível gerar frisos com diferentes medidas de hexágonos, mas também com zonas de aplicação de reforço diferentes. Na Figura 167, Figura 168 e Figura 169, estão presentes alguns exemplos de geometrias geradas por este algoritmo.

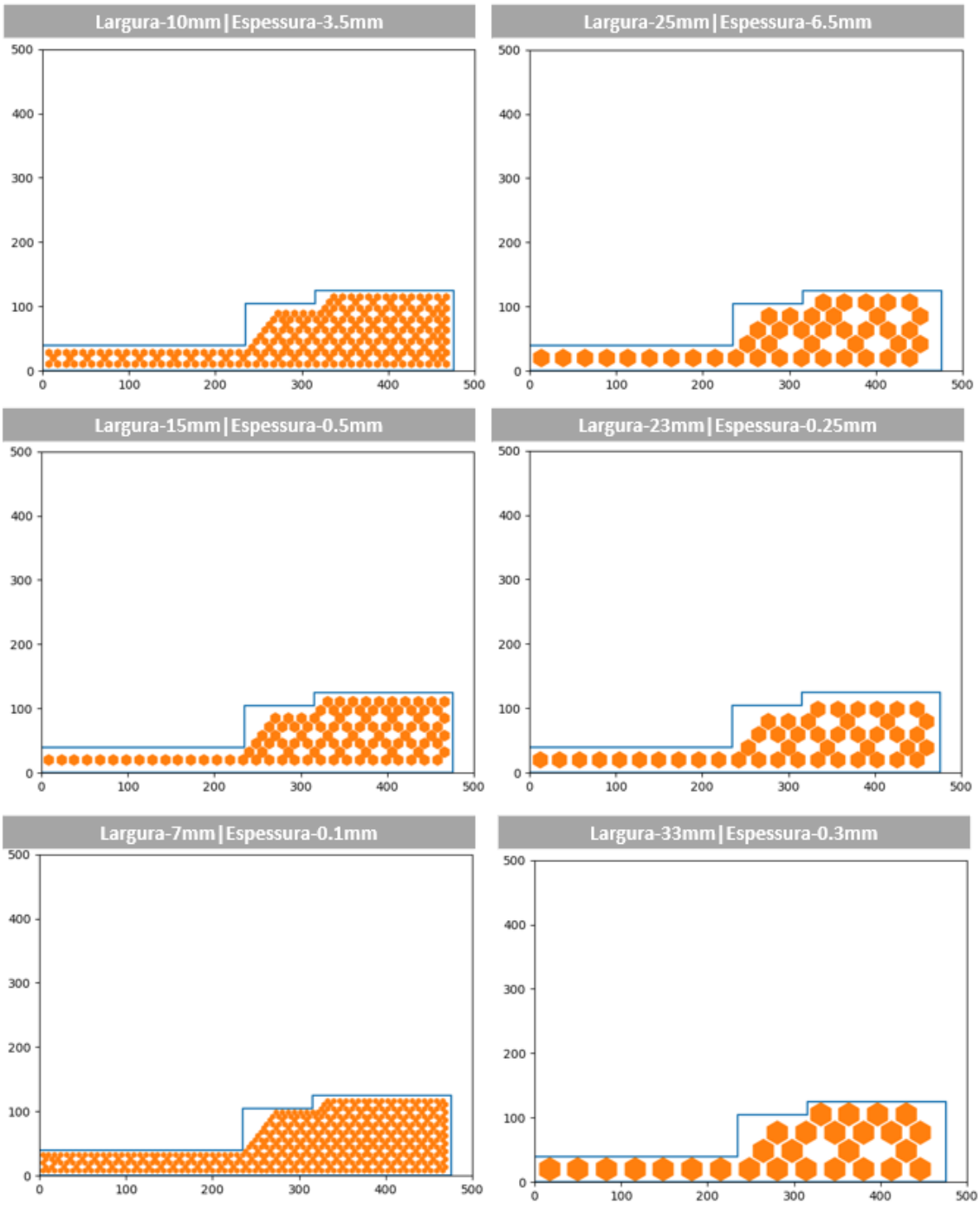


Figura 167 – Exemplos de geometrias de frisos geradas pelo algoritmo de desenho (1/3)

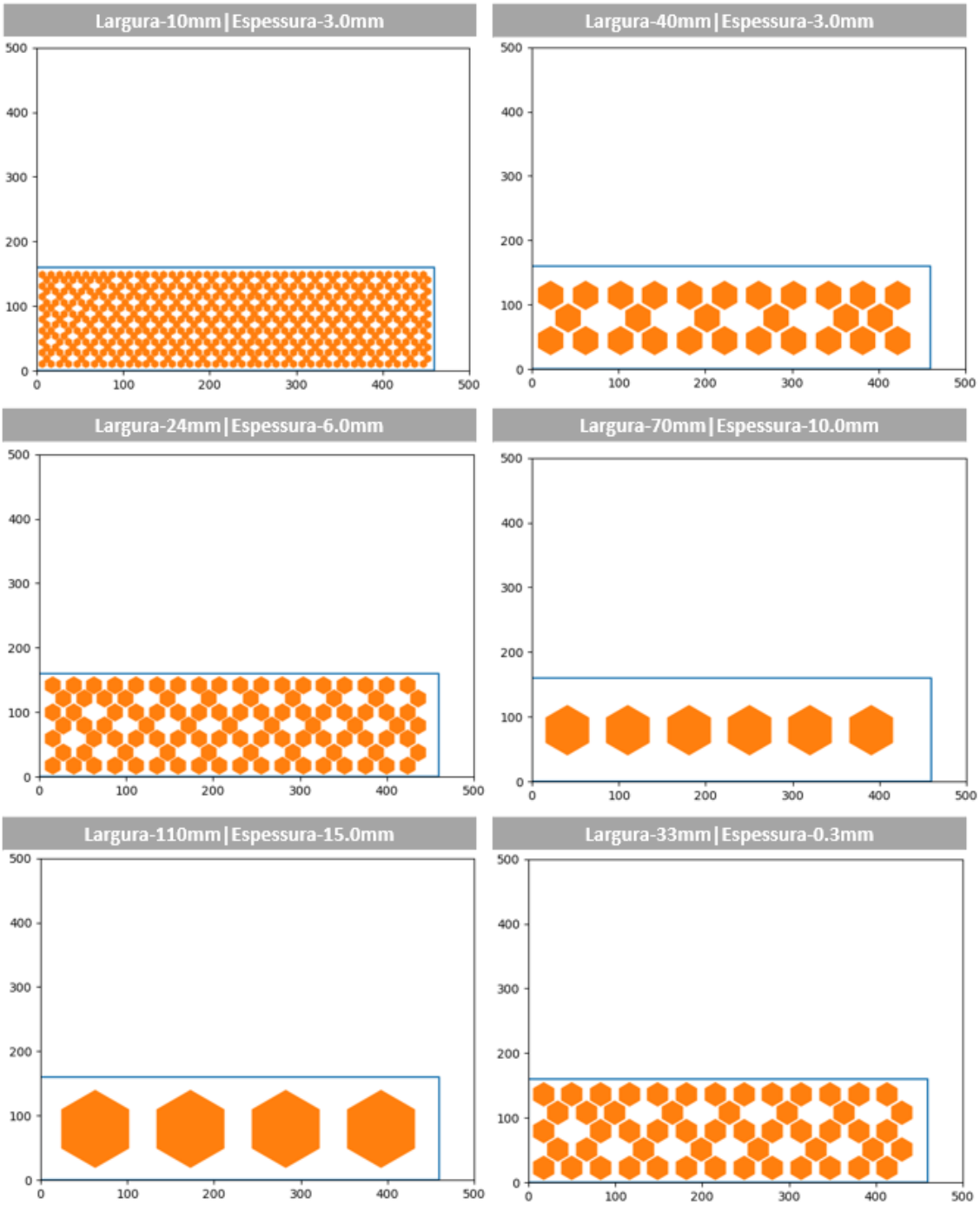


Figura 168 – Exemplos de geometrias de frisos geradas pelo algoritmo de desenho (2/3)

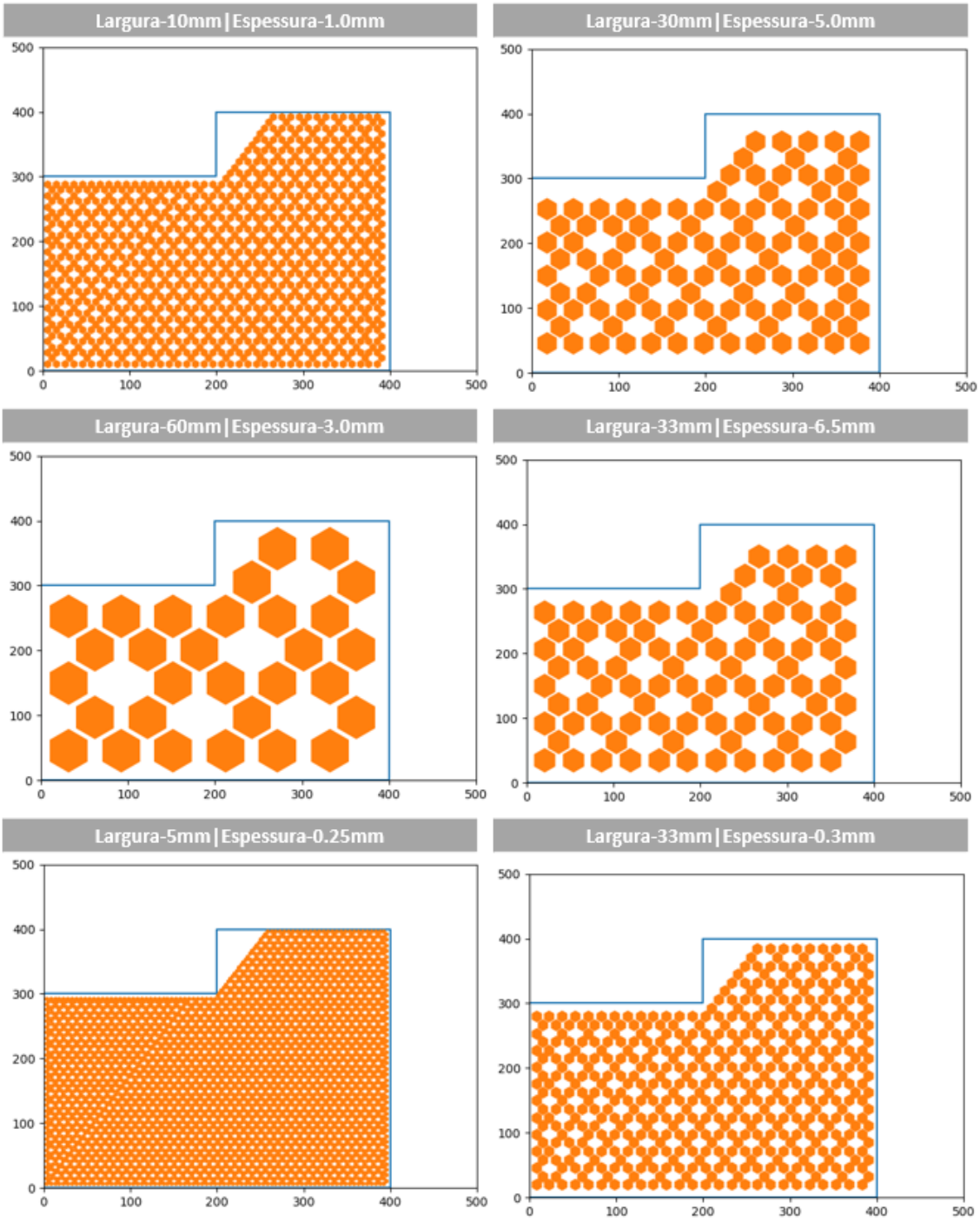


Figura 169 – Exemplos de geometrias de frisos geradas pelo algoritmo de desenho (3/3)

3.5.3.7 Corte dos frisos

No Capítulo 3.5.3.5 mencionou-se a necessidade de o utilizador de ter de criar dois planos, o "*Sketch_Plane*" e "*Sketch_Plane_Bottom*", para que o programa consiga extrudir os frisos na peça em que se pretende realizar a análise. Estes dois planos servem de base para a metodologia implementada no programa, para permitir que a extrusão dos frisos possa ser feita, tanto em superfícies regulares, como irregulares, e, portanto, aumentar o leque de peças que podem ser analisadas através da ferramenta. Como já foi brevemente introduzido no capítulo dos pré-requisitos, a metodologia empregada toma proveito do facto de, dentro do Abaqus®, quando se extrude um elemento até uma determinada posição, se o elemento a ser extrudido intersetar qualquer outro no seu trajeto, os dois sofrem uma partição automática, dividindo os dois elementos nos pontos de interseção entre eles. Este processo pode ser facilmente visualizado, fazendo-se extrudir uma face através de um trajeto que intersete uma terceira, como demonstrado na Figura 170. Aqui pode-se ver, que apesar de inicialmente só existir uma face, por cada lado do friso, cada vez que estas trespassam uma terceira, o friso fica repartido através das interseções, gerando assim três padrões de frisos com alturas diferentes. Isto é muito importante, porque, em determinadas peças, como a da Figura 171, como a superfície onde são inseridos os frisos é muito irregular, o Abaqus® não permite extrudir diretamente até à respetiva face. Sendo assim, a metodologia que se desenvolveu para o programa, consiste em extrudir os frisos, para lá da face onde estes devem ser inseridos, e, como o Abaqus® faz a partição automática, estes ficam repartidos com a forma necessária, para ficarem congruentes com a peça. Os limites de extrusão que são utilizados são então o "*Sketch_Plane*" e o "*Sketch_Plane_Bottom*", mais propriamente, o "*Sketch_Plane*" é o plano onde os frisos são desenhados, e o "*Sketch_Plane_Bottom*" é o plano até onde estes vão ser extrudidos. É importante que os dois planos tenham a face da peça onde é necessário colocar os frisos, a intercetar o trajeto que vai de uma a outra.

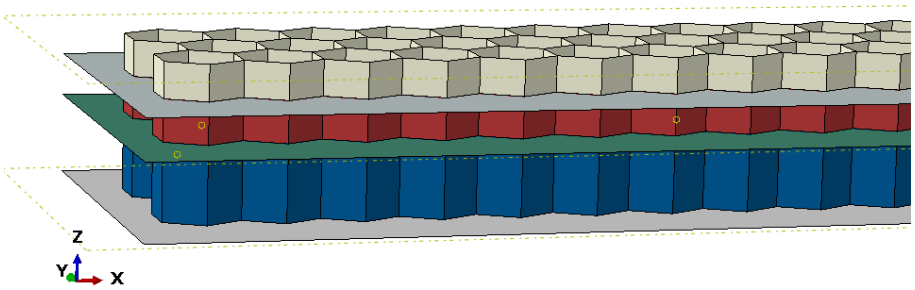


Figura 170 – Partição automática, no processo de extrusão

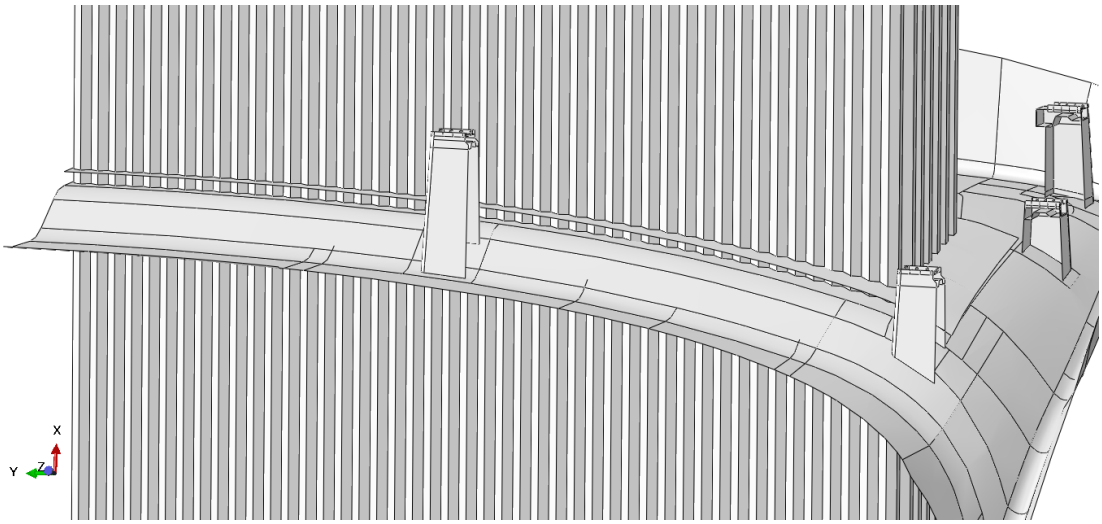


Figura 171 – Partição automática, no processo de extrusão, da peça a estudar

Para além de se permitir que os frisos ficam repartidos segundo a face onde devem de ser colocados, esta metodologia também permite conferir-lhes uma altura regular por toda a sua extensão. Para isto simplesmente é necessário fazer um offset da face onde estes vão ser colocados, a uma distância que lhes permita conferir a altura necessária. Assim, durante a extrusão, os frisos vão ser repartidos duas vezes, uma pela face do offset, e outra pela face que lhes serve de base. Entre as duas faces referidas, ficam desenhados os frisos com os parâmetros corretos, e apenas passa a ser necessário cortar os excessos (Figura 172).

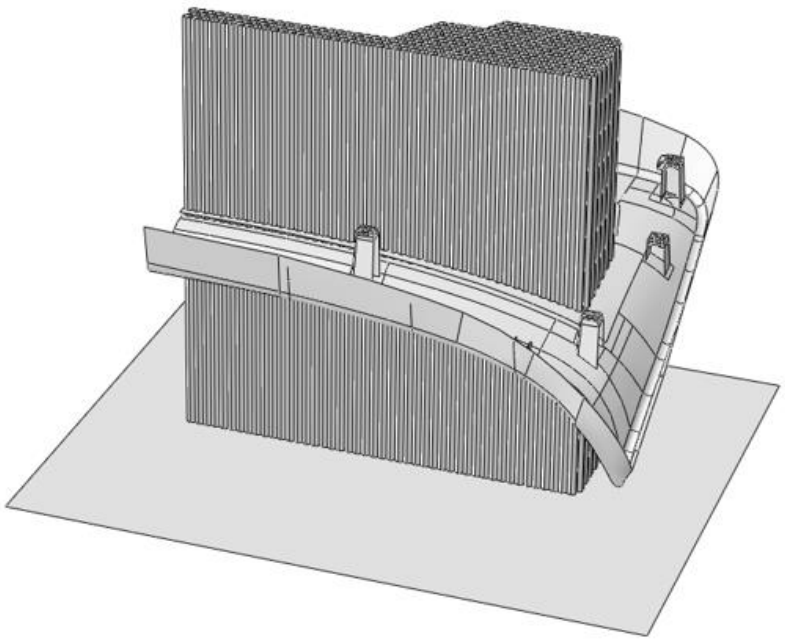


Figura 172 - Frisos com excessos

Como o programa trata de modelos de casca, em plano médio, o cálculo para a distância do offset, que permite conferir a altura necessária dos frios, deve contabilizar com a porção de material correspondente a metade da espessura da base onde estes se inserem (Figura 173)(equação (58)).

$$\text{offset\_distance} = -(hex\_height + \frac{base\_thickness}{2}) \quad (58)$$

onde,

- $offset\_distance$ – "Distância do offset" (mm)
- $base\_thickness$– "Espessura da base" (mm)

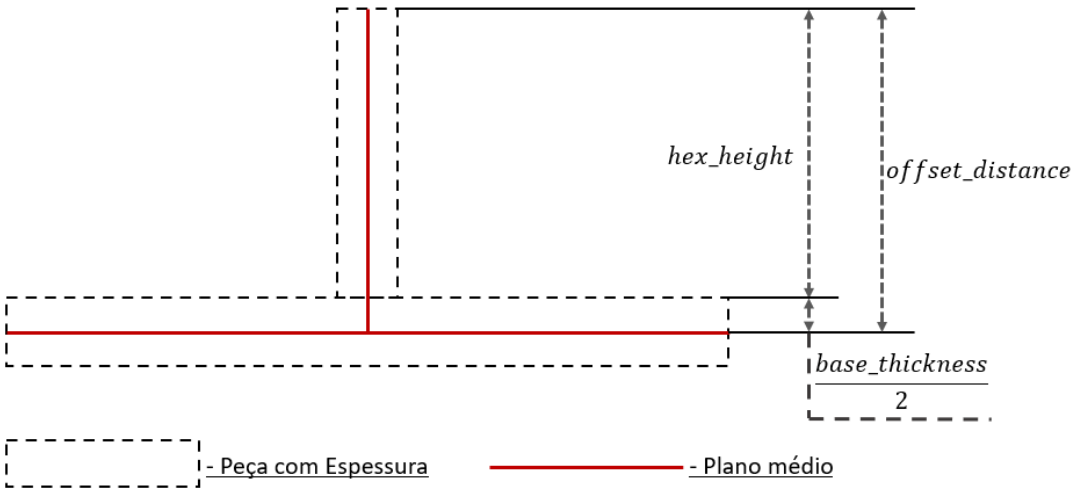


Figura 173 -Ilustração do cálculo da distância do *offset*

O processo de criação do offset para a partição dos frisos, é regido pela função *Create-Particion_Base* (Figura 174), que recebe como argumentos a altura do friso e a espessura da base (Linha 2), e com estes valores calcula a distância necessária para o offset (Figura 174 Linha 8), onde a partir do método *OffsetFaces* cria um offset da face que está guardada no set Base, à distância previamente calculada (Figura 174 Linhas 10-12). Após criado o offset, a função acede à lista de features da árvore do Abaqus®, vai buscar a última feature que foi criada (Figura 174 Linha 15) [73], e a face que lhe está associada (Figura 174 Linha 18) [73]. A face do offset é então guardada dentro de um set, para que mais tarde possa ser removida (Figura 174 Linha 21).

```
# Create an offset to be used as a particion face
def Create_Particion_Face(hex_height, base_thickness):

    # Internal variables
    p = mymodel.parts['Base']

    # Create Particion Face
    offset_distance = (hex_height+base_thickness/2)*-1

    p.OffsetFaces(faceList = p.sets['Base'].faces,
        distance= offset_distance,
        trimToReferenceRep=False)

    # Get the name of the last feature created
    offset_name = p.features.summary()[-1][0]

    # Get faces from the feature offset_name
    offset_faces = p.getFeatureFaces(name = offset_name)

    # Create Particion Face Set
    p.Set(faces=offset_faces, name='Particion_Base')
```

Figura 174 - Função para criar a face do offset (*Create_Particion_Face*)

No que toca ao corte dos excessos, o processo é mais complexo, e é conseguido através de funcionalidades do API do Abaqus® [69]. O API possui um método chamado de *getByBoundingBox*, que, através de dois pontos, cria uma caixa dentro do modelo e apanha todos os elementos que são totalmente abrangidos pela respetiva caixa [74]. Como, para que uma face seja detetada por este método, ela tem de estar completamente cobrida pela caixa, e muitas vezes a geometria da peça não permite que as faces sejam todas colocadas, sem que nenhuma outra face seja também apanhada, o método *getByBoundingBox*, não vai procurar as faces que devem de ser eliminadas, mas sim as arestas. As arestas que se pretendem que sejam apanhadas pela caixa, são as arestas que se encontram inseridas dentro do "*Sketch_Plane*" e "*Sketch_Plane_Bottom*", visto que essas vão estar sempre na mesma posição, e é possível selecioná-las, sem correr-se o risco de selecionar uma outra face do modelo (Figura 175 e Figura 176).

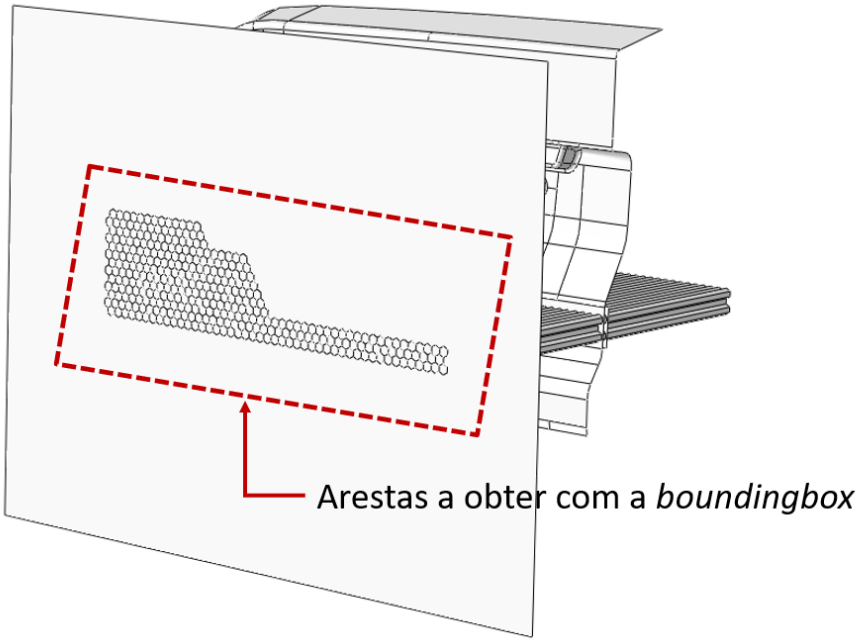


Figura 175 – Arestas a ir buscar com a *boundingbox (Sketch_Plane_bottom*)

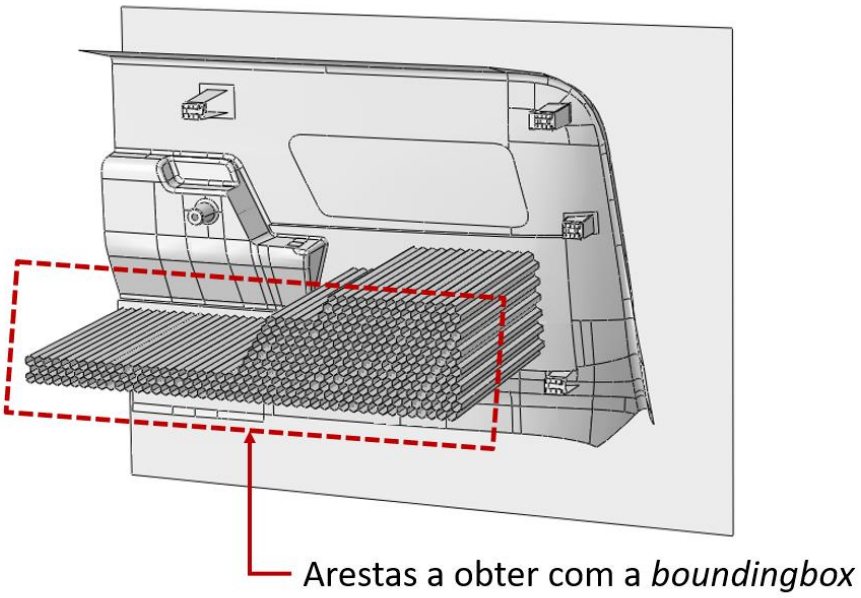


Figura 176 – Arestas a ir buscar com a *boundingbox (Sketch_Plane*)

Com as arestas selecionadas através da *bounding box*, o programa tem de agora descobrir que faces pertencem a estas arestas, e para tal utiliza outro método do API,

o *getFaces*, do objeto *Edge*. Este método retorna as keys das faces cuja aresta recebida faz parte da sua geometria [75]. Com isto em mente, o programa itera por todas as arestas que recebe da *bounding box*, e procura que faces é que estão ligadas à respetiva aresta. No final obtém-se uma lista com todas as chaves das faces que fazem parte desta zona em excesso e eliminam-se todas as que constam nela. Contudo, antes das faces serem eliminadas, o programa utiliza esta lista, para criar um set com os frisos da peça. O set referido é obtido graças ao método *getAdjacentFaces*, que quando dada uma face, devolve todas as que são adjacentes a ela [76], permitindo assim obter um *set* com as faces dos frisos, e os excessos. Quando os excessos são eliminados pelo programa, o que resta dentro do *set* são os frisos da peça.

Para que o processo de corte seja bem executado, o utilizador tem de introduzir quais são os pontos que são utilizados para limitar a *bounding box*, e, portanto, estes têm de ser introduzidos no modelo que se quer estudar, antes de se iniciar o programa. Neste modelo, os pontos têm de possuir os seguintes nomes:

- "*Top_Bounding_Box_P1*" – Ponto 1 da bounding box do "*Sketch_Plane*";
- "*Top_Bounding_Box_P2*" – Ponto 2 da bounding box do "*Sketch_Plane*";
- "*Bottom_Bounding_Box_P1*" – Ponto 1 da bounding box do "*Sketch_Bottom_Plane*";
- "*Bottom_Bounding_Box_P2*" – Ponto 2 da bounding box do "*Sketch_Bottom_Plane*";

O processo de extrusão é implementado dentro do programa, através da função *Extrude_Sketch*, apresentada no Capítulo 3.5.3.6.5 (Figura 166), sendo que os argumentos que são introduzidos na função, neste caso, são as propriedades do "*Sketch_Plane*" e do "*Sketch_Plane_Bottom*", para o respetivo perfil de desenho que se pretende extrudir. Como a função do capítulo anterior foi desenvolvida de forma genérica, esta pode ser utilizada em diferentes partes do programa, desde que a informação necessária seja introduzida corretamente. Isto permite que o *sketch* que se pretende extrudir utilize o "*Sketch_Plane*" como o plano de desenho, e o "*Sketch_Plane_Bottom*" como o alvo da extrusão. Dentro da função o processo de extrusão é regido pelo método *shellExtrude* [77].

Quanto ao processo de corte, este possui a sua própria função, a *Trim_Hex_Assembly* (Figura 179). Esta função recebe como argumentos os nomes dos pontos a utilizar na *bounding box*, e procura as coordenadas destes pontos, para as poder introduzir dentro do *getByBoundingBox* (Figura 179 Linhas 12-15). O getByBoundingBox recebe como argumentos dois sets de coordenadas tridimensionais, um para cada ponto da caixa, contudo estes não podem ser introduzidos de qualquer maneira, os valores têm de ser introduzidos por ordem crescente [74]. Se fosse necessário criar uma *bounding box* com as características da Figura 177 [78], e o utilizador introduzisse como parâmetros, os pontos E e C como sendo respetivamente o ponto 1 e o ponto 2 da *bounding box*, estas coordenadas teriam de ser reorganizadas para que o método não

devolva um erro. Esta reorganização é feita comparando cada uma das coordenadas de cada ponto, e avaliar qual delas é a menor e a maior de um respetivo par, sendo depois introduzidas por ordem, nos argumentos da *bounding box*. No caso mencionado, o menor valor para as coordenadas x e y corresponderiam às componentes do ponto 1, porém na coordenada z, o oposto é verdade, e, portanto, os dados teriam de ser organizados segundo a forma presente na Figura 178.

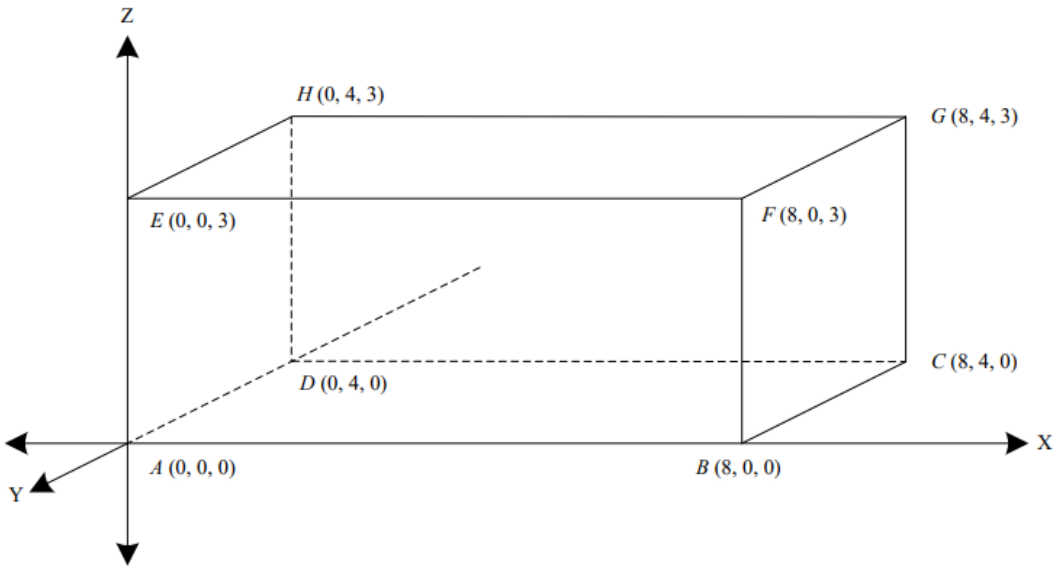


Figura 177 – Exemplo de uma *bounding box* em contexto tridimensional [78]

P1

Coordenadas		
X1	Y1	Z1
0	0	3

P2

Coordenadas		
X2	Y2	Z2
8	4	0

Bounding Box

Argumentos					
X min	X max	Y min	Y max	Z min	Z max
X1 = 0	X2 = 8	Y1 = 0	Y2 = 4	Z2 = 0	Z1 =3

Figura 178 - Conversão de dados para a *bounding box*

O processo de reorganização das coordenadas é realizado dentro da função *Trim_Hex_Assembly*, nas linhas 18-24 da Figura 179, e são introduzidas na *bounding box* nas linhas 27 e 28. Com a lista de arestas obtida, o programa vai buscar as faces associadas a cada uma das arestas e guarda-as em duas listas distintas, uma que guarda a chave da respetiva face (*del_face_list*) (Linha 37), e outra o *face object* correspondente (*del_face_object*) (Linha 38). A primeira é utilizada para procurar as

faces que correspondem aos frisos, na função *Frizes_Set* (Figura 180), e a segunda serve para criar um set que englobe todas as faces do excesso que tem de ser removido. As faces do excesso têm de ser colocadas num set para permitir que o processo de corte dos frisos seja o mais rápido possível, pois, ao introduzi-las dentro do set, a função *Delete_Set* (Figura 181) consegue apagar todas as faces do excesso ao mesmo tempo, o que leva a que o programa só tenha de regenerar a árvore do modelo uma única vez, em vez de o fazer após apagar cada face. Isto é muito importante, pois reduz consideravelmente o tempo de execução do algoritmo, e este processo já é dos mais demorados do programa.

A função *Trim_Hex_Assembly* foi desenvolvida de forma genérica, para se poder utilizar várias vezes dentro do programa, o que acontece, pois, o processo de corte tem de ser feito tanto para os frisos em excesso da zona superior como inferior, em que cada um possui um par de pontos diferentes que delimitam a *bounding box*. Assim, na parte principal do programa, para se apagar os frisos em excesso basta chamar a função *Trim_Hex_Assembly* duas vezes, uma com a "*Top_Bounding_Box_P1*" e a "*Top_Bounding_Box_P2*" como argumentos, e a outra com a "*Bottom_Bounding_Box_P1*" e a "*Bottom_Bounding_Box_P2*". A parte da função responsável por criar o Set dos frisos, só é utilizada quando a *bounding box* a desenhar for a da zona superior (Linha 43-45), visto que é nesta zona onde se encontram os frisos.

```
# Function to trim the Frizes, so that they are at the appropriate height
def Trim_Hex_Assembly(bounding_box_point_1_datum,
    bounding_box_point_2_datum):

    # Internal variables
    p = mymodel.parts['Base']
    f = p.faces
    s = p.edges
    d = p.datums

    # Boundary box Coordinates
    bounding_box_point_1_id = p.features[bounding_box_point_1_datum].id
    bounding_box_point_2_id = p.features[bounding_box_point_2_datum].id
    bounding_box_point_1 = p.getCoordinates(d[bounding_box_point_1_id])
    bounding_box_point_2 = p.getCoordinates(d[bounding_box_point_2_id])

    # Reordering of the boundary box coordinates
    min_x_top = min(bounding_box_point_1[0],bounding_box_point_2[0])
    min_y_top = min(bounding_box_point_1[1],bounding_box_point_2[1])
    min_z_top = min(bounding_box_point_1[2],bounding_box_point_2[2])

    max_x_top = max(bounding_box_point_1[0],bounding_box_point_2[0])
    max_y_top = max(bounding_box_point_1[1],bounding_box_point_2[1])
    max_z_top = max(bounding_box_point_1[2],bounding_box_point_2[2])

    # Get boundary edges from the frizes trim area
    hex_boundary_edges_region = s.getByBoundingBox(min_x_top,min_y_top,
        min_z_top,max_x_top,max_y_top,max_z_top)

    # Get the faces associated with each boundary edge and delete them
    del_face_region = []
    del_face_list = []
    for edge_index in range(len(hex_boundary_edges_region)):
        face_from_edge_tuple = hex_boundary_edges_region[edge_index] \
            .getFaces()
        face_from_edge = face_from_edge_tuple[0]
        del_face_list.append(face_from_edge)
        del_face_region.append(f[face_from_edge:(face_from_edge+1)])
    p.Set(faces=del_face_region, name='Set-remove')

    # Create a set with the frizes faces,
    # if they are from the top bounding box
    if (bounding_box_point_1_datum == 'Top_Bounding_Box_P1') or \
        (bounding_box_point_2_datum == 'Top_Bounding_Box_P2'):
        Frizes_Set(del_face_list)

    # Delete the frizes that are in excession
    Delete_Set('Set-remove')
```

Figura 179 - Função do corte dos frisos (*Trim_Hex_Assembly*)

```python
# Create a Set with the Frizes in it
def Frizes_Set(del_face_region):

    # Internal Variables
    p = mymodel.parts['Base']
    f = p.faces
    friezes_region = []
    friezes_set_region_id = []
    friezes_set_region = []

    # Get adjacent faces from the ones that are in excess
    for i in range(len(del_face_region)):
        friezes_region.extend(f[del_face_region[i]].getAdjacentFaces())

    # Get the adjacent faces index number
    for i in range(len(friezes_region)):
        friezes_set_region_id.append(friezes_region[i].index)

    # Get the adjacent faces in a list, in a form to create a set
    for i in range(len(friezes_set_region_id)):
        friezes_set_region.append( \
            f[friezes_set_region_id[i]:(friezes_set_region_id[i]+1)])

    # Create the Set
    p.Set(faces=friezes_set_region, name='Frizes')
```

Figura 180 – Função para criar o set dos frisos (*Frizes_Set*)

```python
# Delete the faces inside a given set
def Delete_Set(set_for_deletion):

    # Internal variables
    p = mymodel.parts['Base']

    # Remove the faces inside the set that was
    # received as an argument
    set_deletion_region = p.sets[set_for_deletion].faces
    p.RemoveFaces(faceList = set_deletion_region, deleteCells=False)
```

Figura 181 – Função para apagar um Set (*Delete_Set*)

3.5.3.8 *Leitura do material e atribuição de secções*

O material utilizado neste tipo de peças é uma variante do polipropileno, reforçado com fibra de vidro. Devido a acordos de confidencialidade, a designação do material, o nome do fornecedor e os pontos das curvas, não podem ser divulgados.

No que diz respeito aos dados do material utilizado, quando se entrou em contacto com o fornecedor, este forneceu os dados das curvas tensão-deformação obtidos através de ensaios de tração realizados com velocidades, e temperaturas de ensaio diferentes. Sendo assim para a resolução da tese disponibilizaram-se os dados do material para as temperaturas de:

- Temperatura de 23ºC (Figura 182);
- Temperatura de -30ºC (Figura 183);
- Temperatura de 80ºC (Figura 184);

em que para cada uma delas se realizaram ensaios a seis velocidades diferentes:

- Velocidade de 0.06 mm/s – *Strain rate* de 0.001 /s;
- Velocidade de 0.6 mm/s – *Strain rate* de 0.01 /s;
- Velocidade de 6.0 mm/s – *Strain rate* de 0.1 /s;
- Velocidade de 60.0 mm/s – *Strain rate* de 1.0 /s;
- Velocidade de 1000.0 mm/s – *Strain rate* de 16.6 /s;
- Velocidade de 6000.0 mm/s – *Strain rate* de 100.0 /s;

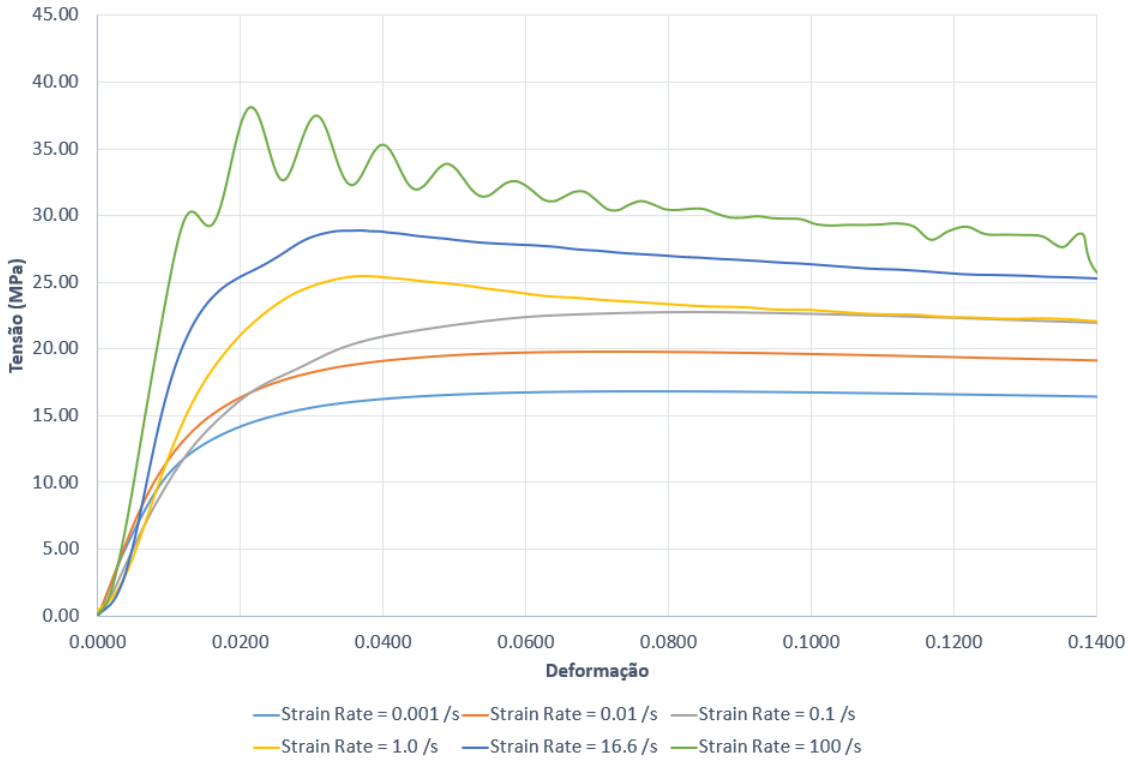


Figura 182 – Curvas de engenharia tensão-deformação para 23ºC

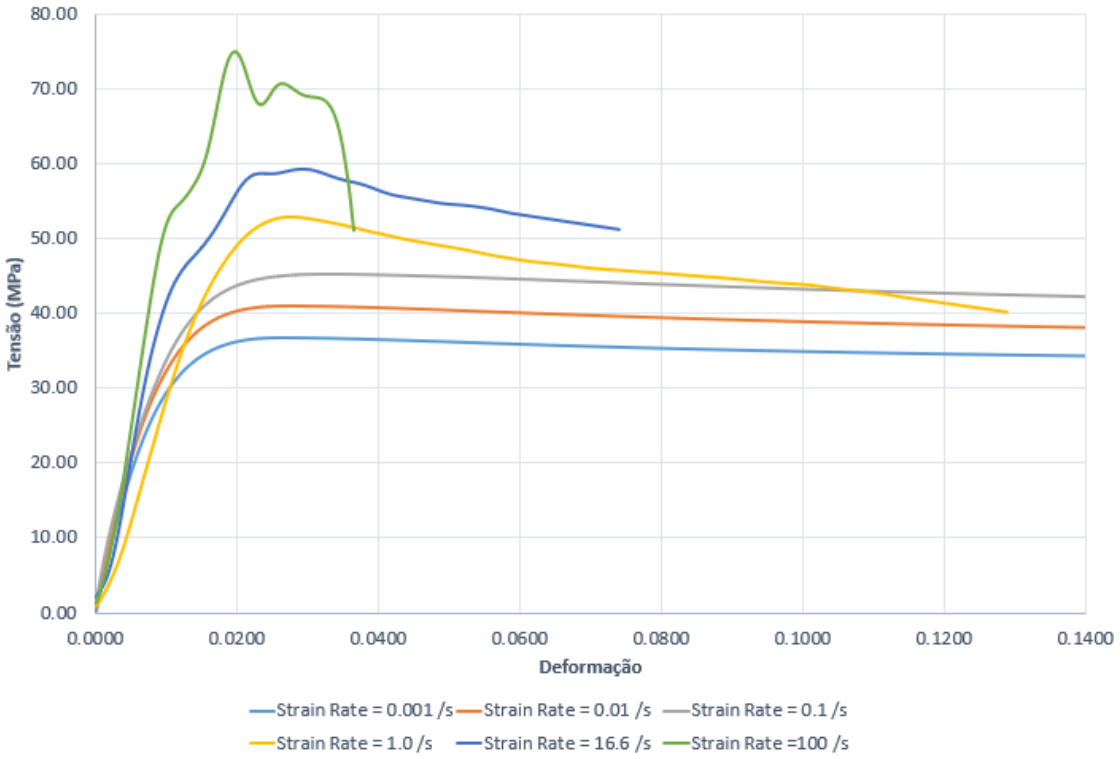


Figura 183 - Curvas de engenharia tensão-deformação para -30ºC

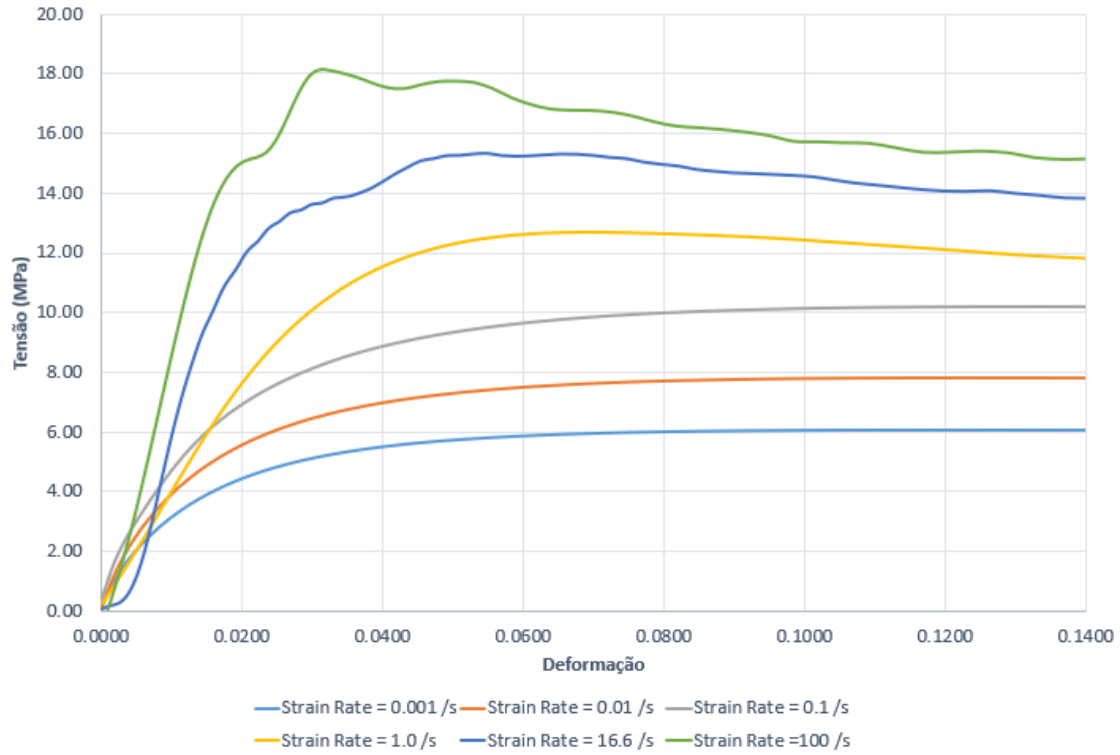


Figura 184 - Curvas de engenharia tensão-deformação para 80ºC

Como os ensaios a que se sujeitou a peça, encontram-se à temperatura ambiente, apenas foram analisados os dados para a temperatura de 23ºC. Para se obterem as propriedades do material, recorreu-se à norma ISO 527-1 [79] e à ISO 527-2 [80], tendo-se obtidos os módulos de elasticidade, tensão de cedência, tensão de rotura e as deformações respetivas a estes dois pontos, para cada uma das velocidades de ensaio. O modulo de elasticidade na maioria dos casos obteve-se através do cálculo apresentado na ISO 527-1 (equação (59)):

$$E_y = \frac{\sigma_2 - \sigma_1}{\varepsilon_2 - \varepsilon_1} \tag{59}$$

onde:

- E_y – "Modulo de elasticidade" (MPa)
- σ_1 – "Valor de tensão obtido para a deformação ε_1" (MPa)
- σ_2 – "Valor de tensão obtido para a deformação ε_2" (MPa)
- ε_1 – "Deformação no ponto 1" = 0.0005
- ε_2 – "Deformação no ponto 2" = 0.0025

Este offset de 0.0005 é utilizado na norma, de forma a contrariar-se as possíveis folgas da máquina, existentes no início do ensaio de tração, porém no caso de alguns dos ensaios, este offset não era suficiente, e, portanto, utilizaram-se intervalos de valores que fossem mais representativos das zonas lineares do material, para a obtenção do módulo de elasticidade, nos casos mencionados. Os pontos de cedência e de rotura foram obtidos conforme a especificação da norma 527-1.

Com as propriedades do regime elástico obtidas, procedeu-se então com a conversão das curvas de engenharia, para curvas reais, e para tal calcularam-se os valores da tensão real e deformação real para cada ensaio. Estas conversões são feitas seguindo as seguintes equações (60) e (61):

$$\sigma_{real} = \sigma_{engenharia} * (1 + \varepsilon_{engenharia}) \tag{60}$$

$$\varepsilon_{real} = \ln(1 + \varepsilon_{engenharia}) \tag{61}$$

onde:

- σ_{real} – "Tensão real" (MPa)
- $\sigma_{engenharia}$ – "Tensão de engenharia" (MPa)
- ε_{real} – "Deformação real"
- $\varepsilon_{engenharia}$ – "Deformação de engenharia"

Estes cálculos só são válidos até ao ponto em que ocorre estricção do material, que ocorre no ponto de tensão máxima, a partir daí, estas equações não podem ser utilizadas para calcular o seu comportamento. Porém, como as especificações do cliente mencionam que, para os ensaios a estudar, a peça não deve ultrapassar o ponto de estricção, o comportamento da curva a partir deste ponto não é necessário para o estudo, basta apenas que o comportamento até este ponto esteja correto. Sendo assim, e de maneira a diminuir a complexidade computacional do modelo, a informação introduzida no software, é que o comportamento é linear entre o ponto de tensão máxima e o ponto de rotura, como ilustrado na Figura 185.

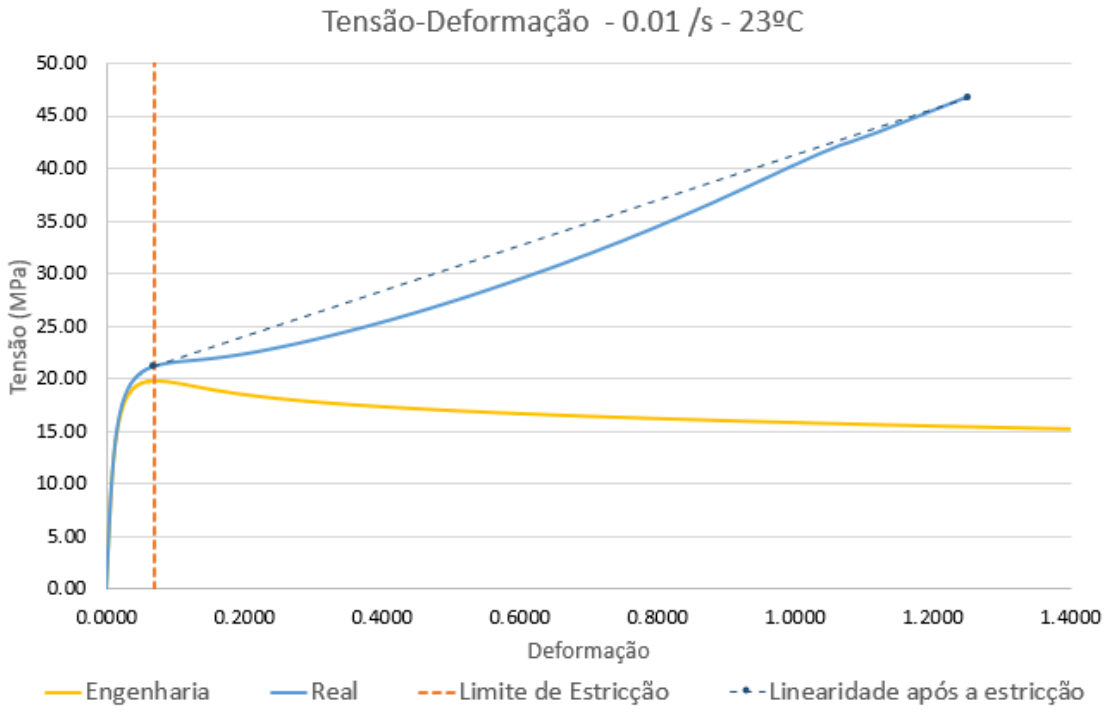


Figura 185 – Conversão da curva de engenharia para a real

Com os valores da curva real calculados, foi também necessário mover a zona do regime plástico de cada uma das curvas, para o ponto de deformação 0 (equação (62)). Isto é feito, para que os valores desta zona possam ser introduzidos no software Abaqus®:

$$\varepsilon_{Plástica\ efetiva} = \varepsilon_{total} - \varepsilon_{cedência} \tag{62}$$

onde,

- $\varepsilon_{Plástica\ efetiva}$ – "Deformação plástica a inserir no Abaqus®"
- ε_{total} – "Deformação total"
- $\varepsilon_{cedência}$ – "Deformação no ponto de cedência do material"

Com os parâmetros das curvas reais calculados, simularam-se os ensaios de tração dentro do Abaqus®. Após realizado cada ensaio compararam-se os valores obtidos no Abaqus®, com os dados experimentais do fornecedor, de forma a confirmar se a análise do material tinha sido feita corretamente. Como a quantidade de pontos que se fornece ao Abaqus® aumenta o tempo de simulação, foram também feitas análises iterativas, no qual se foram reduzindo o número de pontos de cada curva. Isto foi feito até se descobrir qual é a menor quantidade de pontos que se deve colocar dentro do Abaqus®, para que os dados experimentais e do fornecedor coincidam (Figura 186). Após a convergência do material, notou-se que a partir de 25 pontos já se consegue uma boa aproximação dos resultados, e, portanto, não é necessário utilizar os 200 pontos que são dados pelo fornecedor.

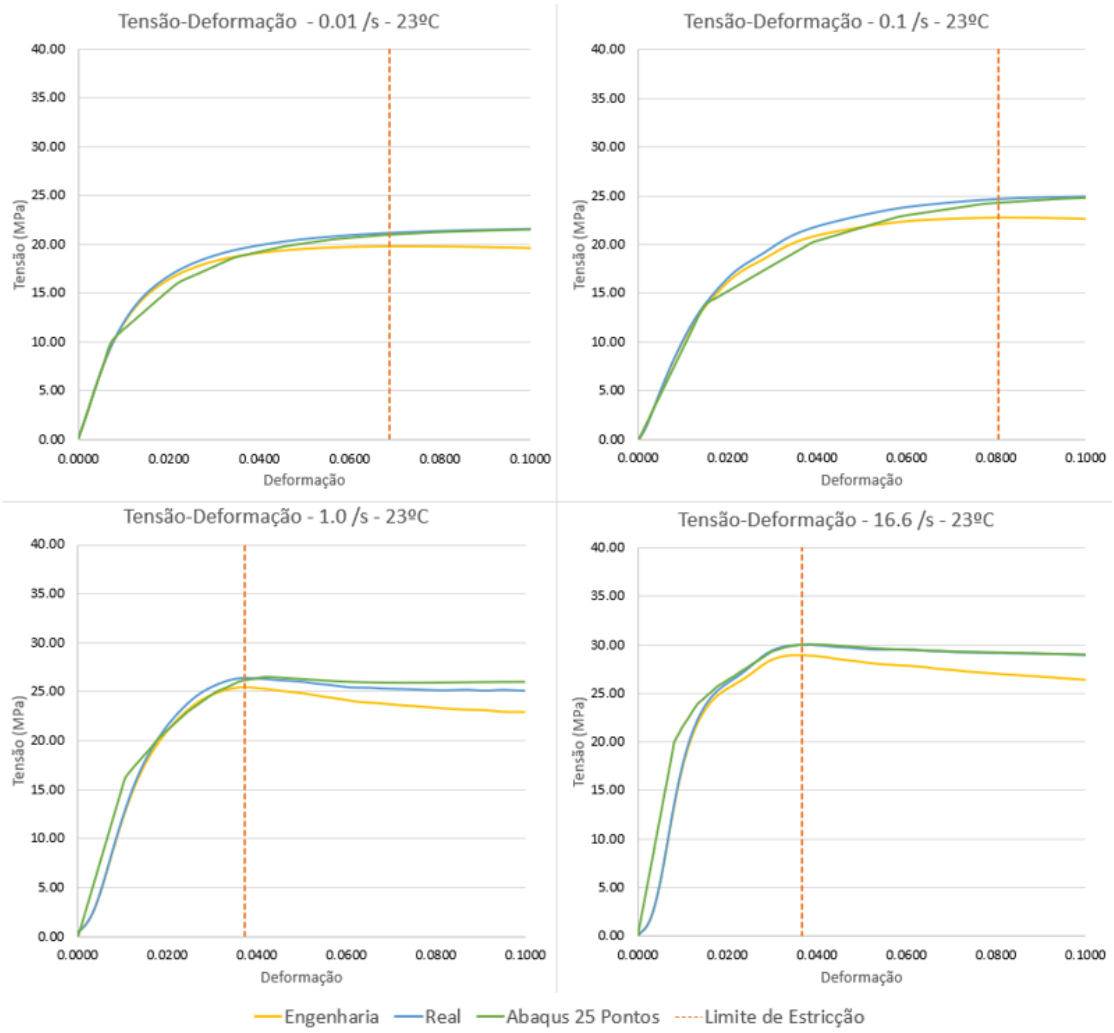


Figura 186 – Dados da convergência do material para diferentes velocidades de deformação

Com a convergência do material feita, sujeitou-se a peça aos dois ensaios que o programa tem de realizar e analisou-se a velocidade de deformação máxima em cada um. Para o ensaio estático, a velocidade de deformação obtida foi de 0.085 /s e para o *Ball Drop* foi de 16.190 /s. Com estes valores em mente, as curvas que seriam mais representativas do comportamento do material, seriam a de 0.1 para o ensaio estático, e a de 16.6 para o *Ball Drop*. Contudo no caso do estático optou-se antes por se utilizar a de 0.01, porque, como pode ser observado na Figura 182, com o aumento da velocidade de ensaio, era expectável que o material demonstrasse um comportamento mais frágil, porém, a curva de 0.1 apresenta um declive da sua parte elástica inferior aos dois que lhe precedem, o que leva a suspeitar que algo de errado terá acontecido durante o seu ensaio, e portanto optou-se por utilizar outra curva que se aproximasse mais da velocidade do ensaio estático, que neste caso é a de 0.01. Sendo assim, os dados calculados para as propriedades do material, para as duas velocidades de ensaio mencionadas são:

Tabela 8 – Dados do material

Velocidade de deformação (/s)	Módulo de Elasticidade (MPa)	Tensão de cedência (MPa)	Deformação no ponto de cedência	Tensão Máxima (MPa)	Deformação no ponto de tensão máxima (MPa)	Deformação plástica limite
0.01	1384.69	10.26	0.0080	19.77	0.0697	0.0545
16.6	2445.68	20.26	0.0117	28.92	0.0366	0.0248

Com os materiais tratados, os valores de cada um foram adicionados a ficheiros de texto diferentes, e guardados numa pasta do computador que serve de base de dados para o programa. Dentro destes ficheiros, os dados estão organizados da seguinte forma (Figura 187):

-Organização do ficheiro do material-

Linha	Coluna						
	Def. Plástica	Tensão (MPa)	Modulo de Elasticidade (MPa)	Poisson	Massa volúmica (Ton/mm^3)	Tensão máxima (MPa)	Def. Plástica limite
1	0	20.26	2445.68	0.4	0.000000001	28.92	0.0248
2	0.0034	23.74					
3	0.0068	25.52					
4	0.010	26.61					
5	0.014	27.79					
6	0.017	29.06					

Figura 187 – Exemplo da organização do ficheiro de materiais (Os dados não são os do material)

Quando o programa necessita de introduzir as propriedades do material, este recorre à função *Read-True_Stress_Strain_Values*, (Figura 188) que recebe como argumento a velocidade de ensaio e acede ao respetivo ficheiro de texto, onde os dados estão armazenados, e retira a informação de cada uma das suas variáveis(Figura 188 Linhas 4-32). Os dados relativos aos pontos da zona plástica, para poderem ser lidos pelo Abaqus® têm primeiro de ser convertidos para uma estrutura de *tuple*, no qual se combinam os dados dos pontos da deformação e da tensão (Figura 188 35-44). Com tudo terminado, a função devolve ao sistema os valores da base de dados (Figura 188 Linha 46 e 47). Com estes dados, o programa chama a função *Atribute_thickness* (Figura 189), que se encarrega de criar secções para os frisos e para a face onde estes se inserem (Figura 188 Linhas 13-29), e depois atribui-las a esses mesmos *sets* (Figura 188 Linhas 32-41).

```
# Read the data base of materials
def Read_True_Stress_Strain_Values(test_speed):

    if test_speed == 16.6:
        material_file_name = (r'D:\Gabriel Ramos\Tese\Teste do software\ \
            Base de Dados Material\*****\*****_16.6.txt')

    elif test_speed == 0.01:
        material_file_name = (r'D:\Gabriel Ramos\Tese\Teste do software\ \
            Base de Dados Material\******\******_0.01.txt')


    # Open the materaial file
    f = open(material_file_name)

    # Read the file
    lines=f.readlines()

    # Get the yield stress
    max_stress = (float(lines[0].split('      ')[5]))

    # Get the Elastic modulus
    modulus = (float(lines[0].split(' ')[2]))

    # Get the poisson modulus
    poisson = (float(lines[0].split(' ')[3]))

    # Get the density
    density = (float(lines[0].split(' ')[4]))

    # Get elastic energy coefficient
    limit_strain = (float(lines[0].split('    ')[6]))

    # Get the platicity curve values
    plasticity_curve_list = []
    for x in lines:
        stress_strain_line = x
        stress_strain_list = stress_strain_line.split('      ')
        plasticity_curve_list.append((float(stress_strain_list[1]),
        float(stress_strain_list[0])))
    f.close()

    # Convert the plasticity curve values into a tuple
    plasticity_curve_tuple = tuple(plasticity_curve_list)

    return plasticity_curve_tuple, max_stress, \
        modulus, poisson, density, limit_strain
```

Figura 188 – Leitura das bases de dados do material (*Read_True_Stress_Strain_Values*)

```
# Create the material properties and assign sections
def Atribute_thickness(density, modulus, poisson,
    hex_thickness, plasticity_curve_tuple):

    # Create Material Properties
    p = mymodel.parts['Base']
    mymaterial = mymodel.Material(name='Script Material')
    mymaterial.Density(table=((density, ), ))
    mymaterial.Elastic(table=((modulus, poisson), ))
    mymaterial.Plastic(table = plasticity_curve_tuple)

    #Create Section - Frizes Section
    frizes_section = mymodel.HomogeneousShellSection(
        name='Section-Frizes', preIntegrate=OFF,
        material='Script Material', thicknessType=UNIFORM,
        thickness=hex_thickness, thicknessField='',
        nodalThicknessField='', idealization=NO_IDEALIZATION,
        poissonDefinition=DEFAULT,
        thicknessModulus=None, temperature=GRADIENT, useDensity=ON,
        integrationRule=SIMPSON, numIntPts=5)

    #Create Section - Base Section
    base_section = mymodel.HomogeneousShellSection(
        name='Section-Base', preIntegrate=OFF,
        material='Script Material', thicknessType=UNIFORM,
        thickness=3.0, thicknessField='', nodalThicknessField='',
        idealization=NO_IDEALIZATION, poissonDefinition=DEFAULT,
        thicknessModulus=None, temperature=GRADIENT, useDensity=ON,
        integrationRule=SIMPSON, numIntPts=5)

    # Assign Section - Friezes Section
    p.SectionAssignment(region=p.sets['Frizes'],
        sectionName=frizes_section.name, offset=0.0,
        offsetType=MIDDLE_SURFACE, offsetField='',
        thicknessAssignment=FROM_SECTION)

    # Assign Section - Base Section
    p.SectionAssignment(region=p.sets['Base'],
        sectionName=base_section.name, offset=0.0,
        offsetType=MIDDLE_SURFACE, offsetField='',
        thicknessAssignment=FROM_SECTION)
```

Figura 189 – Propriedades do material e designação das secções (*Atribute_thickness*)

3.5.3.9 Ensaio Estático

Um dos ensaios impostos pelo cliente para a validação da peça, consiste na aplicação de uma força de 200 N, aplicada por um impactor cilíndrico de 30 mm de diâmetro, na superfície externa da peça que se quer estudar. Esta força demora 1s até atingir o seu valor máximo, e deve ser aplicada à temperatura ambiente (23ºC). Após o referido ensaio, a peça não pode apresentar uma deformação permanente acima dos 5.45%, e não pode ultrapassar a tensão máxima do material (19.77 MPa). Com isto em mente o programa tem de ser capaz de impor estas condições ao modelo da peça, realizar o ensaio e extrair os dados necessários da simulação.

Com os frisos desenhados, as secções atribuídas e a malha construída, resta ao programa impor as condições do ensaio que se pretende realizar, e para isto ele tem de ser capaz de:

1. Impor os contactos entre os diferentes componentes do modelo (Base e impactor);
2. Criar os steps necessários para o ensaio;
3. Criar as *boundary conditions*;
4. Criar o *field output*;
5. Criar o *history output*;
6. Submeter o job para ser calculado;
7. Analisar se o job foi concluído com sucesso;
8. Extrair os resultados da simulação;

Quanto à primeira etapa, o programa tem de impor o contacto entre o impactor e a superfície da peça que entra em contacto com ele. Para isso, primeiro têm de ser definidas quais são as faces que entram em contacto durante a simulação. Neste tipo de problemas existem quatro maneiras de se definir as superfícies em elementos rígidos, estruturais e de superfície [61]:

- *Single-sided surfaces* – O utilizador especifica qual é o lado do elemento que entra em contacto;
- *Double-sided surfaces* - Os dois lados de um elemento, e todas as suas arestas livres são tidas em consideração para a análise do contacto;
- *Edge-based surfaces* – Considera-se que apenas existe contacto nas arestas ao longo do perímetro do elemento;
- *Node-based surfaces* – Os nós do elemento é que são considerados para a análise do contacto;

No contexto do programa, como se pretende analisar o contacto entre a base da peça, e a superfície do impactor, e se sabe quais são os lados dos elementos das superfícies, que entram em contacto, pode-se simplificar a definição das superfícies com o *Single-Sided surfaces*. Assim as condições de análise são menos severas, pois o Abaqus® apenas está a ter em consideração um dos lados de cada elemento. Para que o

contacto seja bem detetado pelo programa, a base da peça e a superfície do impactor têm de ter as suas normais devidamente identificadas.

Em termos das propriedades a utilizar no contacto, existem duas componentes que podem ser utilizadas [61]:

- *Normal Contact* – corresponde a todas as componentes das forças normais às superfícies;
- *Tangential Contact* - trata das componentes relativas ao deslizamento entre as superfícies;

Por sua vez o *Normal Contact* possui duas formulações [61]:

- *Hard Contact* – este modelo declara que, quando a distância entre duas superfícies é nula, elas entram em contacto, e qualquer pressão existente é transferida entre elas. Quando a distância entre as duas aumenta, a pressão passa a ser nula (Figura 190). Apesar desta representação ser a mais fidedigna da realidade, a alteração abrupta da tensão de contacto pode levar a problemas de convergência do modelo, e, portanto, existem diferentes métodos para impor esta variação dentro do *Hard Contact* (*Direct*, *Penalty* e *Augmented Lagrange*);

Figura 190 - Evolução da tensão, pelo comportamento "*Hard Contact*"

- *Soft Contact* – para ultrapassar os problemas que podem ser gerados pelo *Hard Contact*, o *Soft Contact* é um modelo mais simples de realizar em termos computacionais, mas tem um nível de precisão menor. Esta simplificação advém do facto de o aumento da pressão de contacto não ser instantâneo, mas sim gradual, com o aumento da penetração entre as duas superfícies (Figura 191). Tal como o *Hard Contact*, *o Soft Contact* possui diferentes metodologias (*Linear*, *Tabular* e *Exponential*);

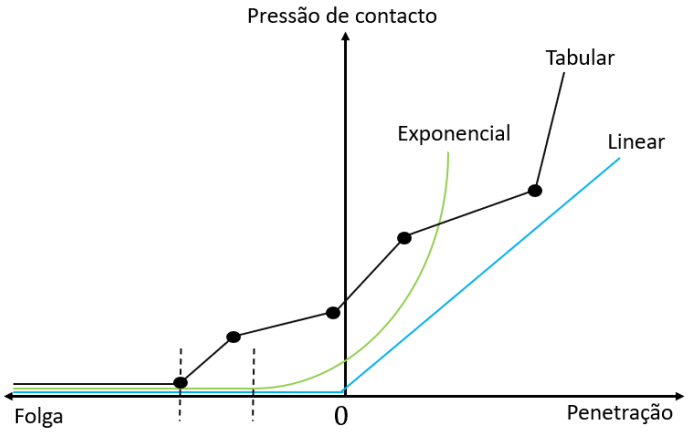


Figura 191 - Evolução dos diferentes tipos de *Soft Contact*

Como o *hard contact* é mais preciso e não causa problemas nos ensaios estáticos da peça que se está a estudar, esta é a formulação que foi empregue no algoritmo do programa. Em termos dos métodos de imposição da variação, entre o *Direct*, o *Penalty* e o *Augmented Lagrange*, optou-se por este último, pois funciona como uma mistura das duas anteriores. Esta característica híbrida, permite ao *Augmented Lagrange* ter a precisão do Direct, mas possuindo um menor risco de problemas de convergência [61].

Quanto ao *Tangential contact*, internamente na Simoldes® o valor utilizado para o coeficiente de atrito entre a superfície do polipropileno e a do aço utilizado nos ensaios reais é de 0.22.

Entre as duas superfícies é ainda necessário mencionar, qual delas é a *master* e qual é a *slave*. Seguindo as indicações do livro [61], existem dois critérios que se devem seguir:

1. A superfície que serve de master deve ser a que possui uma malha mais refinada;
2. Caso as malhas das duas superfícies sejam semelhantes, a *slave* deve ser aquela que possui o material com menor módulo de Young;

Como tanto a peça base como o impactor possuem uma malha de 5mm, a slave foi a peça base, visto que esta é produzida com um material polimérico, enquanto que o impactor é feito de aço.

De forma a impor-se ao modelo todas as condições de contacto mencionadas, desenvolveu-se a função *Interaction_Static_Test* da Figura 192, que procura no modelo, as superfícies da base da peça e do impactor (Linhas 27-28), e com as referências a cada uma, impõem um contacto do tipo *Surface to Surface* entre elas, através do objeto *SurfaceToSurfaceContactStd* (Linhas 30-34) [81]. As propriedades do contacto são introduzidas através do objeto *ContactProperty* (Linhas 9-24) [82].

```
# Create the interaction an interaction properties
# between the different components in the static test
def Interaction_Static_Test():

    # Internal Variable
    a = mymodel.rootAssembly

    # Create interaction Properties: Stamp contact
    mymodel.ContactProperty('IntProp-Stamp')
    mymodel.interactionProperties['IntProp-Stamp'].TangentialBehavior(
        formulation=PENALTY, directionality=ISOTROPIC,
        slipRateDependency=OFF,
        pressureDependency=OFF, temperatureDependency=OFF,
        dependencies=0,
        table=((0.22, ), ), shearStressLimit=None,
         maximumElasticSlip=FRACTION,
        fraction=0.005, elasticSlipStiffness=None)

    mymodel.interactionProperties['IntProp-Stamp'].NormalBehavior(
        pressureOverclosure=HARD, allowSeparation=ON,
        contactStiffness=DEFAULT,
        contactStiffnessScaleFactor=1.0,
        clearanceAtZeroContactPressure=0.0,
        constraintEnforcementMethod=AUGMENTED_LAGRANGE)

    # Create face to face interaction
    region1 = a.instances['Base-1'].surfaces['Base_Surf']
    region2 = a.instances['Stamp-1'].surfaces['Stamp_Surf']

    mymodel.SurfaceToSurfaceContactStd(name='Stamp_Contact',
        createStepName='Initial', master=region2, slave=region1,
        sliding=FINITE, thickness=ON,
        interactionProperty='IntProp-Stamp',adjustMethod=NONE,
        initialClearance=OMIT, datumAxis=None, clearanceRegion=None)
```

Figura 192 -Função de imposição das interações entre componentes (*Interaction_Static_Test*)

Com as propriedades do contacto introduzidas, é necessário impor as condições de ensaio. Como o ensaio se trata de uma simples aplicação de uma força por um impactor, e não existe um enorme movimento relativo entre os diferentes componentes, a simulação a realizar é do tipo estática. Em termos de divisões, o ensaio estático é constituído por três steps:

- *Initial – Step* onde são impostas as condições iniciais do modelo. Aqui são impostos os encastramentos, simetrias e/ou anti-simetrias da peça base, e bloqueiam-se os deslocamentos do impactor, em todas as direções e rotações, com a exceção da translação segundo o eixo dos z's (Figura 193 Linhas 24-29)(Devido à infinidade de possibilidades que a peça base pode ter em termos

das suas condições de fronteira, estas características são impostas no modelo base, que se pretende estudar, e não pelo programa);

- *Step-Static-Test-Displacement* – Como a geometria e a espessura da base da peça podem ser diferentes de modelo para modelo, e de teste para teste, o impactor não pode estar logo em contacto com a base da peça. Este tem de se encontrar a uma determinada distância da base e durante o ensaio deve ser trazido até ela. Para que isto aconteça, este *step* impõem um deslocamento vertical de 0.15mm no impactor, que é o suficiente para que este entre em contacto com a base, e não ocorram problemas de convergência (Figura 193 Linhas 12-15 e 31-33);
- *Step-Static-Test-Force* – Com o impactor na posição correta, o deslocamento vertical da etapa anterior é então desligado, e passa a ser aplicada a carga de 200N que tem de demorar 1 segundo até atingir o seu valor máximo (Figura 193 Linhas 18-21, 36-38 e 41-46);

Para além da imposição das condições de ensaio é necessário impor ao Abaqus® as variáveis que se quer que sejam calculadas, e, portanto, o programa cria um *field output* para todo o modelo, e *um history output* que regista o deslocamento do impactor (Figura 194 Linhas 48-50 e 53-61).

Com as características do ensaio todas impostas, o programa cria um job e, se o utilizador impôs no início do estudo que quer que este seja resolvido, o programa submete o *job* e espera que este seja concluído (Figura 194 Linhas 81-82).

Todo este processo de imposição das condições de ensaio é regido pela função *Static_Test* da Figura 193, Figura 194 e Figura 195 . Para além de impor as condições de ensaio, caso este seja concluído com sucesso, o *Static_test* tem ainda uma série de funções que tratam de monitorizar o job e determinar se este foi bem sucedido, notificar o utilizador através de alertas via email, extrair os resultados para serem analisados e plotados, criar animações dos ensaios, guardar os resultados numa base de dados, e caso algum problema ocorra com a simulação, notificar o utilizar de tal problema, e prosseguir para a geometria/ensaio seguinte (Linhas 72-130).

```
# Create the static test parameters
def Static_test(solve_model, job_name, my_path_1, my_path_2,
    final_report, report_counter, hex_size, hex_thickness,
    hex_height, yield_stress, limit_strain):

    # Internal Variables
    a = mymodel.rootAssembly
    da = a.datums
    Stamp_Load_Point_CSYS_id = a.features['Stamp_Load_Point_CSYS'].id

    # Create Displacement Step
    mymodel.StaticStep(name='Step-Static-Test-Displacement',
        previous='Initial',
        timePeriod=1.0, initialInc=0.1, minInc=2e-05, maxInc=1.0,
        nlgeom=ON)

    # Create Load Step
    mymodel.StaticStep(name='Step-Static-Test-Force',
        previous='Step-Static-Test-Displacement',
        timePeriod=1.0, initialInc=0.1, minInc=2e-05, maxInc=1.0,
        nlgeom=ON)

    # Create Displacement
    region = a.sets['Stamp-1.Stamp_RP']
    mymodel.DisplacementBC(name='BC-Stamp_Displacement',
        createStepName='Initial', region=region, u1=SET, u2=SET,
        u3=UNSET, ur1=SET, ur2=SET, ur3=SET, amplitude=UNSET,
        distributionType=UNIFORM, fieldName='',
        localCsys=da[Stamp_Load_Point_CSYS_id])

    mymodel.boundaryConditions['BC-Stamp_Displacement']. \
        setValuesInStep(stepName='Step-Static-Test-Displacement',
        u3=0.15)

    # Static load amplitude
    myamplitude = mymodel.SmoothStepAmplitude(
        name='Static_Load_Amplitude', timeSpan=STEP,
        data=((0.0, 0.0), (1.0, 1.0)))

    # Create Load
    region = a.sets['Stamp-1.Stamp_RP']
    mymodel.ConcentratedForce(name='Load-Static-Test-Force',
        createStepName='Step-Static-Test-Force', region= region,
        cf3=100.0, distributionType=UNIFORM, field='',
        amplitude=myamplitude.name,
        localCsys=da[Stamp_Load_Point_CSYS_id])
```

Figura 193 - Função do ensaio estático (*Static_Test*)(1/3)

```
47      # Create Field Output
48      mymodel.FieldOutputRequest(name='F-Output-Static_Test',
49          createStepName='Step-Static-Test-Force', variables=(
50              'MISES', 'U', 'UR','PE'),frequency=100)
51
52      # Create History Output
53      mymodel.historyOutputRequests['H-Output-1'].setValues(
54          variables=('ALLFD', 'ALLKE', 'ALLSE'))
55
56      region = a.allInstances['Stamp-1'].sets['Stamp_RP']
57      mymodel.HistoryOutputRequest(
58          name='H-Output-Displacement_Study_Point',
59          createStepName='Step-Static-Test-Force',
60          variables=('U1', 'U2', 'U3', 'UR1', 'UR2', 'UR3'),
61          region=region, sectionPoints=DEFAULT, rebar=EXCLUDE)
62
63      # Create Job
64      myjob = mdb.Job(name= job_name, model='Model-1', description='',
65          type=ANALYSIS, atTime=None, waitMinutes=0, waitHours=0,
66          queue=None, memory=90, memoryUnits=PERCENTAGE,
67          getMemoryFromAnalysis=True, explicitPrecision=SINGLE,
68          nodalOutputPrecision=SINGLE, echoPrint=OFF,
69          modelPrint=OFF, contactPrint=OFF, historyPrint=OFF,
70          userSubroutine='', scratch='', resultsFormat=ODB)
71
72      # Submit the job, if the user has requested for it
73      if solve_model:
74
75          # Monitor the job
76          monitorManager.addMessageCallback(jobName=job_name,
77              messageType=ANY_MESSAGE_TYPE, callback=jobMonitorCallback,
78              userData=None)
79
80          # Submit the job and wait for it's completion
81          myjob.submit()
82          myjob.waitForCompletion()
83
84          # If the job was completed with success, extract the results
85          if jobMonitorCallback.job_completed == True:
86              # Create an animation of the test
87              create_animations(job_name, path = my_path_1)
88
89              # Exclude the fixtures values
90              exclusion_element_list = []
91              exclusion_element_list = Exclusion_Area(job_name)
92
93              # Take a screenshot of the last frame of the test
94              screenshot(job_name, step_name='Step-Static-Test-Force',
95                  path = my_path_1)
```

Figura 194 - Função do ensaio estático (*Static_Test*)(2/3)

```
            # Export stress results
            max_stress = Export_Stress_Results(job_name,
                exclusion_element_list,
                step_name= 'Step-Static-Test-Force',study_frame=-1
                mat_yield_stress = 21, my_path = my_path_1)

            # Export the plastic strain results
             max_plastic_strain=Export_Plastic_Strain_Results(
                job_name,
                exclusion_element_list,
                step_name='Step-Static-Test-Force', study_frame=-1
                mat_limit_strain=limit_strain, my_path=my_path_1)

            # Export the displacement results
            max_displacement = Export_nodal_displacement(job_name,
                my_path = my_path_1)

            # Get the mass of the component
            p = mymodel.parts['Base']
            myview.setValues(displayedObject=p)
            part_mass = p.getMassProperties()['mass']

            # Update report card
            final_report[report_counter] = {'size':hex_size,
                'height':hex_height , 'thickness':hex_thickness,
                'test':'Estatico', 'stress':max_stress,
                'displacement':max_displacement,
                'plastic_strain':max_plastic_strain,
                'weight':part_mass,
                'path':my_path_2}

            report_counter+=1

    return final_report, report_counter
```

Figura 195 - Função do ensaio estático (*Static_Test*)(3/3)

3.5.3.10 Análise do job e notificações via email

Durante um estudo que envolva várias geometrias e simulações, o programa desenvolvido tem de ser capaz de lidar com os erros que podem aparecer durante os diversos cálculos, e perante eles, ajustar o seu processo de análise, para que um erro numa dada geometria ou ensaio não impeça a resolução das restantes. Sendo assim, dentro do programa foi desenvolvida uma função, que, quando um *job* for submetido e o Abaqus® realiza a análise, a função, em paralelo, monitoriza as mensagens dentro do *job monitor*. A referente função que trata deste processo de monitorização é a *JobMonitorCallback* da Figura 198.

O JobMonitorCallback utiliza um método do objeto *MonitorMgr*, o *addMessageCallback*, que permite ao script monitorizar as mensagens que vão aparecendo durante a análise [58]. Existem uma série de mensagens que o Abaqus® gera durante este processo, mas para este caso, as que interessam são [72]:

- "*ABORTED*" – Indica que o job que se estava a realizar abortou;
- "*ERROR*" – Indica que a simulação encontrou um erro e não conseguiu continuar;
- "*JOB_COMPLETED*" – Indica que a simulação foi completa com sucesso;

É através destas mensagens, que o *script* sabe se a simulação falhou, ou conseguiu chegar ao fim sem encontrar nenhum erro. O addMessageCallback necessita então de receber 4 argumentos [72].

1. *jobName* – O nome do *job* que deve ser analisado;
2. *messageType* – O tipo de mensagem que leva o *addMessageCallback* a chamar a função associada. O messageType também é enviado como argumento, para a função do *callBack*;
3. *callBack* – É a função que deve de ser chamada sempre que o addMessageCallback encontra o tipo de mensagem mencionada no *messageType*;
4. *userData* – Corresponde aos dados que se quer enviar para dentro da função do callBack, para além do messageType;

Com tudo isto, quando a função *Static_Test* chega ao ponto de submeter o *job*, para ser calculado, a função ativa o *addMessageCallback*, que continuamente lê todas as mensagens do respetivo job. Quando uma mensagem é detetada a função jobMonitorCallback é chamada, e esta analisa a mensagem para saber se a simulação está a ser realizada corretamente, ou se ocorreu algum problema. Caso tenha ocorrido algum problema, o programa passa para a geometria seguinte (Figura 196 Linhas 77-79). Para além disto, o programa recorre ao método *waitForCompletion*, para fazer com que o script pare, até ao momento em que a simulação é dada como terminada [83] (Figura 196 Linha 83).

```
72      # The program only submits the job, if the user as given
73      # that information at the start
74      if solve_model:
75
76          # Monitor the job
77          monitorManager.addMessageCallback(jobName=job_name,
78              messageType=ANY_MESSAGE_TYPE, callback=jobMonitorCallback,
79              userData=None)
80
81          # Submit the job and wait for it's completion
82          myjob.submit()
83          myjob.waitForCompletion()
```

Figura 196 – Análise do job (*addMessageCallback*)

Com tudo isto, quando o programa submete um job para ser calculado, este espera que o cálculo seja dado como terminado para prosseguir com os restantes processos de análise, mas, em simultâneo está constantemente a monitorizar a simulação, para ver se esta está a correr como previsto. No final do processo, o programa notifica o utilizador, através de um email, com as informações da simulação que acabou de ser realizada (Figura 197).

No processo de análise do job, com a deteção da mensagem pelo *addMessageCallback* a função *jobMonitorCallback*, é chamada (Figura 198), e verifica se a mensagem recebida é igual a “ABORTED”, “ERROR” ou “JOB_COMPLETED” (Figura 198 Linhas 8 e 29). Caso nenhuma delas seja detetada o programa continua à espera por uma nova mensagem, contudo, se a mensagem for equivalente, a função envia uma *tag* ao programa principal que o notifica se a simulação foi concluída com sucesso ou não (Figura 198 Linhas 27 e 48), e desliga o processo iterativo iniciado pelo *addMessageCallback* (Figura 198 Linhas 22-24 e 43-45). Para além disto, consoante a mensagem obtida, o *jobMonitorCallback* cria o título e o corpo da mensagem que deve ser enviada para notificar o utilizador, e manda esta informação para a função responsável pelo envio de email do sistema (Figura 198 Linhas 11-19 e 36-45).

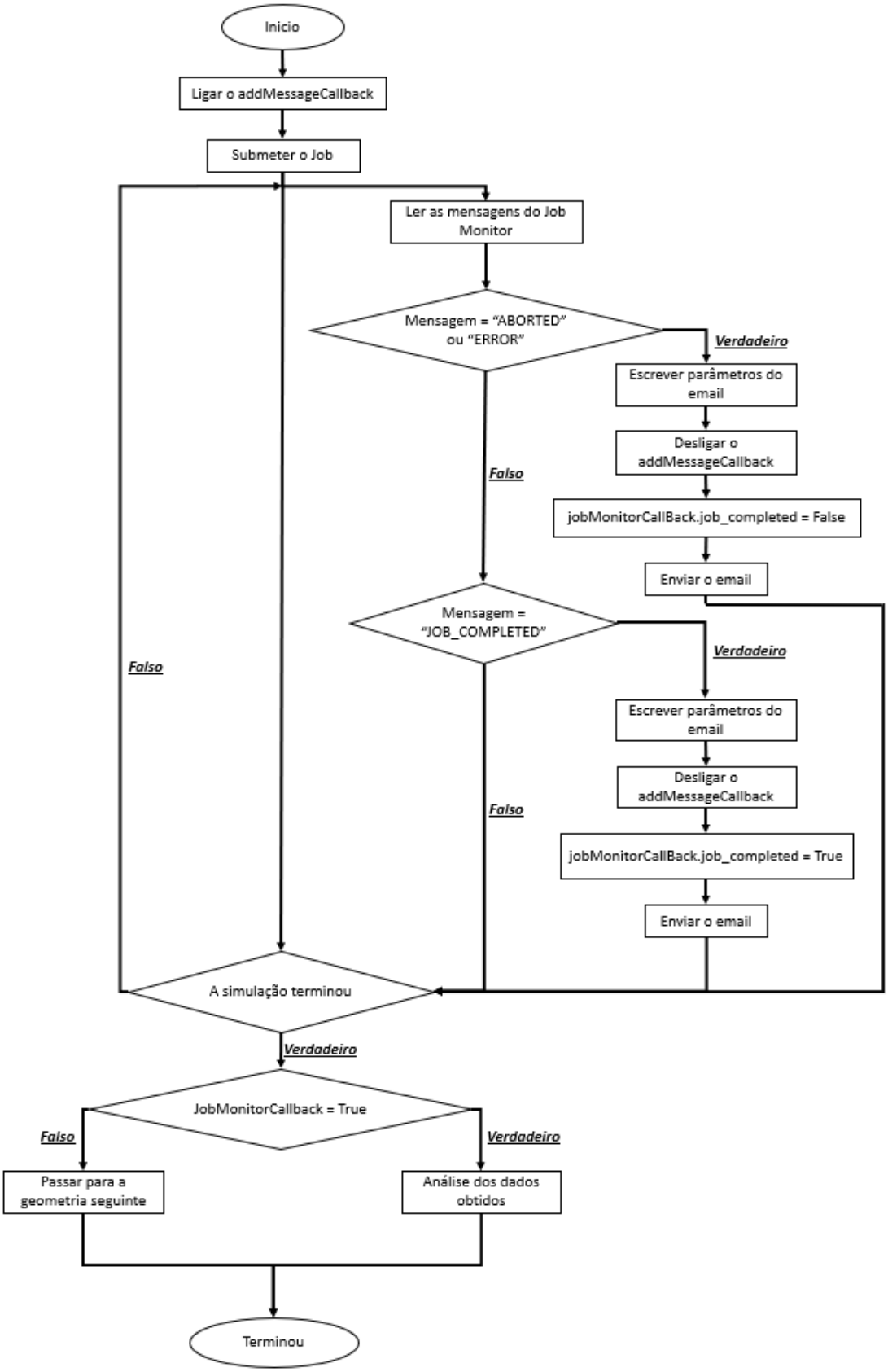


Figura 197 – Fluxograma para a análise do job e notificação via email

```
# Define a callback function jobMonitorCallback()
# That continuously checks the job monitor and if
# The job has concluded successfully
def jobMonitorCallback (job_name, messageType, data, userData):

    # Monitor message that tells the prgram that
    # the simulation has failed
    if((messageType==ABORTED) or (messageType==ERROR)):

        # Email message
        email_message = "Mas noticias - a simulacao: \n " \
            + str(job_name) + "  FALHOU."

        # Email subject message
        subject_message = "[FAIL] " + str(job_name)

        # Send email
        sendEmailMessage(subject_message = subject_message ,
            email_message = email_message)

        # Stop monitoring the job
        monitorManager.removeMessageCallback(jobName = job_name,
            messageType = ANY_MESSAGE_TYPE, callback=jobMonitorCallback,
            userData = None)

        # Tag that tells the main function that the job has failed
        jobMonitorCallback.job_completed = False

    elif (messageType == JOB_COMPLETED):

        # Email message
        email_message = "Boas noticias - a simulacao: \n " \
            + str(job_name) + " CORREU SEM PROBLEMAS "

        # Email subject message
        subject_message = "[GOOD] " + str(job_name)

        # Send email
        sendEmailMessage(subject_message = subject_message ,
            email_message = email_message)

        # Stop monitoring the job
        monitorManager.removeMessageCallback(jobName = job_name,
            messageType = ANY_MESSAGE_TYPE, callback = jobMonitorCallback,
            userData=None)

        # Tag that tells the main function that the job was successfull
        jobMonitorCallback.job_completed = True
```

Figura 198 – Função de análise do job (*JobMonitorCallback*)

Com a informação proveniente do *jobMonitorCallback*, a função *sendEmailMessage* da Figura 200, recorre ao protocolo SMTP (*Simple Mail Transfer Protocol*) e às livrarias *smtplib* e *MIMEText* do Python, para conseguir enviar um email, através da internet. [84] (Figura 199).

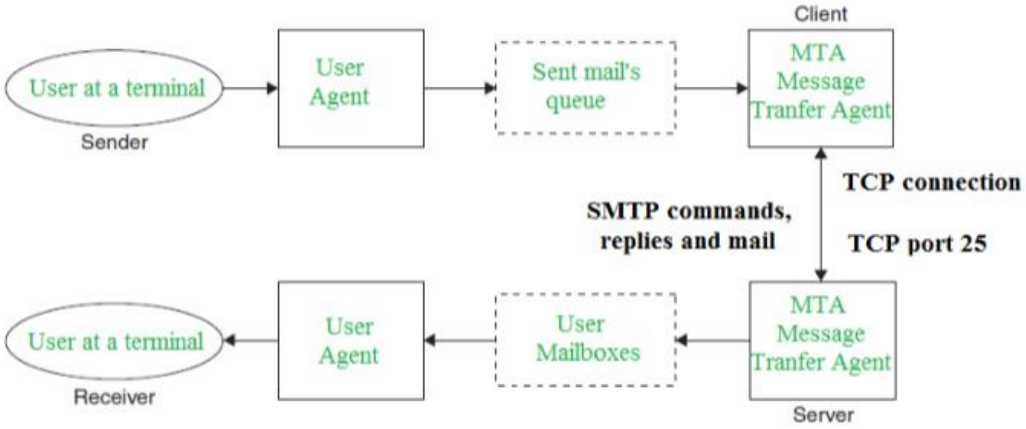


Figura 199 - Representação esquemática do SMTP

A livraria *smtplib* permite ao script aceder às funcionalidades do SMTP podendo assim ligar o script ao respetivo servidor e enviar a mensagem, que é convertida para a forma adequada através do *MIMEText* (Figura 200 Linhas 4-15)[58]. Como a conta de email utilizada para notificar o utilizador se trata de uma conta gmail, o servidor que deve ser utilizado deve ser o da google (smtp.gmail.com), que tem de ser acedido segundo a porta 587 (Figura 200 Linhas 18 - 21). Com estas credênciais o programa cria a conexão ao servidor através do objeto SMTP (Figura 200 Linha 31), encripta a mensagem que necessita de enviar(Figura 200 Linhas 34 a 38), faz o login da conta gmail (Figura 200 Linha 44), e envia a mensagem para si mesma (Figura 200 Linha 47). Quando o processo se der por terminado, então a ligação é cancelada (Figura 200 Linha 48).

```
# Define a function sendEmailMessage() to send the email
def sendEmailMessage(subject_message, email_message):

    sender = 'python.script.alarm.thesis@gmail.com'
    recipient = 'python.script.alarm.thesis@gmail.com'
    subject = subject_message
    contents = email_message

    # Create a text/plain message
    msg = MIMEText(contents)

    # Specify the email subject, sender and recipient
    msg['Subject'] = subject
    msg['From'] = sender
    msg ['To'] = recipient

    # This is Googles Outgoing Mail (SMTP) server
    gmail_smtp_server = 'smtp.gmail.com'

    # This is the port used by Gmail server for outgoing Mail
    gmail_smtp_port = 587

    # Gmail username (SMTP username)
    gmail_username = 'python.script.alarm.thesis@gmail.com'

    # Gmail password
    gmail_password = '********'

    # Create an SMTP object.
    # The SMTP connect() method is called using the name and port
    session = smtplib.SMTP(gmail_smtp_server, gmail_smtp_port)

    # Identify ourselves to the ESMTP server using EHLO
    session.ehlo()

    #Put the SMTP connection in Transport Layer Security (TLS) mode using
    #SMTP.starttls() so all SMTP commands that follow will be encrypted
    session.starttls()

    # Call EHLO again
    session.ehlo()

    # Login to the server using SMTP.login()
    session.login(gmail_username, gmail_password)

    # Send the email using SMTP.send,mail()and end the session
    session.sendmail(sender,[recipient], msg.as_string())
    session.close()
```

Figura 200 – Envio de email (*sendEmailMessage*)

3.5.3.11 Pós processamento e extração de dados

Quando a simulação é dada como terminada, e se a função responsável pela monitorização da simulação chegar à conclusão de que esta foi concluída com sucesso, o script passa então para a parte do pós-processamento e extração de dados do ficheiro de resultados. Esta parte está inserida dentro da função *Static_test* do capítulo 3.5.3.9, logo após a submissão e monitorização do job, e só é acedida se o *JobMonitorCallback* determinou que a simulação foi executada com sucesso (Figura 201 Linha 3). Nesta parte o programa executa 7 etapas:

1. Cria um vídeo da simulação, para cada uma das variáveis a estudar, e guarda-as na respetiva pasta do estudo (Figura 201 Linha 6);
2. Tira uma foto do último *frame* da simulação, para cada uma das variáveis a estudar, e guarda-as nas respetivas pastas (Figura 201 Linhas 9-11);
3. Exporta todos os valores da tensão de Von Mises, de todos os elementos da malha da peça base, e guarda os dados num ficheiro de texto que serve de base de dados (Figura 201 Linhas 14-16);
4. Exporta todos os valores de deformação plástica, de todos os elementos da malha da peça base, e guarda os dados num ficheiro de texto (Figura 201 Linhas 19-21);
5. Exporta todos os valores do deslocamento do impactor durante a simulação, e guarda-os num ficheiro de texto (Figura 201 Linhas 24 e 25);
6. Obtém a massa total da peça, com a geometria de frisos que está a ser analisada (Figura 201 Linhas 28-31);
7. Guarda os dados mais importantes num *Hash Map*, para mais tarde fazer uma tabela resumo dos resultados de todos os ensaios feitos pelo programa (Figura 201 Linhas 33-39);

Cada uma das funcionalidades mencionadas possui uma função com o seu próprio algoritmo de análise, que permite ao programa principal executá-las em qualquer altura do seu funcionamento.

```
        # If the job was completed with success, then
        # the program will export all of the results
        if jobMonitorCallback.job_completed == True:

            # Create an animation of the test
            create_animations(job_name, path = my_path_1)

            # Take a screenshot of the last frame of the test
            screenshot(job_name,
                step_name='Step-Static-Test-Force',
                path = my_path_1)

            # Export stress results
            max_stress = Export_Stress_Results(job_name,
                step_name= 'Step-Static-Test-Force',
                mat_yield_stress = yield_stress, my_path = my_path_1)

            # Export plastic strain
            max_plastic_strain = Export_Plastic_Strain_Results(job_name,
                step_name='Step-Static-Test-Force',
                mat_limit_strain=0.001, my_path=my_path_1)

            # Export displacement results
            max_displacement = Export_nodal_displacement(job_name,
                my_path = my_path_1)

            # Get the mass of the component
            p = mymodel.parts['Base']
            myview.setValues(displayedObject=p)
            part_mass = p.getMassProperties()['mass']

            # Update report card
            final_report[report_counter] = {'size':hex_size,
                'height':hex_height , 'thickness':hex_thickness,
                'test':'Estatico', 'stress':max_stress,
                'displacement':max_displacement,
                'plastic_strain':max_plastic_strain,
                'weight':part_mass,
                'path':my_path_2}

            report_counter+=1

    return final_report, report_counter
```

Figura 201 – Extração de dados (*Static-test*)

3.5.3.11.1 Screenshot e vídeo da simulação

Como o programa foi desenhado para realizar um elevado número de simulações em sequência, para não se correr o risco de se ocupar a memória toda do computador que vai realizar a análise, quando a simulação é concluída e todos os dados necessários para a análise são extraídos, os ficheiros gerados pelo Abaqus® são apagados. Contudo, de forma a permitir ao utilizador visualizar a simulação, o programa gera uma *screenshot* e um vídeo dos resultados do Abaqus®, para cada uma das variáveis que se pretende estudar. Assim, após o estudo da peça, o utilizador pode ver os resultados obtidos e tirar conclusões adicionais ao estudo (Figura 202).

A parte dos *screenshots* é gerada pela função *screenshot* da Figura 203, e esta, quando é chamada pelo programa, abre o ficheiro de resultados que lhe é enviado como argumento, e vai buscar as coordenadas da *viewport* que estão guardadas no perfil "User-1" (Figura 203 Linha 16)[85]. Com esta *viewport* selecionada, a função impõe também condições para melhorar a apresentação da peça (Figura 203 Linhas 7 - 15) e demonstra a respetiva variável que é necessário tirar a *screenshot* (Figura 203 Linha 19-21, 30-32 e 41-43). Com a *viewport* devidamente apresentada, através do método *printToFile* do objeto *session*, o programa grava uma imagem da *viewport* corrente, em formato *PNG (Portable Graphics Format)*, na respetiva diretoria que lhe é introduzida como argumento (Figura 203 Linha 26-27, 37-38 e 48-49) [86].

A função *create_animations* funciona de forma semelhante à *screenshots*, aqui a função abre a odb que recebe como argumento e impõe-lhe as condições necessárias para que a *viewport* fique num formato apresentável (Figura 204 Linhas 10-16, 23 -29 e 36-42), e depois, usando o método *writeImageAnimations* do objeto *session*, cria uma animação em formato *QuickTime* (Figura 204 Linhas 19-20, 32-33 45-46).

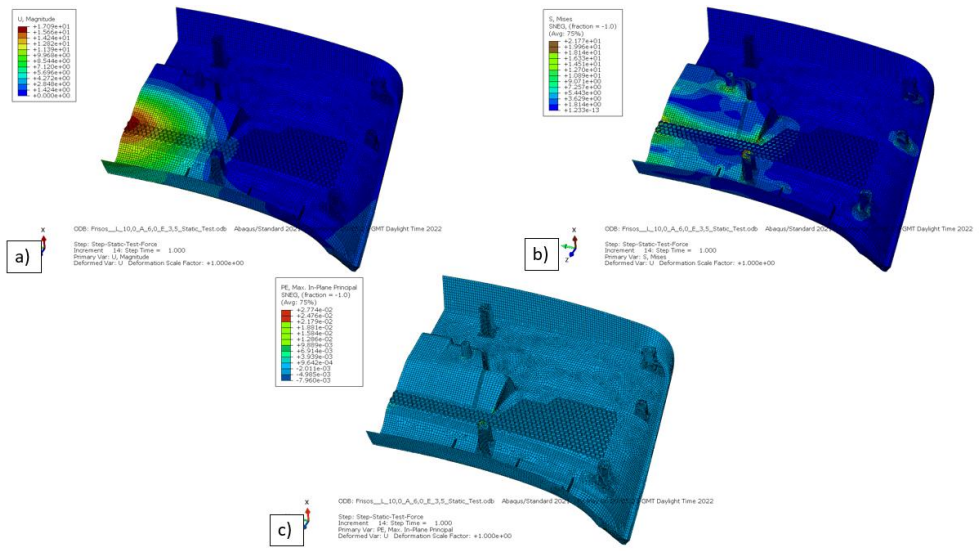


Figura 202 – Exemplos de screenshots geradas a) Deformação b) Tensão c) Deformação Plástica

```
# Create a screenshot of the analysis
def screenshot(odbname,step_name, path):

    # Open database and set view for the screenshot
    odb = session.openOdb(name=odbname + '.odb')
    myview =session.viewports[session.currentViewportName]
    myview.setValues(displayedObject=odb)
    myview.odbDisplay.commonOptions.setValues(renderStyle=SHADED,)
    myview.odbDisplay.display.setValues(plotState=(
    CONTOURS_ON_DEF, ))
    myview.enableMultipleColors()
    myview.setColor(initialColor='#BDBDBD')
    cmap=myview.colorMappings['Default']
    myview.setColor(colorMapping=cmap)
    myview.disableMultipleColors()
    myview.view.setValues(session.views['User-1'])

    # Take the screenshot - Displacement
    myview.odbDisplay.setPrimaryVariable(
        variableLabel='U', outputPosition=NODAL, refinement=(INVARIANT,
        'Magnitude'), )

    file_name = path + '\\' + odbname + '_step_'+ \
        str(step_name) + '_Displacement'

    session.printToFile(fileName=file_name, format=PNG, canvasObjects=(
        myview, ))

    # Take the screenshot - Stress Mises
    myview.odbDisplay.setPrimaryVariable(
        variableLabel='S', outputPosition=INTEGRATION_POINT, refinement=(
        INVARIANT, 'Mises'), )

    file_name = path + '\\' + odbname + '_step_'+ \
        str(step_name) + '_Mises'

    session.printToFile(fileName=file_name, format=PNG, canvasObjects=(
        myview, ))

    # Take the screenshot - Plastic Strain
    myview.odbDisplay.setPrimaryVariable(
        variableLabel='PE', outputPosition=INTEGRATION_POINT, refinement=(
        INVARIANT, 'maxInPlanePrincipal'), )

    file_name = path + '\\' + odbname + '_step_'+ \
        str(step_name) + '_PLStrain'

    session.printToFile(fileName=file_name, format=PNG, canvasObjects=(
        myview, ))
```

Figura 203 – Tirar uma screenshot da simulação (*screenshot*)

```python
# Create Animations of the simulation
def create_animations(odbname, path):
    # Open the correspondant MDB and set common viewport properties
    name =  odbname + '.odb'
    odb_object = session.openOdb(name=name)
    myview.setValues(displayedObject = odb_object)
    myview.odbDisplay.display.setValues(plotState = (CONTOURS_ON_DEF, ))

    # Set specific viewport properties - U
    myview.odbDisplay.setPrimaryVariable(
        variableLabel='U', outputPosition=NODAL,
        refinement=(INVARIANT, 'Magnitude'), )
    myview.animationController.setValues(animationType = TIME_HISTORY)
    myview.animationController.play(duration = UNLIMITED)
    session.imageAnimationOptions.setValues(vpDecorations = OFF,
        vpBackground = ON, compass = OFF, timeScale = 1)
    # Create a video of the current viewport - U
    file_name = path + '\\' + odbname + 'U'
    session.writeImageAnimation(fileName = file_name,
        format = QUICKTIME , canvasObjects = (myview, ))

    # Set specific viewport properties - S_Mises
    myview.odbDisplay.setPrimaryVariable(
        variableLabel='S', outputPosition=INTEGRATION_POINT, refinement=(
        INVARIANT, 'Mises'), )
    myview.animationController.setValues(animationType = TIME_HISTORY)
    myview.animationController.play(duration = UNLIMITED)
    session.imageAnimationOptions.setValues(vpDecorations = OFF,
        vpBackground = ON, compass = OFF, timeScale = 1)
    # Create a video of the current viewport - S_Mises
    file_name = path + '\\' + odbname + '_S_Mises'
    session.writeImageAnimation(fileName = file_name,
        format = QUICKTIME , canvasObjects = (myview, ))

    # Set specific viewport properties - PL_Strain
    myview.odbDisplay.setPrimaryVariable(
        variableLabel='PE', outputPosition=INTEGRATION_POINT, refinement=(
        INVARIANT, 'maxInPlanePrincipal'), )
    myview.animationController.setValues(animationType = TIME_HISTORY)
    myview.animationController.play(duration = UNLIMITED)
    session.imageAnimationOptions.setValues(vpDecorations = OFF,
        vpBackground = ON, compass = OFF, timeScale = 1)
    # Create a video of the current viewport - PL_Strain
    file_name = path + '\\' + odbname + '_PL_Strain'
    session.writeImageAnimation(fileName = file_name,
        format = QUICKTIME , canvasObjects = (myview, ))

    # Stop the animation
    myview.animationController.setValues(animationType=NONE)
```

Figura 204 – Criar uma animação da simulação (*create_animations*)

3.5.3.11.2 Extração dos valores de tensão

No que toca aos valores de tensão, o processo é regido pela função *Export_Stress_Results*, da Figura 206. Como argumentos, a função recebe:

- *odb_name* – o nome da ODB onde se encontram os resultados da simulação;
- *step_name* – nome do step donde se quer retirar os resultados;
- *mat_yield_stress* – tensão de cedência do material;
- *my_path* – diretoria donde devem de ser guardados os resultados;
- *exclusion_element_list* – lista com as *labels* dos elementos que não devem de ser analisados;

Com esta informação, a função abre o ficheiro de resultados e itera por todos os elementos da base da peça, retirando o valor da tensão sob a forma de *Von Mises*, e guardando-as numa lista (Figura 206 Linhas 20-23). Este valor é depois comparado com o valor da tensão de cedência do material tendo um fator de segurança associado, e se este ultrapassar o limite, incrementa-se o valor do número de nós que se encontra fora do valor máximo (Figura 206 Linhas 24 e 25). Esta comparação também é feita sem se recorrer ao fator de segurança (Figura 206 Linhas 26 e 27). Para além disto a função verifica se o valor que se está a analisar na respetiva iteração do *loop* é o maior valor encontrado até ao momento, se for, esse valor é então guardado em memória (Figura 206 Linhas 28 e 29). No final deste processo iterativo, o programa fica com uma lista com todos os valores de tensão da peça base, possui o valor máximo de tensão guardado em memória e tem calculado o número de nós que se encontra fora da especificação.

Com todos os dados obtidos, o programa guarda-os em listas separadas (Figura 206 Linhas 34-38), e converte essas mesmas listas sobe a forma de colunas, num ficheiro de texto que é guardado na respetiva diretoria e é mais tarde utilizado como base de dados (Figura 206 Linhas 41-48). Os dados guardados nesse mesmo ficheiro estão organizados segundo a estrutura representada na Figura 205.

Linha	Coluna				
	stress_values_list[]	num_of_elem_outspec_list[]	num_of_elem_outspec_with_sec_list[]	safety_factor_list[]	my_path_list[]
1	0	10	20	0.8	(...)
2	5				
3	10				
4	15				
5	20				
6	25				

Figura 205 - Organização do ficheiro de texto dos valores de tensão

Com todo o processo realizado, a função devolve o valor da tensão máxima ao programa que a chamou (Figura 206 Linhas 49).

```
# Export all the stress values in a given odb
def Export_Stress_Results(odb_name, exclusion_element_list,
    step_name, study_frame, mat_yield_stress, my_path):

    # Open the odb
    odb = session.openOdb(name=odb_name + '.odb')
    myview =session.viewports[session.currentViewportName]
    myview.setValues(displayedObject=odb)

    # Specifying the step and variable
    fieldVar = odb.steps[step_name].frames[study_frame].fieldOutputs['S']

    # Starting variable values and Read stress values for every node
    max_value = 0
    number_of_elem_Outspec = 0
    number_of_elem_Outspec_with_sec = 0
    safety_factor = 0.8
    stress_values_list = []

    for value in fieldVar.values:
        if not(value.elementLabel in exclusion_element_list):
            element_stress = value.mises
            stress_values_list.append(element_stress)
            if element_stress > (safety_factor * mat_yield_stress):
                number_of_elem_Outspec_with_sec += 1
                if element_stress > mat_yield_stress:
                    number_of_elem_Outspec += 1
            if element_stress > max_value:
                max_value = element_stress

    # Text file directory and Initiate lists
    title = 'Valores de tensao'
    txt_path = my_path + '\\' + title + '.txt'
    mat_yield_stress_list = [mat_yield_stress]
    number_of_elem_Outspec_list = [number_of_elem_Outspec]
    number_of_elem_Outspec_with_sec_list=[number_of_elem_Outspec_with_sec]
    safety_factor_list = [safety_factor]
    my_path_list = [my_path]

    # Save data into a text file
    with open(txt_path, 'w') as f:
        writer = csv.writer(f, delimiter='\t')
        for x in it.izip_longest(
            stress_values_list,mat_yield_stress_list,
            number_of_elem_Outspec_list,
            number_of_elem_Outspec_with_sec_list,
            safety_factor_list, my_path_list, fillvalue=''):
            writer.writerow(x)
      return max_value
```

Figura 206– Exportação da tensão (*Export_Stress_Results*)

3.5.3.11.3 Extração dos valores de deformação plástica

A extração da deformação plástica é tratada pela função *Export_Plastic_Strain_Results* da Figura 208, e esta funciona de forma semelhante ao *Export_Stress_Results* do capítulo 3.5.3.11.2.

Tal como na tensão, a função recebe o nome do ficheiro de resultados que é necessário analisar, o step, a diretoria onde se deve guardar os dados e o valor limite de deformação plástica imposta (Figura 208 Linhas 2 e 4). Com estes valores a função itera por todos os nós da peça base e guarda os valores de cada um numa lista (Figura 208 Linhas 20 a 25). Para além disto analisa também se o valor ultrapassou o limite, e se é o maior valor do modelo (Figura 208 Linhas 26 e 27).

Com todos os valores obtidos, estes são então convertidos em listas, que por sua vez são armazenadas em ficheiros de texto sobre a forma de colunas, resultando na estrutura representada na Figura 207 (Figura 208 Linhas 34-44).

Linha	Coluna			
	strain_values_list[]	mat_limit_strain[]	num_of_elem_outspec_list[]	my_path_list[]
1	0	0.0248	20	(...)
2	0.005			
3	0.010			
4	0.015			
5	0.020			
6	0.025			

Figura 207 - Organização do ficheiro de texto dos valores de deformação plástica

Com tudo concluído o valor máximo de deformação plástica do modelo é devolvido ao programa que chamou a função (Figura 208 Linhas 46).

```python
# Export all the plastic strain values in a given odb
def Export_Plastic_Strain_Results(odb_name,
    exclusion_element_list, step_name, study_frame,
    mat_limit_strain, my_path):

    # Open the odb
    odb = session.openOdb(name=odb_name + '.odb')
    myview =session.viewports[session.currentViewportName]
    myview.setValues(displayedObject=odb)

    # Specifying the step and variable
    fieldVar = odb.steps[step_name].frames[study_frame].fieldOutputs['PE']

    # Starting variable values
    max_value = 0
    number_of_elem_Outspec = 0
    strain_values_list = []

    # Read plastic strain values for every node
    for value in fieldVar.values:
        if not(value.elementLabel in exclusion_element_list):
            element_strain = value.maxInPlanePrincipal
            strain_values_list.append(element_strain)
            if element_strain > mat_limit_strain:
                number_of_elem_Outspec += 1
            if element_strain > max_value:
                max_value = element_strain

    # Text file directory
    title = 'Valores de plasticidade'
    txt_path = my_path + '\\' + title + '.txt'

    # Initiate lists
    mat_limit_strain_list = [mat_limit_strain]
    number_of_elem_Outspec_list = [number_of_elem_Outspec]
    my_path_list = [my_path]

    # Save data into a text file
    with open(txt_path, 'w') as f:
        writer = csv.writer(f, delimiter='\t')
        for x in it.izip_longest(strain_values_list,mat_limit_strain_list,
            number_of_elem_Outspec_list,
            my_path_list, fillvalue=''):
            writer.writerow(x)

    return max_value
```

Figura 208 – Extração da deformação plástica (*Plastic_Strain_Stress_Results*)

3.5.3.11.4 Extração dos valores do deslocamento do impactor

A extração de valores funciona de forma diferente das restantes funções, aqui a função necessita apenas da odb e da diretoria onde deve guardar os ficheiros, e com isto vai buscar os dados do Node Set "*STAMP-1.STAMP_RP*", relativos ao deslocamento (Figura 210 Linhas 8 e 9). Como os dados do *xyDatalistFromField* são extraídos todos agrupados e em formato de *tuple*, a estrutura resultante tem de ser dividida nas suas 5 componentes (tempo, U1, U2, U3 e U) e cada uma delas tem de ser convertida para uma lista (Figura 210 Linhas 12-18). Com todos parâmetros convertidos, estes são então guardados num ficheiro de texto, na correspondente diretoria, com a estrutura da Figura 209 (Figura 210 Linhas 28-32).

Linha	Coluna					
	Tempo	Deformação Magnitude (mm)	Deformação na direção do eixo dos x (mm)	Deformação na direção do eixo dos y (mm)	Deformação na direção do eixo dos z (mm)	path
1	0	0	0	0	0	(…)
2	0.1	0.015	0.015	0	0	
3	0.2	0.030	0.029	0	0.006	
4	0.35	0.053	0.052	0	0.01	
5	0.57	0.086	0.084	0	0.018	
6	0.91	0.137	0.134	-0.0003	0.027	

Figura 209 - Organização do ficheiro de texto dos valores de deslocamento do impactor

No fim da análise a função devolve o valor máximo da deformação encontrada no modelo (Figura 210 Linha 34).

```python
# Export the nodal displacement from the impactor
def Export_nodal_displacement(odb_name, my_path):

    # Open the odb
    odb = session.openOdb(name=odb_name + '.odb')

    # Extract displacement from the Load Point in different directions
    datalist = session.xyDataListFromField(odb=odb, outputPosition=NODAL,
        variable=(('U', NODAL,),), nodeSets=("STAMP-1.STAMP_RP", ))

    # Split the tuple in different directions
    data_um, data_ux, data_uy, data_uz = datalist

    # Convert the tuple to lists
    t, um = zip(*data_um.data)
    t, ux = zip(*data_ux.data)
    t, uy = zip(*data_uy.data)
    t, uz = zip(*data_uz.data)

    # Text file directory
    txt_path = my_path + '\\' \
        + 'Deformacao no ponto de aplicacao da carga.txt'

    # Initiate lists
    path_list = [my_path]

    # Save the data into a text file
    with open(txt_path, 'w') as f:
        writer = csv.writer(f, delimiter='\t')
        for x in it.izip_longest(t,
            um, ux, uy, uz, path_list, fillvalue=''):
            writer.writerow(x)

    return max(um)
```

Figura 210 – Extração do deslocamento do impactor (*Export_nodal_displacement*)

3.5.3.12 Tratamento dos dados do ensaio estático

Como se pode ver no capítulo 3.5.3.9, após realizado o ensaio estático, cada vez que se dá a extração de cada um dos tipos de dados, as respetivas funções devolvem o valor máximo encontrado à função principal. Isto é feito para que o programa possa criar um dicionário com todos os parâmetros mais importantes do ensaio para uma determinada geometria. Estes parâmetros são depois utilizados no final do estudo, para fazer a análise de uma determinada geometria face a todos os ensaios a que foi submetida, e também, como esta se compara com as outras geometrias.

Os dados do dicionário do ensaio estático são então armazenados da seguinte forma (Figura 211):

- [‘*size*’] – Largura do hexágono da geometria;
- [‘*height*’] – Altura do hexágono da geometria;
- [‘*thickness*’] – Espessura do hexágono da geometria;
- [‘*test*’] – Teste que foi realizado. Neste contexto o valor é sempre igual a ‘estatico’;
- [‘*stress*’] – Valor máximo de tensão detetado durante o ensaio, em Mega Pascal;
- [‘*displacement*’] – Valor máximo do deslocamento do impactor, em milímetros;
- [‘*plastic_strain’*] – Valor máximo da deformação plástica;
- [‘*weight*’] – Massa da peça, em toneladas;
- [‘*path*’] – Guarda a diretoria onde se deve executar o ensaio seguinte;

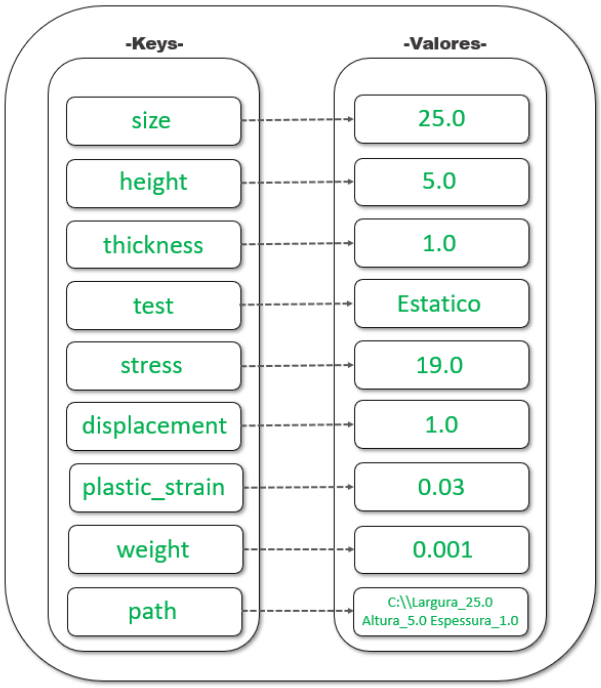


Figura 211 - Representação visual do dicionário gerado no fim do ensaio estático

Este dicionário, por sua vez é armazenado dentro de uma lista, no qual cada valor representa um dado ensaio. Esta estrutura de dados para armazenamento interno de resultados, no ramo de ciências dos computadores tem a designação de *Hash Table* ou *Hash Map*, e permite associar uma chave, a um determinado conjunto de dados [63]. Esta estrutura permite ao programa armazenar, alterar e analisar os dados de forma eficiente, tornando assim o tratamento de dados muito mais rápido. No contexto do caso de estudo atual, o *Hash Map* tem os seus dados armazenados numa lista, em que cada um dos seus valores corresponde a um ensaio, e dentro dela existe um dicionário com os dados das características e resultados desse mesmo ensaio (Figura 212).

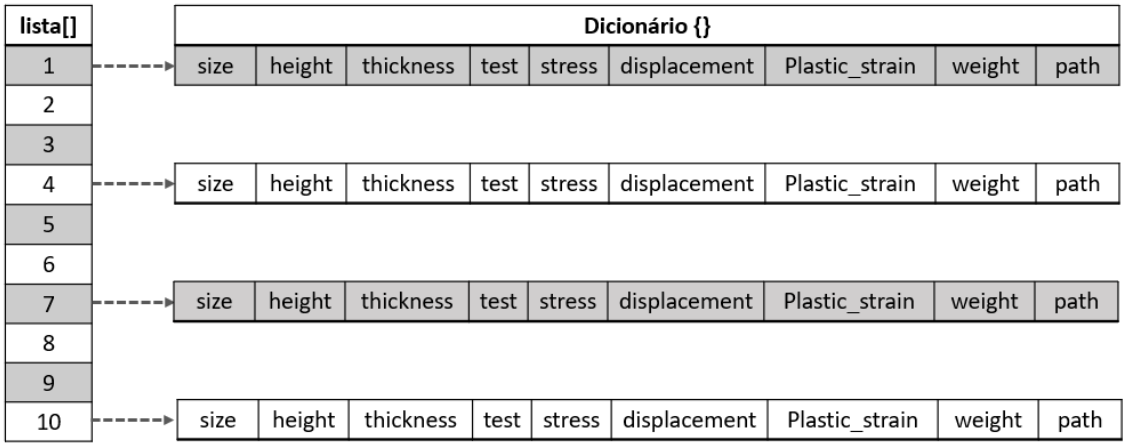


Figura 212 - Demonstração da *Hash Table* que guarda os resultados

Quando terminados todos os ensaios estáticos de um dado estudo, o programa tem de analisar os dados obtidos, de forma a saber quais foram as geometrias que passaram no ensaio estático, e dentro deste grupo ordená-las por ordem crescente da quantidade de material que é necessário para as produzir. Para este processo desenvolveu-se a função *Analise_data*, que recebe o *Hash Map* com os dados das simulações, e os valores limites de tensão e deformação plástica (Figura 213 Linha 2). Com estes valores, a função *Analise_data* filtra o *Hash Map*, através da função *Filter_dict*, que devolve um novo *Hash Map* com apenas os dados de uma dada geometria (Figura 213 Linha 17 e 18) (Figura 214). Com o *Hash Map* filtrado, esta nova estrutura de dados e os valores de falha são enviados para a função *result_confirmation*, que para uma dada geometria, analisa os dados dos ensaios que foram feitos, e se esta passou ou não no teste (Figura 213 Linha 21 e 22) (Figura 215). Se a geometria passar, no seu dicionário é acrescentado um novo parâmetro '*status*', com o valor de '*passed*', se não passar, o valor armazenado é igual a '*failed*'. Depois desta análise, se a geometria passou no ensaio, o seu peso vai ser analisado e se este for o menor valor registado até ao instante, a geometria e o seu peso são ambos guardados em memória (Figura 213 Linha 27-32) (Figura 216). Quando todas as geometrias forem analisadas, a função *Analise_data* vai à geometria com o menor peso, e que passou nos ensaios, e altera o valor do '*status*' para '*best*', informando o

programa de que esta se trata da melhor geometria (Figura 213 Linha 40-46). Analisados os pesos, a função *sort_report* recebe o *Hash Map*, e ordena-o por ordem crescente dos valores de peso das geometrias da peça (Figura 217).

Estando o tratamento dos dados do ensaio estático concluídos, estes são então guardados num ficheiro de texto. Para este processo o *Hash Map* com os dados já todos analisados é enviado para a função *Export_final_report* (Figura 218), que se encarrega de dividir os parâmetros do mapa em listas diferentes, e armazená-las segundo a forma de colunas, em que cada linha representa um ensaio.

Com os dados do ensaio estático todos tratados, o mapa tem de ser alterado de forma a que só as geometrias que passaram no ensaio estático são enviadas para o *Ball drop*. Para isto a função *ball_drop_test_list* (Figura 219) recebe o *Hash Map* dos dados do ensaio estático, e filtra-os pelo valor guardado em '*status*'. Assim o mapa gerado, apenas possui as simulações que passaram no ensaio estático, e estas estão ordenadas por ordem crescente de peso.

```python
# Classifies a geometry and saves the data
def Analise_data(final_report, yield_stress, mat_plastic_limit):

    min_weight = 0
    min_geometry = ''
    # Check if a given geometry passed all it's tests and classify it
    for x in final_report:

        # Starting variables
        weight = float(final_report[x]['weight'])
        size = final_report[x]['size']
        height = final_report[x]['height']
        thickness = final_report[x]['thickness']
        geometry = [size, height, thickness]

        # Filter the dictionary by the geometry
        new_dic = Filter_dict(final_report, size,
            height, thickness)

        # Check if it passed the tests
        result_confirmation=Result_check(new_dic,
            yield_stress, mat_plastic_limit)

        # Classification based on performance
        if result_confirmation:
            # Check if it is the lightest geometry
            if min_weight == 0:
                min_weight = weight
                min_geometry = geometry
            else:
                min_weight, min_geometry = Minimum_weight(weight,
                    min_weight, geometry, min_geometry)
            # Passed - Passed all the tests
            final_report[x]['status']='passed'
        else:
            # feiled - Failed the tests
            final_report[x]['status']='failed'

    # Check wich geometry, is the lightest and passed the test
    for x in final_report:
        weight = float(final_report[x]['weight'])
        size = final_report[x]['size']
        height = final_report[x]['height']
        geometry = [size, height, thickness]
        if geometry == min_geometry:
            final_report[x]['status']='best'

    # Return the new dictionary
    return final_report
```

Figura 213 – Analise dos dados de todos os ensaios estáticos (*Analise_data*)

```
# Filter the report card for geometries with the same parameters
def Filter_dict(my_dict, size, height, thickness):

  new_dict=[my_dict[x] for x in my_dict if \
    (my_dict[x]['size']==size) and
    (my_dict[x]['height']==height) and
    (my_dict[x]['thickness']==thickness)]

  return new_dict
```

Figura 214 – Filtração dos dados, por geometria (*Filter_dict*)

```
# Report Card analysis
# Checks if a given geometry passes all
# the tests it was subjected to
def Result_check(my_dict, yield_stress, limit_strain):
  result_confirmation = True
  for geometry in my_dict:

    # Passing Criteria for the static test
    if geometry['test'] =='Estatico':
      if not(float(geometry['stress'])<=yield_stress and
         float(geometry['plastic_strain'])<= limit_strain):
           result_confirmation = False
           break

    # Passing Criteria for the Ball Drop test
    if geometry['test']=='Ball Drop':
        if not(float(geometry['stress'])<=yield_stress  and
            float(geometry['plastic_strain'])<= \
                limit_strain):
            result_confirmation = False
            break

    # If a new test is added to the program
    # Write it's pass criteria here
  return result_confirmation
```

Figura 215 – Verificação dos resultados obtidos, com os do cliente (*Result_Check*)

```
# Checks what geometry, from the ones that pass all the tests
# has the least weight amongst them
def Minimum_weight(weight, min_weight, geometry, min_geometry):

  if weight <= min_weight:
    min_weight = weight
    min_geometry = geometry

  return min_weight, min_geometry
```

Figura 216 – Cálculo da geometria de menor peso (*MInimum_weight*)

```
# Sort the report card by weight
def sort_report(final_report):

    sorted_report = OrderedDict(sorted(final_report.items(),
       key = lambda x: getitem(x[1], 'weight')))

    return sorted_report
```

Figura 217 – Ordenação da base de dados para se obter (*sort_report*)

```python
# Save the Report Card data into a text file
def Export_final_report(final_report, my_path):

    # Text file directory
    title = 'Resultados'
    txt_path = my_path + '\\' + title + '.txt'

    # Data management - Create lists
    size=[]
    height=[]
    thickness=[]
    test=[]
    stress=[]
    displacement=[]
    plastic_strain=[]
    weight=[]
    status=[]
    path=[]
    my_path_list = [my_path]

    # Fill the lists with the respective values
    for x in final_report:
        size.append(float(final_report[x]['size']))
        height.append(float(final_report[x]['height']))
        thickness.append(float(final_report[x]['thickness']))
        test.append(final_report[x]['test'])
        stress.append(final_report[x]['stress'])
        displacement.append(final_report[x]['displacement'])
        plastic_strain.append(final_report[x]['plastic_strain'])
        weight.append(final_report[x]['weight'])
        status.append(final_report[x]['status'])
        path.append(final_report[x]['path'])

    # Save data into a text file
    with open(txt_path, 'w') as f:
        writer = csv.writer(f, delimiter='\t')
        for x in it.izip_longest(size, height,
            thickness, test,
            stress,displacement,
            plastic_strain, weight, status, path,
            my_path_list, fillvalue=''):

            writer.writerow(x)
```

Figura 218 -Exportação dos resultados (*Export_final_report*)

```
1 # Get the geometrys that passed the static test
2 def ball_drop_test_list(my_dict):
3 
4     new_dict=[my_dict[x] for x in my_dict if (
5         my_dict[x]['status']=='passed' or
6         my_dict[x]['status']=='best')]
7 
8     return new_dict
```

Figura 219 – Ordenação do hashmap, para o *Ball drop* (*ball_drop_test_list*)

3.5.3.13 Ensaio do Ball Drop

O segundo teste para a validação das peças, que foi pedido pela empresa, consiste em largar uma esfera de aço, com uma massa e um diâmetro de respetivamente 500 g e 50 mm, a uma altura de 500 mm da peça base, à temperatura de 23ºC, e verificar se o impacto resultante faz com que peça ultrapasse a sua tensão máxima (28.92 MPa) ou, se a deformação permanente é superior à requerida (2.48%). Sendo assim, tal como no ensaio estático, o programa tem de possuir uma função que seja capaz de:

1. Impor os contactos entre os diferentes componentes do modelo (Base e Bola);
2. Criar os *steps* necessários para o ensaio;
3. Criar as *boundary conditions*;
4. Criar o *field output*;
5. Criar o *history output*;
6. Submeter o job para ser calculado;
7. Analisar se o job foi concluído com sucesso;
8. Extrair os resultados da simulação;

A interação entre a peça base e a bola é feita através da função *Interaction_Ball_Drop_Test* (Figura 220) que, tal como no ensaio estático, estabelece uma interação *Surface to Surface contact* entre a superfície da base e a da bola, através do objeto *SurfaceToSurfaceContactExp* [87] (Figura 220 Linha 23-32). As propriedades desta interação são também as mesmas que no caso estático, sendo aplicado um *Normal Contact* com um modelo de *Hard Contact*, através da metodologia *Augmented Lagrange* (Figura 220 Linha 9-20) [61].

```
# Create the interaction an interaction properties
# between the different components in the static test
def Interaction_Ball_Drop_Test():

    # Internal Variable
    a = mymodel.rootAssembly

    # Create interaction: Ball contact
    mymodel.ContactProperty('IntProp-Ball')
    mymodel.interactionProperties['IntProp-Ball'].TangentialBehavior(
        formulation=PENALTY, directionality=ISOTROPIC,
        slipRateDependency=OFF, pressureDependency=OFF,
        temperatureDependency=OFF, dependencies=0,
        table=((0.22, ), ), shearStressLimit=None,
        maximumElasticSlip=FRACTION,
        fraction=0.005, elasticSlipStiffness=None)

    mymodel.interactionProperties['IntProp-Ball'].NormalBehavior(
        pressureOverclosure=HARD, allowSeparation=ON,
        constraintEnforcementMethod=DEFAULT)

    # Create face to face interaction
    region1=a.instances['Ball-1'].surfaces['SURF']
    region2=a.instances['Base-1'].surfaces['SURF']

    mymodel.SurfaceToSurfaceContactExp(name ='Int-Ball',
        createStepName='Initial', master = region1,
        slave = region2, mechanicalConstraint=KINEMATIC,
        sliding=FINITE,
        interactionProperty='IntProp-Ball',
        initialClearance=OMIT, datumAxis=None,
        clearanceRegion=None)
```

Figura 220 - Função de imposição das interações entre componentes (*Interaction_Ball_Drop_Test*)

Neste ensaio é usada uma análise do tipo explícita. Como simulações deste tipo têm um tempo de resolução maior, tentou-se reduzir ao máximo a duração do tempo real do ensaio. Uma simplificação que foi empregue na simulação do programa, foi o de não largar a bola, da sua posição de repouso, a uma distância de 500 mm da peça, mas sim a uma distância de apenas 1.6mm (1.5mm da espessura da peça e 0.1mm de folga). No caso referido, a velocidade inicial da bola não é nula, mas sim um determinado valor que foi calculado através das equações do movimento (equação (63)) [50].

$$\begin{Bmatrix} x(t) = x_0 + v_o t + \frac{1}{2} g t^2 \\ v(t) = \frac{dx_{(t)}}{dt} = v_o + gt \end{Bmatrix} \quad (63)$$

resultando:

$$v(t) = \sqrt{v_0^2 + 2g * (x(t) - x_0)} \quad (64)$$

onde,

- $x(t)$ – "Posição da bola após t segundos" (mm)
- x_0 – "Posição da bola no instante $t = 0$ segundos" (mm)
- $v(t)$– "Velocidade da bola após t segundos" (mm/s)
- v_0 – "Velocidade da bola no instante $t = 0$ segundos" (mm/s)
- t – "Instante de tempo" (s)
- g- "Aceleração da gravidade na Terra" (mm/s^2)

Sendo $x(t)$ igual a 500 mm, x_0 e v_0 tomados como nulos e g igual a 9810 mm/s^2 (Figura 221), consegue-se calcular o instante de tempo que a bola demora a chegar à posição final, e qual a velocidade da bola nesse instante.

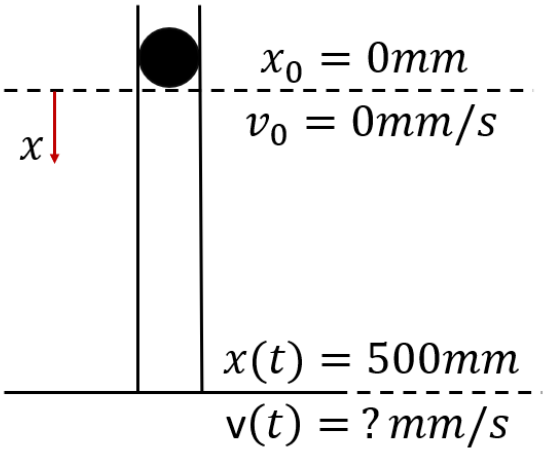


Figura 221 – Demonstração do cálculo da velocidade da bola

$$\begin{cases} 500 = \frac{1}{2} * 9810 * t^2 \Leftrightarrow t = 0.3193\ s \\ v(t) = 9810t = 3130\ mm/s \end{cases}$$

Com o valor da velocidade da bola no momento de impacto, escusa-se de se fazer a simulação desde o ponto onde a bola é largada, até ao impacto, reduzindo o tempo real da simulação em 0.319 segundos, o que em termos da simulação do Abaqus®, corresponde a uma redução de várias horas. Para além da velocidade inicial e do tipo de contacto, é importante definir como é que a bola se pode movimentar durante o ensaio. No teste físico, a bola é colocada dentro de um tubo, onde é posteriormente largada, este tubo não permite à bola movimentar-se em qualquer outra direção, que não seja na vertical, porém permite que esta possa rodar. Como a simulação tem uma duração de apenas 10 milissegundos e a bola é largada do seu repouso, considerou-se que esta também não roda durante o processo. Com tudo isto em mente, o programa cria dois steps diferentes:

- *Initial – Step* onde são impostas as condições iniciais do modelo. Aqui são impostos os encastramentos, simetrias e/ou anti-simetrias da peça base, e bloqueiam-se os deslocamentos da bola, em todas as direções e rotações, com a exceção da translação segundo o eixo dos x's (Figura 222 Linhas 20-27)(Devido à infinidade de possibilidades que a peça base pode ter em termos das suas condições de fronteira, estas características são impostas no modelo base, que se pretende estudar);
- *Step-Ball_Drop* – Neste step é então aplicada a velocidade inicial na bola (Figura 222 Linhas 14, 17-18 e 30-32);

Para além da imposição das condições de ensaio é necessário impor ao Abaqus® quais são as variáveis que se quer que sejam calculadas, e, portanto, o programa cria um *field output* para todo o modelo, que estuda a Tensão e a Deformação plástica (Figura 222 Linhas 35-37).

Com as características do ensaio todas impostas, o programa cria um job e, se o utilizador impôs no início do estudo que quer que este seja resolvido, o programa submete o job e espera que este seja concluído (Figura 222 Linhas 40-46)(Figura 223 Linhas 49-58).

Tal como no ensaio estático e função correspondente, a função *Ball_Drop_Test*, enquanto calcula os resultados da simulação, está em paralelo a monitorizar a simulação, e a ver se esta é feita corretamente, e caso o seja, utiliza as mesmas funções para extrair os resultados, armazená-los e avaliá-los da mesma forma (Figura 223 e Figura 224). Contudo, a função *Ball_Drop_Test* possui uma funcionalidade, que o ensaio estático não tem. No ensaio estático, o momento mais crítico da simulação (instante de tempo onde as tensões da peça são mais elevadas) corresponde ao momento em que a força de aplicação possui o seu valor máximo, o que no referido ensaio corresponde ao último *frame* da simulação. No *Ball Drop*, este já não é o caso, como a bola embate com a superfície da peça, e subsequentemente ressalta, o programa tem de ser capaz de encontrar qual é o instante mais crítico dentro deste

intervalo de tempo, e para isto desenvolveu-se a função *Critical_Frame* que é tratada no capítulo 3.5.3.13.1.

```
# Create the ball drop test parameters
def Ball_Drop_Test(solve_model, job_name, my_path_1,
    final_report_ball_drop, report_counter,
    hex_size, hex_height, hex_thickness, yield_stress, limit_strain):

    # Internal Variables
    a = mymodel.rootAssembly
    da = a.datums
    converted_report = []
    best_result = "False"
    Ball_Load_Point_CSYS_id = a.features['Ball_Drop_CSYS'].id

    # Initial velocity
    v0 = 3130 #mm/s

    # Step
    mymodel.ExplicitDynamicsStep(name='Step-Ball_Drop',
        previous='Initial', timePeriod=0.005, improvedDtMethod=ON)

    # BC: Displacement
    region = a.sets['Ball_Drop_RP']
    local_CSYS = da[Ball_Load_Point_CSYS_id]
    mymodel.DisplacementBC(name='BC-PROJECTILE',
        createStepName='Initial', region=region,
        u1=UNSET, u2=SET, u3=SET,
        ur1=SET, ur2=SET, ur3=SET,amplitude=UNSET,
        distributionType=UNIFORM, fieldName='', localCsys=local_CSYS)

    # Initial condition: velocity
    mymodel.Velocity(name='PROJECTILE_VELOCITY', region=region,
        field='', distributionType=MAGNITUDE, velocity1=v0,
        velocity2=0.0, velocity3=0.0, omega=0.0)

    # Field Output
    mymodel.FieldOutputRequest(name='F-Output-Ball_Drop_Test',
        createStepName='Step-Ball_Drop', variables=('S','PE'),
        frequency=100)

    # Job
    myjob = mdb.Job(name= job_name, model=mymodel.name, description='',
        type=ANALYSIS, atTime=None, waitMinutes=0, waitHours=0,
        queue=None, memory=90, memoryUnits=PERCENTAGE,
        explicitPrecision=DOUBLE_PLUS_PACK, nodalOutputPrecision=SINGLE,
        echoPrint=OFF, modelPrint=OFF,
        contactPrint=OFF, historyPrint=OFF, userSubroutine='',
        scratch='', resultsFormat=ODB, numCpus=1)
```

Figura 222 - Função do ensaio do *Ball Drop* (*Ball_Drop_Test*)(1/3)

```
48      # Submit the job
49      if solve_model:
50
51          # Monitor the job
52          monitorManager.addMessageCallback(jobName=job_name,
53          messageType=ANY_MESSAGE_TYPE, callback=jobMonitorCallback,
54          userData=None)
55
56          # Submit the job and wait for it's completion
57          myjob.submit()
58          myjob.waitForCompletion()
59
60          # Extract the data, if the job was completed successfully
61          if jobMonitorCallback.job_completed == True:
62
63              # Step name
64              step_name = 'Step-Ball_Drop'
65
66              # Find the critical element
67              study_element_label = critical_elements()
68
69              # Find the critical frame
70              study_frame =Critical_Frame(job_name,
71                  step_name, study_element_label)
72
73              # Create animations and screenshots of the results
74              create_animations(job_name, path = my_path_1)
75              screenshot(job_name, step_name ,path = my_path_1)
76
77              # Exclude the fixtures values
78              exclusion_element_list=[]
79              exclusion_element_list = Exclusion_Area(job_name)
80
81              # Extract the data from the odb
82              max_stress = Export_Stress_Results(job_name,
83                  exclusion_element_list, step_name,
84                  study_frame, mat_yield_stress = yield_stress,
85                  my_path = my_path_1)
86
87              max_plastic_strain = Export_Plastic_Strain_Results(job_name,
88                  exclusion_element_list, step_name, study_frame,
89                  mat_limit_strain=limit_strain, my_path=my_path_1)
90
91              # Get the total mass of the component
92              p = mymodel.parts['Base']
93              myview.setValues(displayedObject=p)
94              part_mass = mymodel.parts['Base'].getMassProperties()['mass']
```

Figura 223 - Função do ensaio do *Ball Drop* (*Ball_Drop_Test*)(2/3)

```
95            # Update Report Card
96             final_report_ball_drop[report_counter] = {'size':hex_size,
97                 'height':hex_height ,'thickness':hex_thickness,
98                 'test':'Ball Drop','stress':max_stress,
99                 'plastic_strain':max_plastic_strain,
100                'weight':part_mass, 'status':best_result}
101
102            # Check if the geometry passed the simulation
103            converted_report.append(\
104                final_report_ball_drop[report_counter])
105
106            result_confirmation = Result_check(converted_report,
107                yield_stress, limit_strain)
108
109            # If the geometry passed the tests change it's status value
110            if result_confirmation:
111                best_result = "best"
112            final_report_ball_drop[report_counter]['status']=best_result
113
114            report_counter+=1
115
116
117    return final_report_ball_drop, report_counter, best_result
```

Figura 224 - Função do ensaio do *Ball Drop* (*Ball_Drop_Test*)(3/3)

3.5.3.13.1 Procura do instante critico e concentrações de tensões

Como foi mencionado no capítulo anterior, o programa tem de ser capaz de procurar qual é o instante de tempo mais crítico de toda a simulação, para poder extrair os dados e avaliá-los corretamente. Uma abordagem a este problema seria o de inspecionar o momento em que a bola inverte o sentido do movimento devido ao ressalto, ou seja, o momento em que a velocidade é igual a 0 mm/s, porém esta metodologia possui o seu erro associado. Como se pode ver no exemplo da Figura 225, para os elementos em estudo, o instante crítico dá-se instantes antes da bola inverter o sentido do movimento, e, portanto, esta metodologia não é a mais precisa. Outra metodologia consiste em avaliar apenas um elemento do modelo e ver qual é o instante em que a tensão instalada possui o seu valor máximo (instante crítico), sendo depois esse instante utilizado para avaliar toda a peça. Isto tem um bom nível de proximidade, como se pode ver na Figura 225, porém corre-se o risco de o elemento que se está a avaliar não ser o mais correto para a simulação, pode acontecer que o elemento escolhido possua uma concentração de tensões num instante que não corresponda ao momento crítico da peça, ou que durante todo o processo este possua valores nulos (Figura 226). Sendo assim, a metodologia empregue, não analisa apenas um elemento, analisa um *set* completo deles, e vê qual é o instante de tempo que a maioria classifica como crítico.

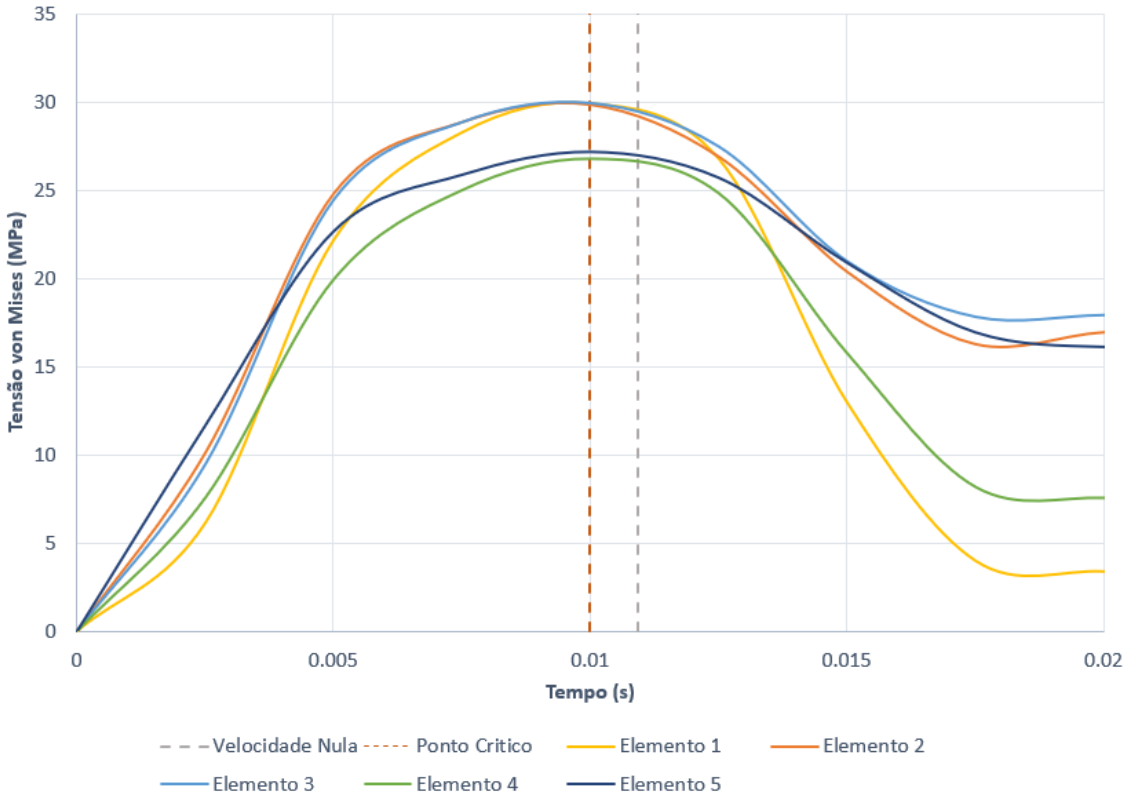


Figura 225 – Discrepância da metodologia da velocidade nula para o ensaio do *Ball Drop*

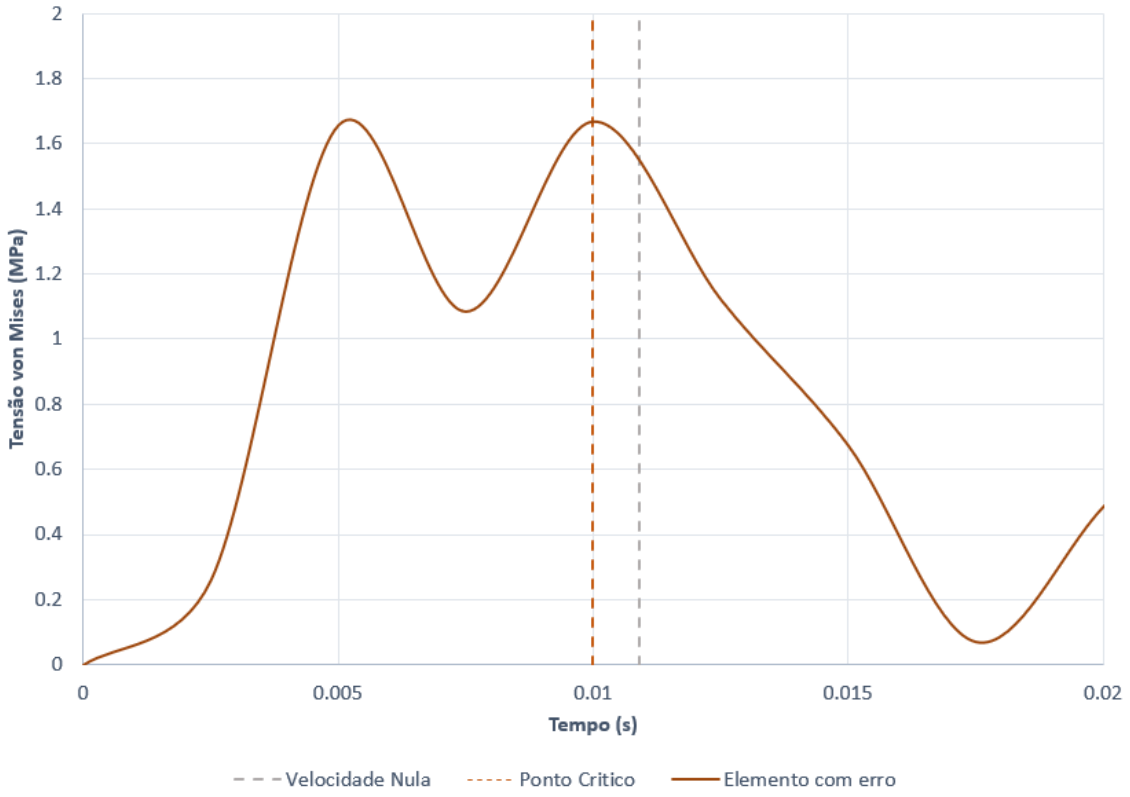


Figura 226 – Discrepância de alguns elementos com comportamentos anormais

O processo de análise do instante de tempo crítico é então realizado por duas funções: *critical_elements* e *Critical_Frame* (Figura 231 e Figura 232). O algoritmo resultante destes dois processos consiste em criar uma zona circular na ponta do impactor, através do método *getByBoundingSphere* [74], que vai buscar os *MeshElement objects* de cada um dos elementos da peça base que se encontra completamente inserido dentro da referida zona (Figura 227) [88]. Destes elementos são extraídas as *labels* de cada um, e inseridas dentro de uma lista, que depois é enviada para a função *Critical_Frame* que trata de analisar o *frame* que corresponde ao instante de tempo crítico de cada um e armazená-lo num *Hash Map*. Depois da análise, o programa analisa qual é o frame que foi declarado mais vezes e utiliza-o para a análise do *Ball Drop*. Isto permite que o algoritmo não seja influenciado por concentrações de tensões em certos elementos, como os elementos 1 e 2 da Figura 228, que devido à vibração da peça, acusam um instante crítico diferente. Este processo pode ser descrito pelo fluxograma da Figura 229 e da Figura 230.

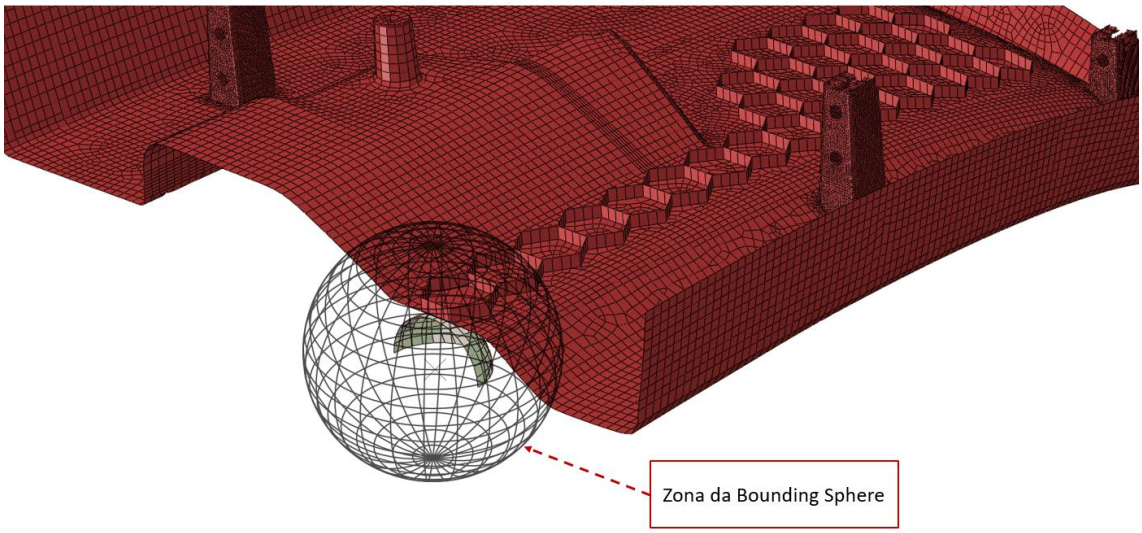


Figura 227 – Zona da *bounding sphere* gerada na bola

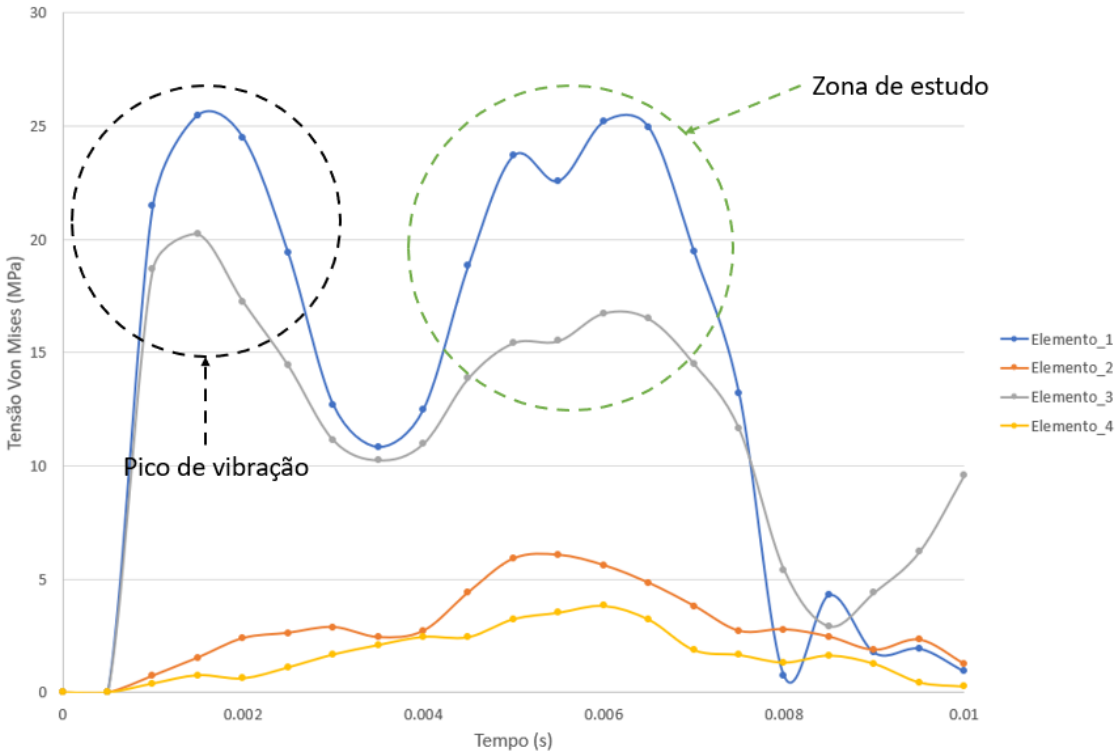


Figura 228 – Demonstração de um pico de concentrações de tensões

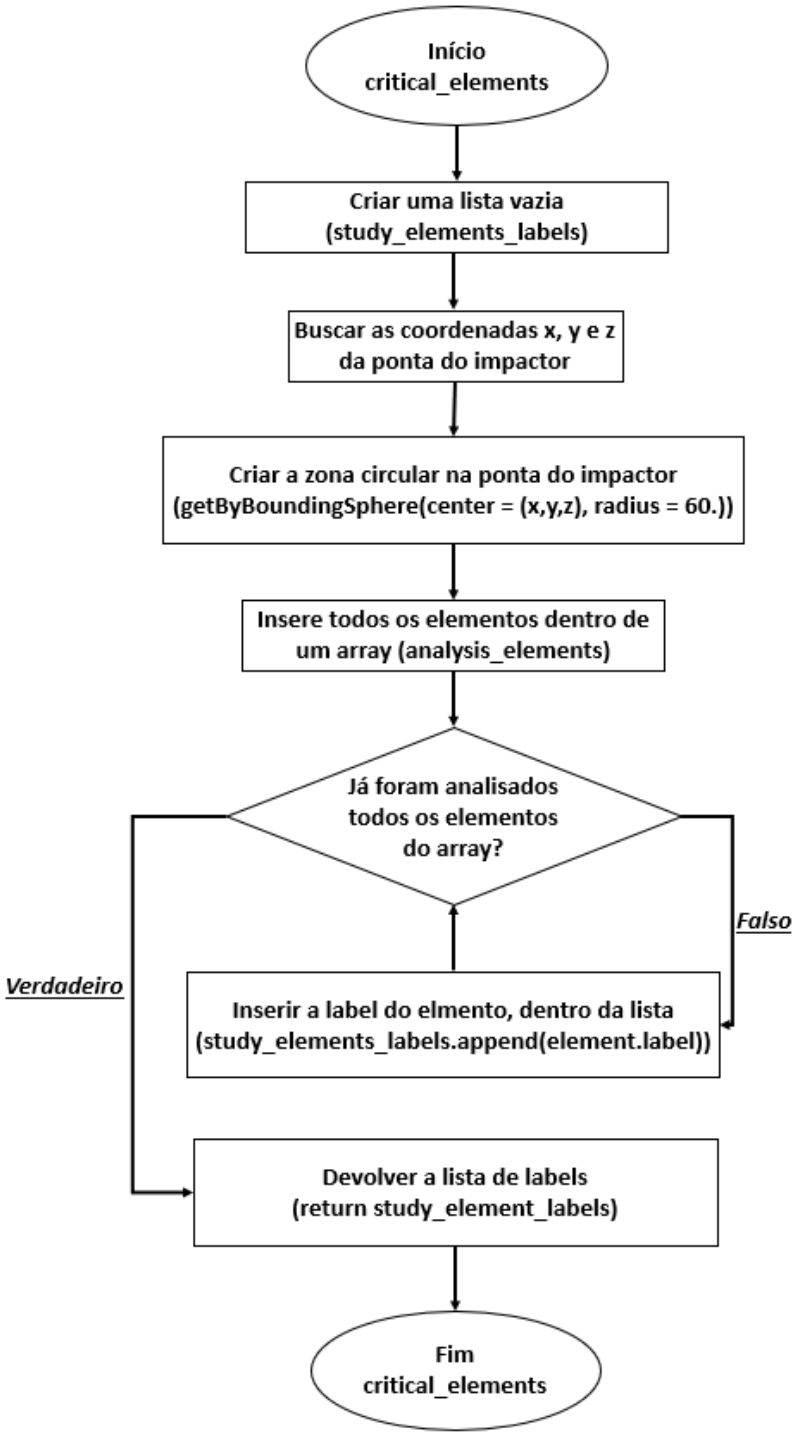


Figura 229 – Procura dos elementos para análise (critical_elements)(fluxograma)

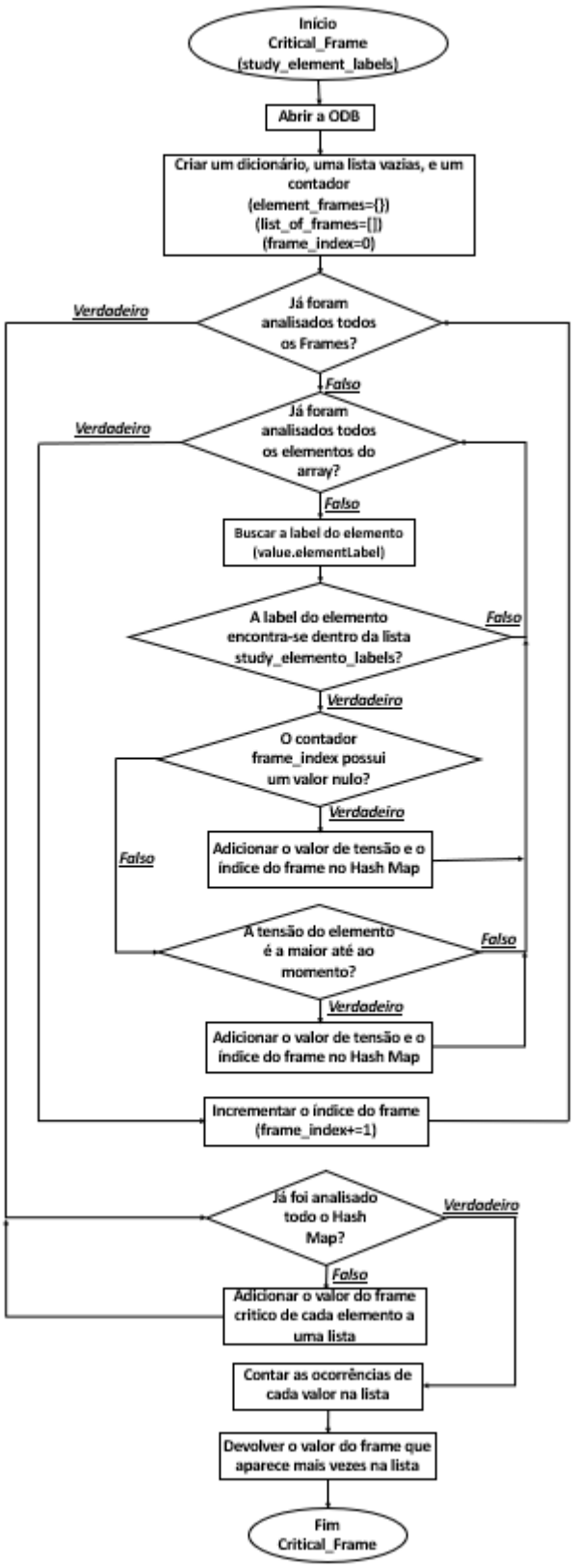


Figura 230 – Procura dos elementos para análise (Critical_Frame)(fluxograma)

```
# Get elements for the frame study
def critical_elements():

    # Abaqus variables
    a = mymodel.rootAssembly
    p = mymodel.parts['Base']
    study_elements_labels=[]

    # Get impactor coordinates
    impactor_point = a.sets['Ball_Drop_RP'].referencePoints[0]
    impactor_point_RP = a.getCoordinates(impactor_point)

    # Get center point x, y and z values
    center_x = impactor_point_RP[0]
    center_y = impactor_point_RP[1]
    center_z = impactor_point_RP[2]

    # Get array of elements to study
    analysis_element = p.elements.getByBoundingSphere(\
        center = (center_x, center_y, center_z), radius = 60.)

    # Convert the array into a list of labels
    for element in analysis_element:
        study_elements_labels.append(element.label)

    # Return the list of element labels
    return study_elements_labels
```

Figura 231 – Função de busca dos elementos (*critical_elements*)

```
# Export all the stress values in a given odb
def Critical_Frame(odb_name, step_name, study_element_label):

    # Open the odb
    odb = session.openOdb(name=odb_name + '.odb')

    # Internal Variables
    element_frames = {}
    list_of_frames=[]

    # Get the critical frames from the selected elements
    fieldVarFrames = odb.steps[step_name].frames
    frame_index=0
    for frame in fieldVarFrames:
        for value in frame.fieldOutputs['S'].values:
            element_label = value.elementLabel
            if element_label in study_element_label:

                if frame_index == 0:
                    element_frames[element_label]={'t_max':value.mises,
                    'f_max':frame_index}
                else:
                    if value.mises>= \
                        element_frames[element_label]['t_max']:
                        element_frames[element_label]['t_max']=value.mises
                        element_frames[element_label]['f_max']=frame_index
        frame_index+=1

    # Check witch one ocurress the most
    for element_table in element_frames:
        list_of_frames.append(element_frames[element_table]['f_max'])
    occurence_count = Counter(list_of_frames)
    study_frame = occurence_count.most_common(1)[0][0]

    return study_frame
```

Figura 232 – Função de busca do *frame* critico (*Critical_Frame*)

3.5.3.13.2 Tratamento e análise de dados do *Ball Drop*

Com as simulações dos ensaios dinâmicos concluídos, o mapa com os dados dos resultados do *Ball Drop* tem de ser combinado com o do ensaio estático de forma a obter-se uma estrutura de dados com os resultados devidamente organizados e de fácil acesso. Para este processo empregou-se a função *Combine_dict* (Figura 234 e Figura 235), que recebe os dois mapas dos ensaios, e organiza-os por geometrias. Em concreto, a função acede primeiro ao primeiro ensaio do mapa do estudo estático, e adiciona-o ao novo mapa combinado, e depois filtra o mapa do *Ball Drop*, de forma a que devolva apenas os resultados do ensaio com a mesma geometria do estático, e adiciona-o ao novo mapa. Este processo repete-se para todas as geometrias, e no final obtém-se um mapa em que todas as geometrias estão ordenadas por ordem crescente do material que necessitam para ser produzidas e estão também organizadas por ensaios, como representado na Figura 233.

-Mapa do ensaio estático-

lista[]	Dicionário {}									
	size	height	thickness	test	stress	displacement	plastic_strain	weight	status	path
1	25	3	3.5	estatico	20	2	0.06	0.001	failed	(...)
2	20	3	3.5	estatico	16	1.7	0.035	0.002	best	(...)
3	25	6	3.5	estatico	17	1.5	0.04	0.003	passed	(...)
4	20	6	3.5	estatico	15	1	0.03	0.004	passed	(...)

+

-Mapa do Ball Drop-

lista[]	Dicionário {}									
	size	height	thickness	test	stress	displacement	plastic_strain	weight	status	path
1	20	3	3.5	ball drop	18	NC	0.045	0.002	best	(...)
2	25	6	3.5	ball drop	22	NC	0.07	0.003	failed	(...)
3	20	6	3.5	ball drop	16	NC	0.035	0.004	passed	(...)

=

-Mapa Combinado-

lista[]	Dicionário {}									
	size	height	thickness	test	stress	displacement	plastic_strain	weight	status	path
1	20	3	3.5	estatico	16	1.7	0.035	0.002	NC	(...)
2	20	3	3.5	ball drop	18	NC	0.045	0.002	best	(...)
3	25	6	3.5	estatico	17	1.5	0.04	0.003	NC	(...)
4	25	6	3.5	ball drop	22	NC	0.07	0.003	failed	(...)
5	20	6	3.5	estatico	15	1	0.03	0.004	NC	(...)
6	20	6	3.5	ball drop	16	NC	0.035	0.004	passed	(...)

Figura 233 – Representação esquemática da combinação de mapas

```
# Combine the Static test Hash Map and the
# Ball drop Hash Map into one
def Combine_dict(final_report_ball_drop, ball_drop_list):

    # Internal Variables
    combined_dict = {}
    combined_index = 0

    # Add the values from the static test Hash Map
    # To the new combined Hash Map
    for static_test in ball_drop_list:
        combined_dict[combined_index] = {}
        combined_dict[combined_index]['size']= \
            static_test['size']
        combined_dict[combined_index]['height']= \
            static_test['height']
        combined_dict[combined_index]['thickness']= \
            static_test['thickness']
        combined_dict[combined_index]['test']= \
            static_test['test']
        combined_dict[combined_index]['stress']= \
            static_test['stress']
        combined_dict[combined_index]['displacement']= \
            static_test['displacement']
        combined_dict[combined_index]['plastic_strain']= \
            static_test['plastic_strain']
        combined_dict[combined_index]['weight']= \
            static_test['weight']
        combined_dict[combined_index]['status']='NC'
```

Figura 234 – Combinação dos mapas dos dois ensaios (*Combine_dict*) (1/2)

```
31          combined_index+=1
32          size=static_test['size']
33          height=static_test['height']
34          thickness=static_test['thickness']
35
36          # Filter the Ball Drop Hash Map so that the
37          # only geometry that appears, is the same as the one
38          # in the static test
39          filtered_dict = Filter_dict(final_report_ball_drop,
40              size, height, thickness)
41
42          # if the Hash Map value for the Ball Drop isn't Null
43          # That the value will be added into the new combined
44          # Hash Map
45          if filtered_dict:
46              combined_dict[combined_index] = {}
47              combined_dict[combined_index]['size']= \
48                  filtered_dict[0]['size']
49              combined_dict[combined_index]['height']= \
50                  filtered_dict[0]['height']
51              combined_dict[combined_index]['thickness']= \
52                  filtered_dict[0]['thickness']
53              combined_dict[combined_index]['test']= \
54                  filtered_dict[0]['test']
55              combined_dict[combined_index]['stress']= \
56                  filtered_dict[0]['stress']
57              combined_dict[combined_index]['displacement']='NC'
58              combined_dict[combined_index]['plastic_strain']= \
59                  filtered_dict[0]['plastic_strain']
60              combined_dict[combined_index]['weight']= \
61                  filtered_dict[0]['weight']
62              combined_dict[combined_index]['status']= \
63                  filtered_dict[0]['status']
64
65              combined_index+=1
66
67      # Return the new Combined Hash Map
68      return combined_dict
```

Figura 235 – Combinação dos mapas dos dois ensaios (*Combine_dict*) (2/2)

```python
# Save the Report Card data into a text file
def Results(combined_dict, my_path):

    # Text file directory
    title = 'Resultados_finais'
    txt_path = my_path + '\\' + title + '.txt'

    # Data management - Create lists
    size=[]
    height=[]
    thickness=[]
    test=[]
    stress=[]
    displacement=[]
    plastic_strain=[]
    weight=[]
    color=[]
    for x in combined_dict:
        size.append(combined_dict[x]['size'])
        height.append(combined_dict[x]['height'])
        thickness.append(combined_dict[x]['thickness'])
        test.append(combined_dict[x]['test'])
        stress.append(combined_dict[x]['stress'])
        displacement.append(combined_dict[x]['displacement'])
        plastic_strain.append(combined_dict[x]['plastic_strain'])
        weight.append(combined_dict[x]['weight'])
        color.append(combined_dict[x]['status'])

    # Save data into a text file
    with open(txt_path, 'w') as f:
        writer = csv.writer(f, delimiter='\t')
        for x in it.izip_longest(size, height,
            thickness, test,
            stress,displacement,
            plastic_strain, weight, color, fillvalue=''):
            writer.writerow(x)
```

Figura 236 – Extração dos dados do mapa combinado (*Results*)

3.5.3.14 Função principal do programa

Para que todas as funções do programa possam interagir umas com as outras e serem utilizadas nas alturas corretas, desenvolveu-se a função *main*, que funciona como a função principal de todo o programa. Aqui, todas as funções dos capítulos anteriores são chamadas pela ordem necessária para resolver um dado caso de estudo. A função da Figura 241 à Figura 247, pode ser representada pelo fluxograma da Figura 237 à Figura 240.

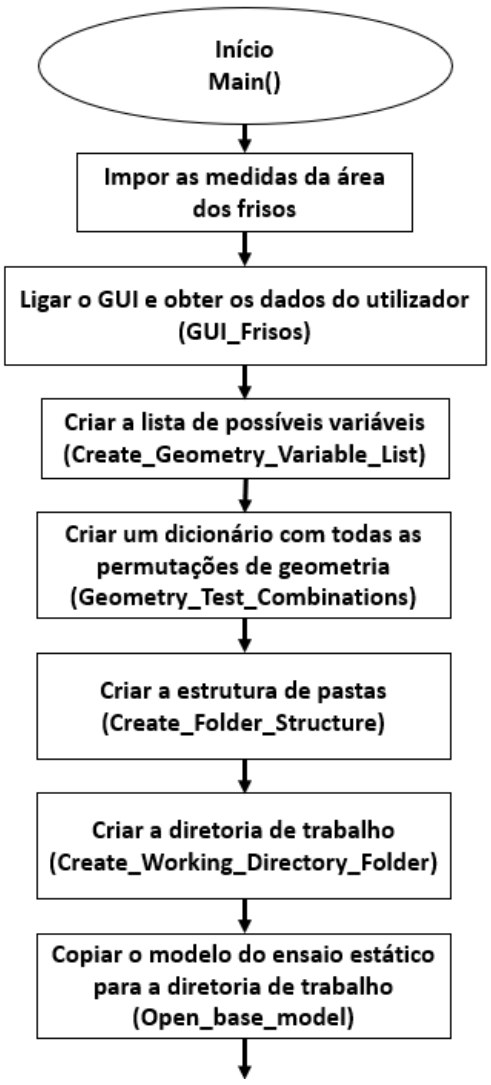


Figura 237 – Fluxograma do programa principal (1/4)

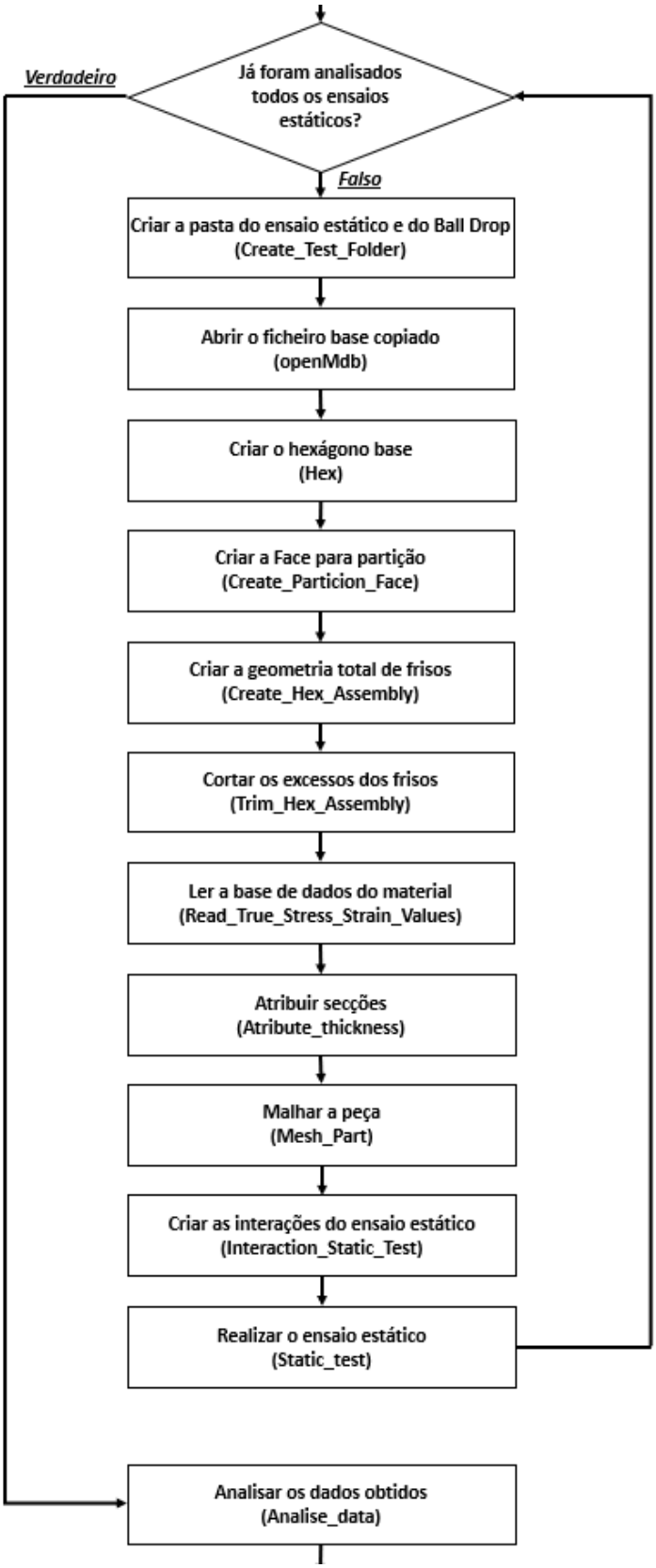


Figura 238 – Fluxograma do programa principal (2/4)

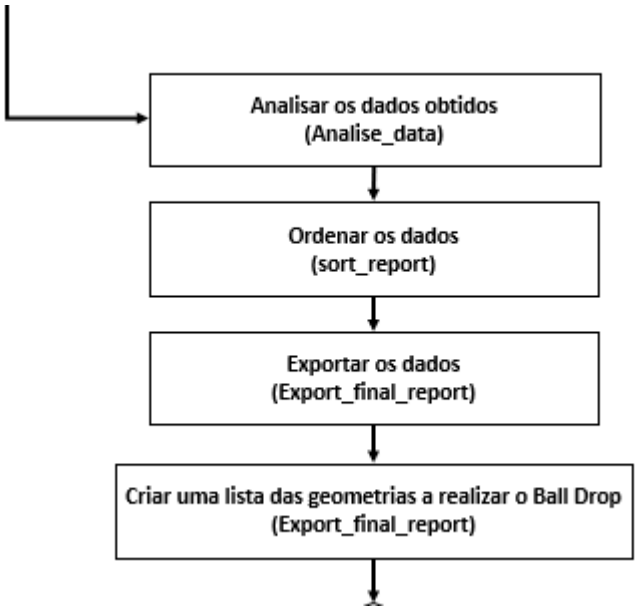


Figura 239 – Fluxograma do programa principal (3/4)

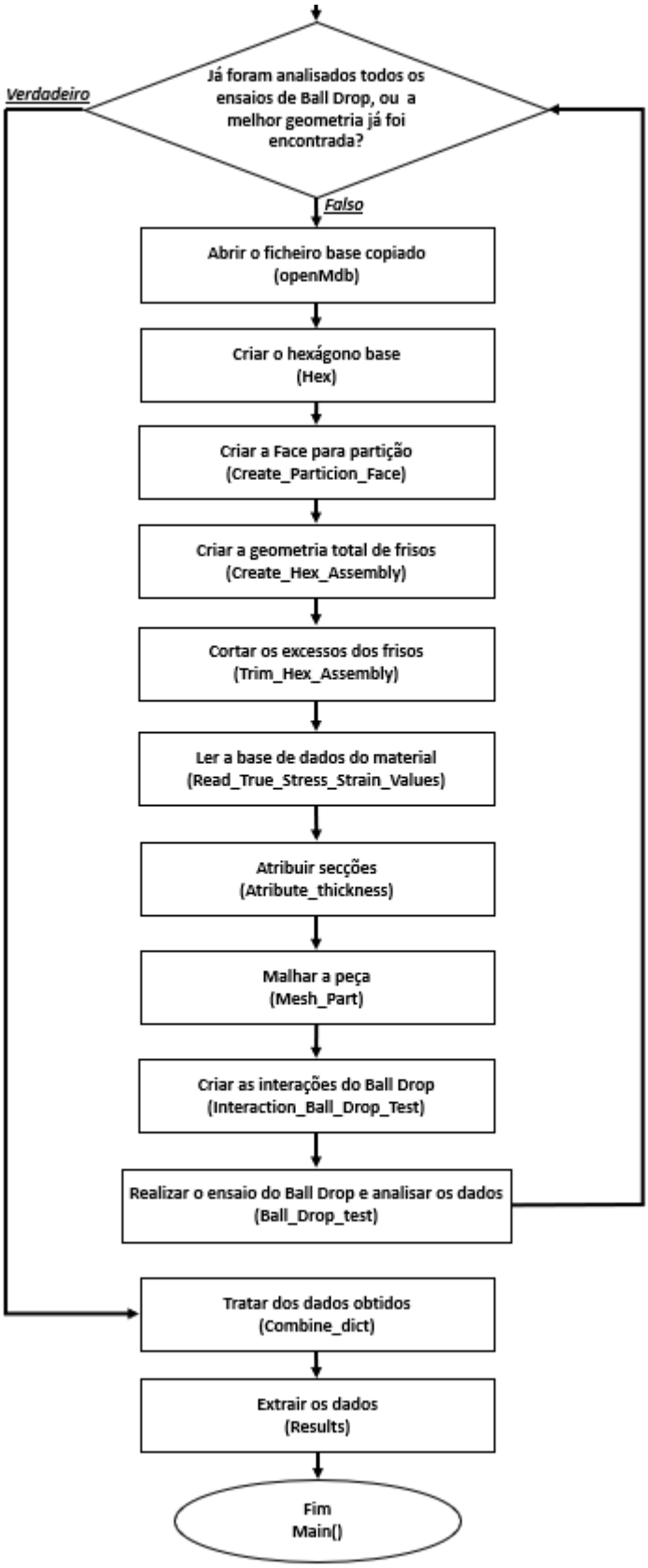


Figura 240 – Fluxograma do programa principal (4/4)

```python
def main():

    # Global variables
    global myview
    global mymodel
    myview = session.viewports['Viewport: 1']
    mymodel = mdb.models['Model-1']
    final_report={}
    report_counter = 0

    # Hex area parameters
    A = 40
    B = 476
    C = 125
    D = 160
    E = 21.15
    F = 81
    G = C-E-A
    H = B-F-D

    # Call the GUI window and get a
    # dictionary with all the simulation values
    GUI_values_main = GUI_Frisos()

    # Get a list with all the values
    # that each of the hexagon variables can have
    list_val_lenght = Create_Geometry_Variable_List(
        GUI_values_main['Lenght_max_GUI'],
        GUI_values_main['Lenght_min_GUI'],
        GUI_values_main['Lenght_n_val_GUI'])
    list_val_thickness = Create_Geometry_Variable_List(
        GUI_values_main['Thickness_max_GUI'],
        GUI_values_main['Thickness_min_GUI'],
        GUI_values_main['Thickness_n_val_GUI'])
    list_val_height = Create_Geometry_Variable_List(
        GUI_values_main['Height_max_GUI'],
        GUI_values_main['Height_min_GUI'],
        GUI_values_main['Height_n_val_GUI'])
    test_combinations_dict = Geometry_Test_Combinations(
        list_val_lenght, list_val_thickness, list_val_height)

    # Create folder structure
    print('A criar o sistema de pastas. [1/10]')
    dir_path_main = Create_Folder_Structure()
    cwd_path_main = Create_Working_Directory_Folder(dir_path_main)
    model_path = Open_base_model(cwd_path_main, 'Estatico')
```

Figura 241 – Função principal (*main*)(1/7)

```
48      # Set main variables
49      base_thickness_main=GUI_values_main['base_thickness_GUI']
50      if GUI_values_main['simulation_GUI'] == 1:
51          solve_model = True
52      else:
53          solve_model = False
54
55      # Loop throught tests
56      for test_number in range(len(test_combinations_dict)):
57
58          # Hexagon parameters
59          hex_size_main = test_combinations_dict[test_number]['Lenght']
60          hex_height_main = test_combinations_dict[test_number]['Height']
61          hex_thickness_main = \
62              test_combinations_dict[test_number]['Thickness']
63          hex_circle_height_main = hex_size_main/np.cos(np.radians(30))
64          hex_lenght_main = hex_size_main*np.tan(np.radians(30))
65
66          # Job names for each test
67          job_name_static_test=''+str(GUI_values_main['job_name_GUI'])+\
68              '_L_'+str(hex_size_main).replace(".",",")+\
69              '_A_'+str(hex_height_main).replace(".",",")+\
70              '_E_'+str(hex_thickness_main).replace(".",",")+\
71              '_Static_Test'
72
73          job_name_ball_drop=''+str(GUI_values_main['job_name_GUI'])+\
74              '_L_'+str(hex_size_main).replace(".",",")+\
75              '_A_'+str(hex_height_main).replace(".",",")+\
76              '_E_'+str(hex_thickness_main).replace(".",",")+\
77              '_Ball_Drop'
78
79          mesh_size_main = GUI_values_main['mesh_GUI']
80
81          # Create the folder for a specific geometry
82          test_name = 'L = ' + str (hex_size_main) + \
83              ' A = ' + str(hex_height_main) + \
84              ' E = ' + str(hex_thickness_main) + ''
85
86          print('A criar o sistema de pastas para ' \
87              + test_name  + ' (mm). [2/10] ')
88          path_static_main, path_ball_drop_main = Create_Test_Folder(
89              hex_size_main, hex_height_main,
90              hex_thickness_main, dir_path_main)
91
92          # Copy Base file
93          print(test_name + ': A copiar o ficheiro Base. [3/10]')
94          openMdb(pathName = model_path)
95          mymodel = mdb.models['Model-1']
96          myview = session.viewports['Viewport: 1']
```

Figura 242 – Função principal (*main*)(2/7)

```
        # Draw the base Hexagon
        print(test_name + ': A criar o hexagono base. [4/10]')
        sketch_name = Hex(hex_size_main,hex_height_main)

        # Draw the frizes assembly
        print(test_name + ': A desenhar os frisos. [5/10]')
        Create_Particion_Face(hex_height_main,base_thickness_main)

        Create_Hex_Assembly(hex_size_main, hex_lenght_main,
            hex_circle_height_main, hex_thickness_main,
            sketch_name, A,B,C,D,E,F,G,H)

        # Trim the excess from the frizes
        print(test_name + ': A cortar os frisos. [6/10]')
        top_bounding_box_point_1_datum = 'Top_Bounding_Box_P1'
        top_bounding_box_point_2_datum = 'Top_Bounding_Box_P2'
        bottom_bounding_box_point_1_datum = 'Bottom_Bounding_Box_P1'
        bottom_bounding_box_point_2_datum = 'Bottom_Bounding_Box_P2'
        particion_base_set = 'Particion_Base'
        sketch_plane_bottom = 'Sketch_Plane_Bottom'

        Delete_Set(sketch_plane_bottom)

        Trim_Hex_Assembly(top_bounding_box_point_1_datum,
            top_bounding_box_point_2_datum)

        Trim_Hex_Assembly(bottom_bounding_box_point_1_datum,
            bottom_bounding_box_point_2_datum)

        Delete_Set(particion_base_set)

        # Get material data from the database
        print(test_name + ': A ler a base de dados de materiais. [7/10]')
        plasticity_curve_tuple, yield_stress, modulus, \
            poisson, density, limit_strain = \
            Read_True_Stress_Strain_Values(0.01)

        # Create material sections
        print(test_name + ': A atribuir o material e seccoes. [8/10]')
        Atribute_thickness(density, modulus, poisson,
            hex_thickness_main, plasticity_curve_tuple)

        # Create Mesh
        print(test_name + ': A construir a malha. [9/10]')
        Mesh_Part(mesh_size_main)

        # Switch directory to static test
        print(test_name + ': Switching current directory')
        switch_working_directory(cwd_path_main)
```

Figura 243 – Função principal (*main*)(3/7)

```
148         # Create Interactions for the static test
149         print(test_name + ': A realizar o ensaio estatico. [10/10]')
150         Interaction_Static_Test()
151 
152         # Static test
153         final_report, report_counter = Static_test(solve_model,
154             job_name_static_test,
155             path_static_main, path_ball_drop_main, final_report,
156             report_counter, hex_size_main,
157             hex_thickness_main, hex_height_main, yield_stress,
158             limit_strain)
159 
160 
161         # If the user set it to only make the model
162         # The program will only make the first geometry,
163         # And not delete any files
164         if not(solve_model):
165             break
166 
167         # Open a new file
168         Mdb()
169 
170         #Delete files
171         Delete_files(cwd_path_main)
172 
173 
174     # Analise the final report data
175     final_report = Analise_data(final_report,
176         yield_stress, limit_strain)
177 
178     # Sort the final report by weight
179     final_report = sort_report(final_report)
180 
181     # Export the final report data to a file
182     Export_final_report(final_report, dir_path_main)
183 
184     # Create a Hashmap for the Ball Drop
185     ball_drop_list = ball_drop_test_list(final_report)
186 
187     # Internal Variables
188     final_report_ball_drop={}
189     report_counter = 0
190 
191     # Open the base Model
192     model_path = Open_base_model(cwd_path_main, 'Ball Drop')
```

Figura 244 – Função principal (*main*)(4/7)

```
194     # Loop through the testes
195     for test_number in ball_drop_list:
196 
197         # Hexagon parameters
198         hex_size_main = test_number['size']
199         hex_height_main = test_number['height']
200         hex_thickness_main = test_number['thickness']
201         hex_circle_height_main = hex_size_main/np.cos(np.radians(30))
202         hex_lenght_main = hex_size_main*np.tan(np.radians(30))
203 
204         # Job names for each test
205         job_name_ball_drop=''+str(GUI_values_main['job_name_GUI'])+\
206             '_L_'+str(hex_size_main).replace(".",",")+\
207             '_A_'+str(hex_height_main).replace(".",",")+\
208             '_E_'+str(hex_thickness_main).replace(".",",")+\
209             '_Ball_Drop'
210 
211         # Mesh value
212         mesh_size_main = GUI_values_main['mesh_GUI']
213 
214         # Create the test name for the geometry
215         test_name = 'L = ' + str (hex_size_main) + \
216             ' A = ' + str(hex_height_main) + \
217             ' E = ' + str(hex_thickness_main) + ''
218         path_ball_drop_main = test_number['path']
219 
220         # Copy Base file
221         print(test_name + ': A copiar o ficheiro Base. [2/9]')
222         openMdb(pathName = model_path)
223 
224         # Draw the base Hexagon
225         print(test_name + ': A criar o hexagono base. [3/9]')
226         sketch_name = Hex(hex_size_main,hex_height_main)
227 
228         # Draw the frizes assembly
229         print(test_name + ': A desenhar os frisos. [4/9]')
230 
231         Create_Particion_Face(hex_height_main,base_thickness_main)
232 
233         Create_Hex_Assembly(hex_size_main, hex_lenght_main,
234             hex_circle_height_main, hex_thickness_main,
235             sketch_name, A,B,C,D,E,F,G,H)
```

Figura 245 – Função principal (*main*)(5/7)

```
237         # Trim the excess from the frizes
238         print(test_name + ': A cortar os frisos. [5/9]')
239         top_bounding_box_point_1_datum = 'Top_Bounding_Box_P1'
240         top_bounding_box_point_2_datum = 'Top_Bounding_Box_P2'
241         bottom_bounding_box_point_1_datum = 'Bottom_Bounding_Box_P1'
242         bottom_bounding_box_point_2_datum = 'Bottom_Bounding_Box_P2'
243         particion_base_set = 'Particion_Base'
244         sketch_plane_bottom = 'Sketch_Plane_Bottom'
245
246         Delete_Set(sketch_plane_bottom)
247
248         Trim_Hex_Assembly(top_bounding_box_point_1_datum,
249             top_bounding_box_point_2_datum)
250
251         Trim_Hex_Assembly(bottom_bounding_box_point_1_datum,
252             bottom_bounding_box_point_2_datum)
253
254         Delete_Set(particion_base_set)
255
256         # Get material data from the database
257         print(test_name + ': A ler a base de dados de materiais. [6/9]')
258         plasticity_curve_tuple, yield_stress, modulus, \
259             poisson, density, limit_strain = \
260             Read_True_Stress_Strain_Values(16.6)
261
262         # Create material sections
263         print(test_name + ': A atribuir o material e seccoes. [7/9]')
264         Atribute_thickness(density, modulus, poisson,
265             hex_thickness_main, plasticity_curve_tuple)
266
267         # Create Mesh
268         print(test_name + ': A construir a malha. [8/9]')
269         Mesh_Part(mesh_size_main)
270
271         # Switch directory to static test
272         print(test_name + ': Switching current directory')
273         switch_working_directory(cwd_path_main)
274
275         # Create Interactions for the static test
276         Interaction_Ball_Drop_Test()
```

Figura 246 – Função principal (*main*)(6/7)

```
278         # Ball Drop test
279         print(test_name + ': A realizar o ensaio do ball drop. [9/9]')
280         final_report_ball_drop, report_counter, best_result = \
281             Ball_Drop_Test(solve_model,
282             job_name_ball_drop, path_ball_drop_main,
283             final_report_ball_drop,
284             report_counter, hex_size_main,
285             hex_height_main, hex_thickness_main, yield_stress,
286             limit_strain)
287 
288         # If the user set it to only make the model
289         # The program will only make the first geometry,
290         # And not delete any files
291         if best_result=='best':
292             break
293 
294         #Open a new file
295         Mdb()
296 
297         # Delete files
298         Delete_files(cwd_path_main)
299 
300     #Search_txt_files(dir_path_main)
301     combined_dict = Combine_dict(final_report_ball_drop,ball_drop_list)
302     Results(combined_dict, dir_path_main)
```

Figura 247 – Função principal (*main*)(7/7)

3.5.4 Tratamento automático de dados

Após realizar-se o estudo paramétrico, deve proceder-se ao tratamento dos dados obtidos. Ora, como o número de ensaios a realizar-se é imposto pelo utilizador, este pode ser relativamente pequeno (1-5 simulações), ou pode ter uma dimensão muito mais considerável (300-500 simulações), sendo assim não é viável que a realização de tabelas e gráficos dos dados obtidos, seja feita à mão pelo próprio utilizador, e para isto o software possui uma segundo ficheiro Python, que, quando o software principal termina o estudo paramétrico da peça, procura todos os ficheiros de texto com os dados que foram gerados e procede à criação dos respetivos gráficos e tabelas.

Na Figura 248 e na Figura 249 está presente a função principal do ficheiro de análise de dados. Esta função trata de procurar na pasta onde foi realizado o estudo paramétrico todos os ficheiros de texto que foram extraidos durante a sua realização, sendo que quando encontra um ficheiro, dependendo do seu nome, o programa chama então uma função que trata da leitura dos seus dados e de construir os respetivos gráficos e tabelas. Os tipos de ficheiros que o programa pode encontrar são:

- “Valores de tensao.txt” - possui os valores de tensão de cada um dos elementos da respetiva geometria e ensaio (Capítulo 3.5.3.11.2);
- “Deformacao no ponto de aplicação da carga.txt” – possui os valores de deslocamento do impactor, ao longo do tempo, no ensaio estático (Capítulo 3.5.3.11.4);
- “Valores de plasticidade.txt” - possui os valores da deformação plástica de cada um dos elementos da respetiva geometria e ensaio (Capítulo 3.5.3.11.3);
- “Resultados.txt” - aqui estão armazenados os resultados finais de cada geometria, na realização do ensaio estático (Capítulo 3.5.3.12);
- “Resultados_finais.txt” – aqui constam os dados finais das geometrias que passaram no ensaio estático, e procederam para o teste de *Ball Drop*. Aqui as geometrias estão ordenadas por ordem decrescente de desempenho e é também assinalada a melhor geometria para os frisos (Capítulo 3.5.3.13.2);

```
# Search for the text files in the study directory
def Search_txt_files(root_path):
    list_of_files = os.listdir(root_path)

    # Search all files in the folder directory
    for entry in list_of_files:
        if (entry != 'Diretoria de Trabalho') and \
            (entry != 'Resultados.txt') and \
                (entry != 'Resultados_finais.txt'):
            sub_path = os.path.join(root_path,entry)
            list_of_files_2 = os.listdir(sub_path)

            # Search all files in the geometry directory
            for folder_file in list_of_files_2:
                sub_path_2 = os.path.join(sub_path, folder_file)
                list_of_files_3 = os.listdir(sub_path_2)

                # Initiate plot
                if folder_file == 'Teste Estatico':
                    fig, axes = plt.subplots(nrows=3, ncols=1,
                        figsize=(8.3,11.7), dpi=80)
                else:
                    fig, axes = plt.subplots(nrows=2, ncols=1,
                        figsize=(8.3,11.7), dpi=80)

                # Define Plot Title
                fig.suptitle(entry, x= 0.7, y = 0.99, fontsize=12)

                # Search all files in the test directory
                for txt_file in list_of_files_3:

                    # Read the files and plot the data
                    if 'Valores de tensao' in txt_file:
                        file_path = os.path.join(sub_path_2, txt_file,)
                        Read_stress_test_values(file_path,
                            fig, axes, folder_file)
                    elif 'Deformacao no ponto de aplicacao da carga' \
                         in txt_file:
                        file_path = os.path.join(sub_path_2, txt_file)
                        Read_displacement_test_values(file_path,
                            fig, axes)
                    elif 'Valores de plasticidade' in txt_file:
                        file_path = os.path.join(sub_path_2, txt_file,)
                        Read_plastic_strain_test_values(file_path,
                            fig, axes, folder_file
```

Figura 248 – Procura dos ficheiros de resultados (*Search_txt_files*)(1/2)

```
47          # Analise the data from the report card
48          # of the static tests
49          # and split it into blocks of 40 values
50          elif entry == 'Resultados.txt':
51              # Create Report card directory
52              path = os.path.join(root_path,entry)
53
54              # Read Report Card Values
55              final_report = Read_report_card(path)
56
57              # Split the dictionary in multiple pages
58              final_report = Split_dict(final_report)
59
60              # Plot the Report card
61              iterator = 0
62              for n in final_report:
63                  Final_report_card(n, iterator,
64                      root_path, file_name='Resultados')
65                  iterator+=1
66
67          # Analise the data from the report card
68          # and split it into blocks of 40 values
69          elif entry == 'Resultados_finais.txt':
70              # Create Report card directory
71              path = os.path.join(root_path,entry)
72
73              # Read Report Card Values
74              final_report = Read_final_results(path)
75
76              # Split the dictionary in multiple pages
77              final_report = Split_dict(final_report)
78
79              # Plot the Report card
80              iterator = 0
81              for n in final_report:
82                  Final_report_card(n, iterator,
83                      root_path, file_name='Resultados_finais')
84                  iterator+=1
```

Figura 249 – Procura dos ficheiros de resultados (*Search_txt_files*)(2/2)

Cada um dos ficheiros de texto possui duas funções associadas, uma que trata da leitura dos dados armazenados, e outra que trata do *plot* do respetivo gráfico ou tabela. No caso dos valores de tensão, a leitura é feita pela função *Read_stress_test_values*, que lê os valores que constam dentro do ficheiro "Valores de tensao.txt", correlaciona-os com as respetivas variáveis, e depois envia-as para a função *Stress_Graph* para produzir um gráfico de barras, semelhante ao da Figura 250. A produção dos gráficos e tabelas é realizada através da livraria *Matplotlib* do *Python*, que permite gerar gráficos e tabelas através de um *DataSet*, que neste caso são os valores dos ficheiros de texto. Para além dos valores de tensão de um ensaio, existem funções para o tratamento dos dados da deformação plástica, e para o deslocamento do impactor, que são respetivamente a *Read_plastic_strain_test_values*, *Plastic_Strain_Graphs*, *Read_displacement_test_values* e *Plot_nodal_displacement*. Os códigos de cada uma das diferentes funções estão presentes na Figura 256 à Figura 260, enquanto que os respetivos gráficos estão apresentados na Figura 251 e Figura 252.

Como a tensão instalada, a deformação plástica e o deslocamento do impactor são sempre analisados para um determinado ensaio, a função da Figura 248 e Figura 249 agrupa estes três gráficos numa única folha. No caso de um ensaio estático, a folha resultante é igual à da Figura 261, enquanto que no *Ball Drop*, como não se estuda o deslocamento do impactor, este fica igual ao da Figura 262.

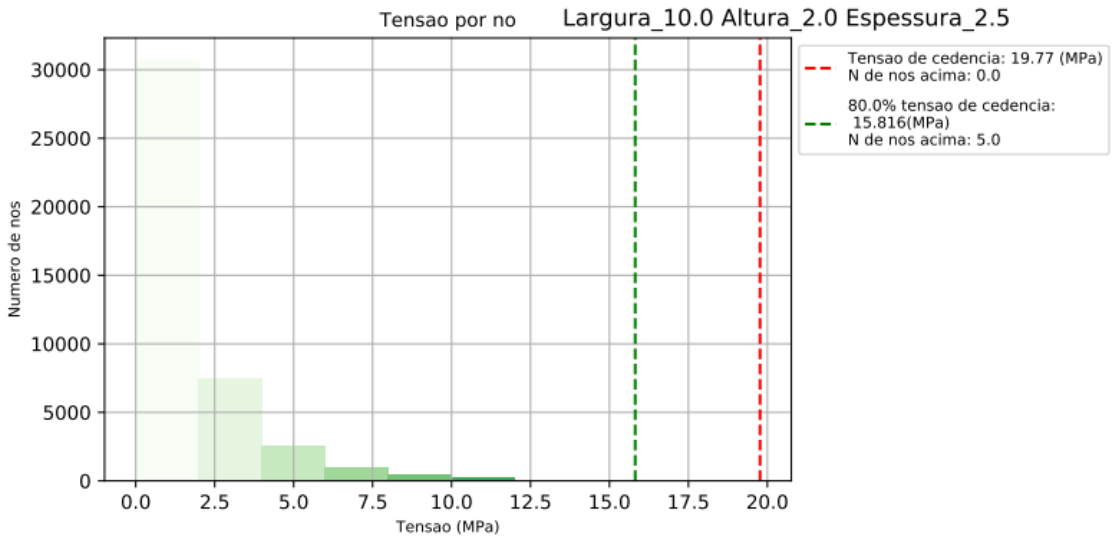


Figura 250 – Demonstração do histograma para a análise da tensão dos elementos da peça

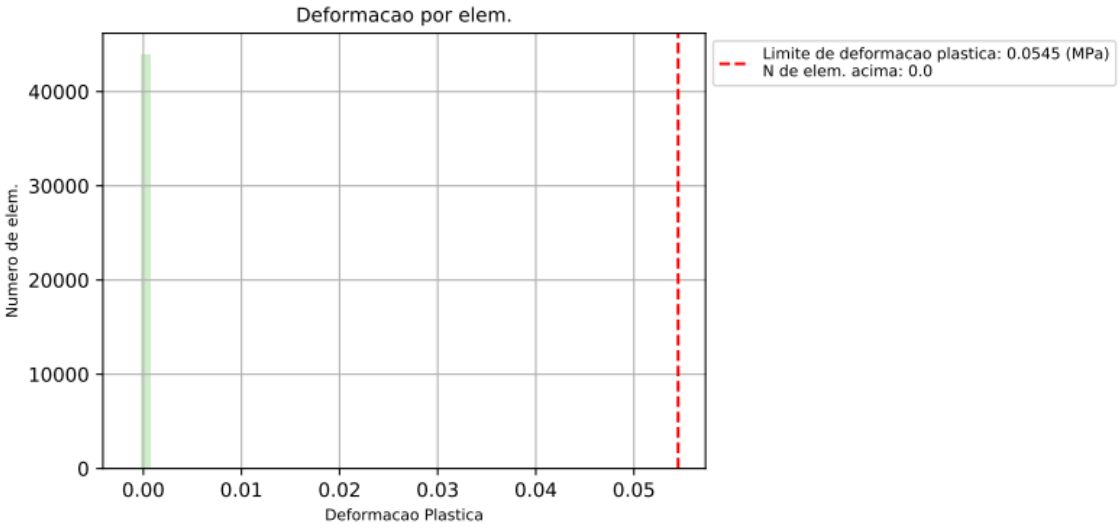


Figura 251 – Demonstração do gráfico de análise do deslocamento do impactor

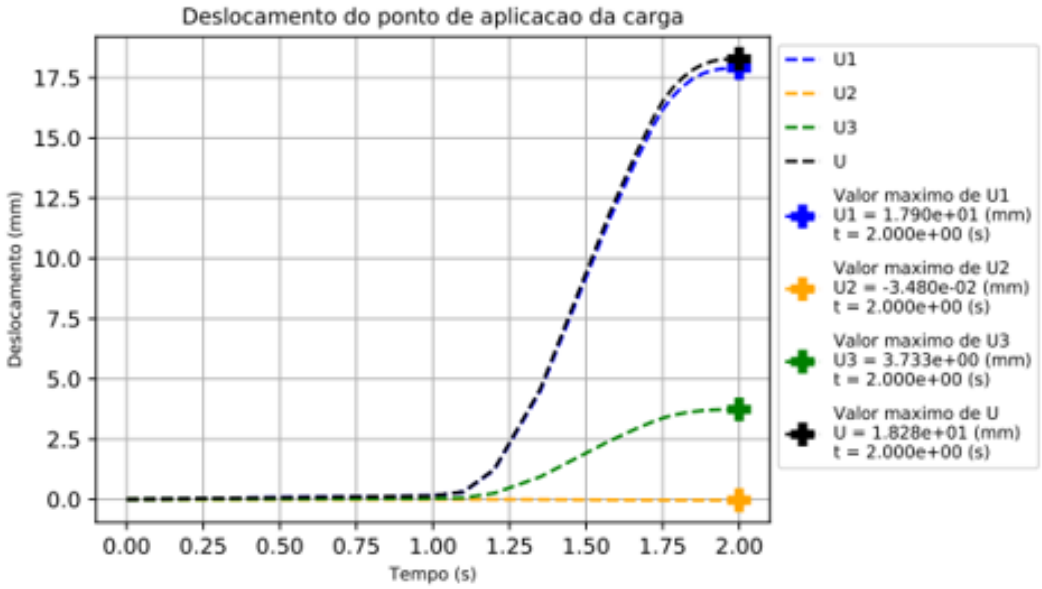


Figura 252 – Demonstração do gráfico de análise do deslocamento do impactor

```
# Read the Stress values data base
def Read_stress_test_values(file, fig, axes, folder_file):

    # Open the text file with the test data
    f=open(file,"r")
    lines=f.readlines()

    # Initiate the lists
    stress_values_list=[]

    # Retrieve each of the variables
    # from the txt file
    mat_yield_stress = (float(lines[0].split('        ')[1]))
    number_of_elem_Outspec = (float(lines[0].split('  ')[2]))
    number_of_elem_Outspec_with_sec = (float(lines[0].split('')[3]))
    safety_factor = (float(lines[0].split('   ')[4]))
    my_path = (lines[0].split('        ')[5])
    size = len(my_path)
    my_path = my_path[:size - 2]

    # Retrieve each line of the stress
    # collum as a value for the stress list
    for x in lines:
        stress_values_list.append(float(x.split('     ')[0]))

    # Close the txt file
    f.close()

    # Plot the data
    Stress_Graphs(stress_values_list, mat_yield_stress,
        number_of_elem_Outspec, number_of_elem_Outspec_with_sec,
        safety_factor, my_path,fig, axes, folder_file
```

Figura 253 – Leitura da base de dados dos valores de tensão do ensaio (*Read_stress_test_values*)

```
# Plot stress values
def Stress_Graphs(stress_values_list, mat_yield_stress, n_nodes_above, \
    n_nodes_above_perc, safety_factor, my_path, fig, axes, folder_file):

    # Bin width
    binwidth = 2

    if folder_file == 'Teste Estatico':
        mysubplot=plt.subplot(311)
    else:
        mysubplot=plt.subplot(211)

    # Get Stress values and number of nodes for each one
    number_of_nodes_hist, stress_values_hist, patches =  \
        plt.hist(stress_values_list,
        bins=np.arange(min(stress_values_list),
        max(stress_values_list) +  \
        binwidth, binwidth))

    # Set label names
    plt.xlabel('Tensao (MPa)', size = 8)
    plt.ylabel('Numero de elementos', size = 8)
    plt.title('Tensao por elemento', size = 10)
    plt.grid(True)

    # Colouring parameters
    yield_fraction = stress_values_hist/mat_yield_stress
    norm_Green = colors.Normalize(vmin=yield_fraction.min(),
        vmax=safety_factor)
    norm_Orange = colors.Normalize(vmin=safety_factor, vmax= 1.0)
    norm_Red = colors.Normalize(vmin=1.0, vmax=yield_fraction.max())

    # Colouring
    for thisfrac, thispatch in zip(yield_fraction, patches):
        if thisfrac <= safety_factor:
          color = plt.cm.Greens(norm_Green(thisfrac))
        elif thisfrac <= 1.0:
          color = plt.cm.YlOrBr(norm_Orange(thisfrac))
        else:
          color = plt.cm.Reds(norm_Red(thisfrac))
        thispatch.set_facecolor(color)
```

Figura 254 – Produção dos gráficos de tensão (*Stress_Graphs*) (1/2)

```
43      # Label of yield stress
44      plt.axvline(x=mat_yield_stress, color='r',
45          label='Tensao de cedencia: ' + \
46          str(mat_yield_stress) + ' (MPa)\nN de elem. acima: ' + \
47          str(n_nodes_above))
48
49      plt.axvline(x=safety_factor*mat_yield_stress, color='g',
50          label= str(safety_factor*100)+'% tensao de cedencia:\n ' + \
51          str(0.8*mat_yield_stress) +'(MPa)\nN de elem. acima: ' + \
52          str(n_nodes_above_perc))
53
54      # Plot legend
55      mysubplot.legend(bbox_to_anchor=(1.0, 1.0), loc='upper left',
56          labelspacing = 1.0, fontsize = 8)
57
58      # Plot and save the graph
59      fig.tight_layout()
60      fig.savefig(my_path + '\Resultados.pdf')
61      plt.close()
```

Figura 255 – Produção dos gráficos de tensão (*Stress_Graphs*) (2/2)

```
# Read the displacement data base
def Read_displacement_test_values(file, fig, axes):

    # Open the text file with the test data
    f=open(file,"r")
    lines=f.readlines()

    # Initiate the lists
    t_list=[]
    um_list=[]
    ux_list=[]
    uy_list=[]
    uz_list=[]

    # Retrieve each of the variables
    # from the txt file
    my_path = (lines[0].split('       ')[5])
    size = len(my_path)
    my_path = my_path[:size - 2]

    # Retrieve the values from the text file
    # as a list
    for x in lines:
        t_list.append(float(x.split(' ')[0]))
        um_list.append(float(x.split('')[1]))
        ux_list.append(float(x.split('')[2]))
        uy_list.append(float(x.split('')[3]))
        uz_list.append(float(x.split('')[4]))

    # Close the txt file
    f.close()

    # Plot the data
    Plot_nodal_displacement(t_list, um_list,
        ux_list,uy_list, uz_list,
        my_path, fig, axes)
```

Figura 256 – Leitura da base de dados dos valores do deslocamento do impactor (*Read_displacement_test_values*)

```
# Plot the displacement
def Plot_nodal_displacement(t, um, ux, uy, uz,
    my_path, fig, axes):

    # Convert list to array
    t = np.asarray(t)
    um = np.asarray(um)
    ux = np.asarray(ux)
    uy = np.asarray(uy)
    uz = np.asarray(uz)


    # Limit deflection
    lim_deflection = 2.0

    # Subplot number
    myplot = plt.subplot(313)
    ax = axes[2]

    # Create Graphs
    # Change Graph Line Representattion (Global)
    mpl.rcParams['lines.linestyle'] = '--'

    # Define the subplot matrix, number of rows,
    # collums and the size of the figure
    myplot.plot(t, ux, color = 'b', label='U1')
    myplot.plot(t, uy, color = 'orange', label='U2')
    myplot.plot(t, uz, color = 'g', label='U3')
    myplot.plot(t, um, color = 'k', label='U')
    plt.xlabel('Tempo (s)', size = 8)
    plt.ylabel('Deslocamento (mm)', size = 8)
    plt.title('Deslocamento do ponto de aplicacao da carga', size = 10)
    plt.grid(True)

    # Max values
    # Displacement - U1-max
    if np.amax(ux)*-1 < np.amin(ux):
        max_ux = np.amax(ux)
    else :
        max_ux = np.amin(ux)
    t_max_ux = t[np.where(ux == max_ux)]

    # Plot displacement - U1-max
    myplot.plot(t_max_ux[0], max_ux, color = 'b',
        label = 'Valor maximo de U1\nU1 = '+ \
        "{:.3e}".format(max_ux) + ' (mm)\nt = '\
        + "{:.3e}".format(t_max_ux[0]) + ' (s)', \
        marker = 'P', ms = 10)
```

Figura 257 – Produção dos gráficos do deslocamento do impactor (*Plot_nodal_displacement*) (1/2)

```
    # Plot displacement - U2-max
    myplot.plot(t_max_uy[0], max_uy, color = 'orange',
        label = 'Valor maximo de U2\nU2 = ' + \
        "{:.3e}".format(max_uy) + ' (mm)\nt = ' + \
        "{:.3e}".format(t_max_uy[0]) + ' (s)',
        marker = 'P', ms = 10)


    # Displacement - U3-max
    if np.amax(uz)*-1 < np.amin(uz):
        max_uz = np.amax(uz)
    else :
        max_uz = np.amin(uz)
    t_max_uz = t[np.where(uz == max_uz)]

    # Plot displacement - U3-max
    myplot.plot(t_max_uz[0], max_uz, color = 'g',
        label = 'Valor maximo de U3\nU3 = ' +
        "{:.3e}".format(max_uz) + ' (mm)\nt = ' + \
        "{:.3e}".format(t_max_uz[0]) + ' (s)' ,
        marker = 'P', ms = 10)


    # Displacement - U-max
    if np.amax(um)*-1 < np.amin(um):
        max_um = np.amax(um)
    else :
        max_um = np.amin(um)
    t_max_um = t[np.where(um == max_um)]

    # Plot displacement - U-max
    myplot.plot(t_max_um[0], max_um, color = 'k',
        label = 'Valor maximo de U\nU = ' + \
        "{:.3e}".format(max_um)+' (mm)\nt = ' + \
        "{:.3e}".format(t_max_um[0]) + ' (s)',
        marker = 'P', ms = 10)


    # Plot the Graph and Legend
    myplot.legend(bbox_to_anchor=(1.0, 1), loc='upper left',
        labelspacing = 1.0, fontsize = 8)
    fig.tight_layout()
```

Figura 258 – Produção dos gráficos do deslocamento do impactor (*Plot_nodal_displacement*) (2/2)

```
# Read the Plastic Strain values data base
def Read_plastic_strain_test_values(file, fig, axes, folder_file):

    # Open the text file with the test data
    f=open(file,"r")
    lines=f.readlines()

    # Initiate the lists
    strain_values_list=[]

    # Retrieve each of the variables
    # from the txt file
    mat_limit_strain = (float(lines[0].split('        ')[1]))
    number_of_elem_Outspec = (float(lines[0].split('  ')[2]))
    my_path = (lines[0].split('        ')[3])
    size = len(my_path)
    my_path = my_path[:size - 2]

    # Retrieve each line of the stress
    # collum as a value for the stress list
    for x in lines:
        strain_values_list.append(float(x.split('     ')[0]))

    # Close the txt file
    f.close()

    # Plot the data
    Plastic_Strain_Graphs(strain_values_list,
        mat_limit_strain, number_of_elem_Outspec,
         my_path,fig, axes, folder_file)
```

Figura 259 – Leitura da base de dados dos valores de deformação plástica (*Read_displacement_plastic_strain_test_values*)

```
# Plot the plastic strain values
def Plastic_Strain_Graphs(strain_values_list,
    mat_limit_strain, n_elements_above,
    my_path, fig, axes, folder_file):

    # Bin width
    binwidth = 0.001
    if folder_file == 'Teste Estatico':
        mysubplot=plt.subplot(312)
    else:
        mysubplot=plt.subplot(212)

    # Get Stress values and number of nodes for each one
    number_of_elem_hist, stress_values_hist, patches =  \
        plt.hist(strain_values_list,
        bins=np.arange(min(strain_values_list),
        max(strain_values_list) + binwidth, binwidth))

    # Set label names
    plt.xlabel('Deformacao Plastica', size = 8)
    plt.ylabel('Numero de elem.', size = 8)
    plt.title('Deformacao por elem.', size = 10)
    plt.grid(True)

    # Colouring parameters
    yield_fraction = stress_values_hist/mat_limit_strain
    norm_Green = colors.Normalize(vmin=yield_fraction.min(), vmax=1)
    norm_Red = colors.Normalize(vmin=1, vmax=yield_fraction.max())

    # Colouring
    for thisfrac, thispatch in zip(yield_fraction, patches):
        if thisfrac <= 1:
          color = plt.cm.Greens(norm_Green(thisfrac+0.2))
        else:
          color = plt.cm.Reds(norm_Red(thisfrac+0.2))
        thispatch.set_facecolor(color)

       # Limit Strain line
    plt.axvline(x=mat_limit_strain, color='r',
        label='Limite de deformacao plastica: ' + \
        str(mat_limit_strain) + ' (MPa)\nN de elem. acima: ' + \
        str(n_elements_above))

    # Plot legend
    mysubplot.legend(bbox_to_anchor=(1.0, 1.0), loc='upper left',
        labelspacing = 1.0, fontsize = 8)
```

Figura 260– Produção dos gráficos da deformação plástica (*Plostic_Strain_Graphs*)

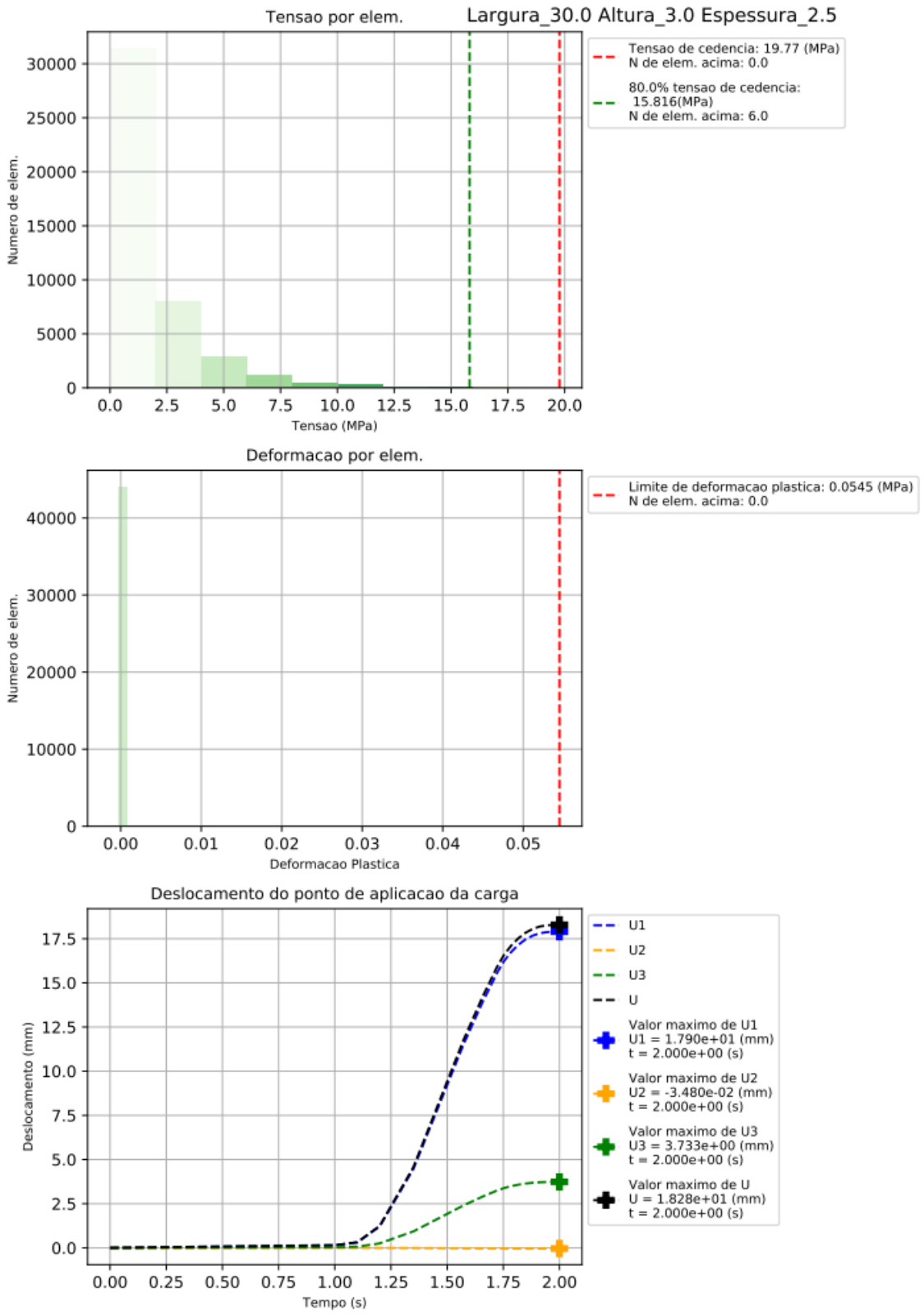


Figura 261 – Folha de resultados, para o ensaio estático

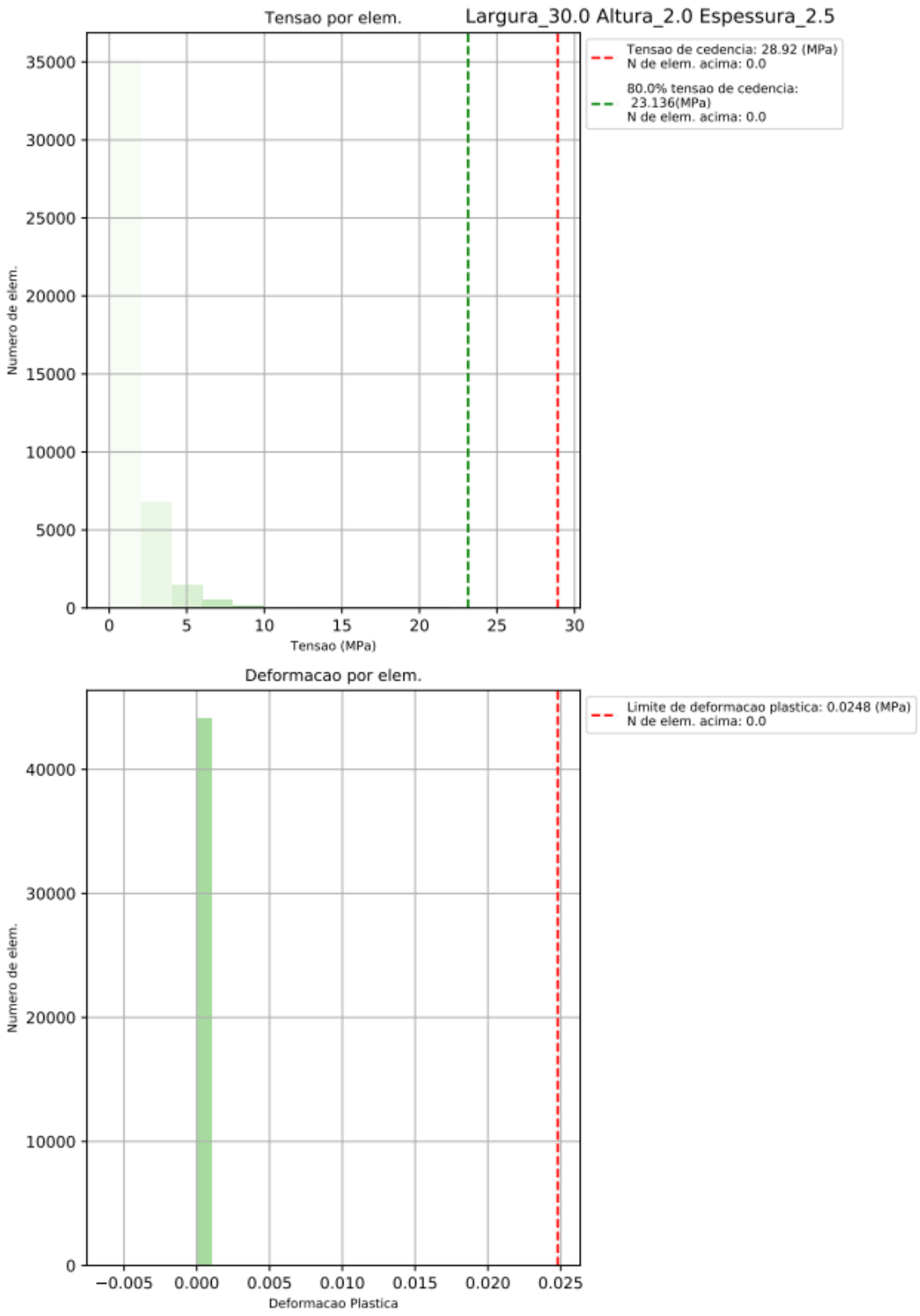


Figura 262 – Folha de resultados para o ensaio de *Ball Drop*

Para além do tratamento dos dados dos elementos de cada um dos modelos o programa também cria uma tabela de resumo, no qual constam apenas os valores mais importantes de cada ensaio. Essencialmente o programa gera duas tabelas de resumo:

- "Resultados" - Constam os resultados de todos os ensaios estáticos, ordenados por ordem crescente do peso da peça analisada, por sua vez estão também assinaladas todas as geometrias que levaram a que a peça passasse no ensaio estático;
- "Resultados Finais" - Aqui já só constam os dados das geometrias que passaram nos ensaios estáticos, visto que foram estas as que poderiam ser sujeitas ao ensaio do *Ball Drop*. Para cada uma das geometrias constam os dados obtidos no ensaio estático, e para algumas, ou todas elas, os dados obtidos durante o *Ball Drop*. Só constam os dados do *Ball Drop*, para aquelas geometrias em que foram realizados o ensaio, pois o programa pára, assim que descobre a geometria ideal, que na tabela se encontra assinalada a amarelo;

Os dados destas tabelas estão armazenados nos ficheiros "Resultados.txt" e "Resultados_finais.txt", e estes são lidos através da função *Read_report_card*, e antes de serem levados para a função responsável pela criação das tabelas (*Final_Report_Card*), a função *Split_dict* divide o *Hash Map* onde são armazenados os dados, em parcelas de 40 geometrias, permitindo assim que, caso a tabela seja demasiado longa, esta possa ser dividida em múltiplas páginas com um máximo de 40 geometrias cada. Um exemplo de cada uma delas está representado na Figura 263 e Figura 264.

Resumo dos resultados do estudo dos frisos

Largura(mm)	Altura(mm)	Esp.(mm)	Teste	Tens.(MPa)	Def.(mm)	Def.Pl	Peso(Ton)
30.0	2.0	2.5	Estatico	1.699e+01	1.856e+01	1.136e-02	1.082e-03
30.0	3.0	2.5	Estatico	1.684e+01	1.828e+01	1.099e-02	1.087e-03
30.0	2.0	3.5	Estatico	1.695e+01	1.843e+01	1.127e-02	1.089e-03
30.0	4.0	2.5	Estatico	1.675e+01	1.804e+01	1.076e-02	1.093e-03
25.0	2.0	3.5	Estatico	1.694e+01	1.831e+01	1.126e-02	1.095e-03
15.0	2.0	2.5	Estatico	1.698e+01	1.844e+01	1.136e-02	1.095e-03
20.0	3.0	2.5	Estatico	1.654e+01	1.773e+01	1.023e-02	1.097e-03
30.0	5.0	2.5	Estatico	1.663e+01	1.779e+01	1.046e-02	1.098e-03
25.0	4.0	2.5	Estatico	1.669e+01	1.791e+01	1.061e-02	1.099e-03
20.0	2.0	3.5	Estatico	1.657e+01	1.779e+01	1.030e-02	1.100e-03
30.0	6.0	2.5	Estatico	1.642e+01	1.741e+01	1.053e-02	1.104e-03
20.0	4.0	2.5	Estatico	1.651e+01	1.761e+01	1.016e-02	1.105e-03
25.0	3.0	3.5	Estatico	1.687e+01	1.804e+01	1.109e-02	1.105e-03
15.0	3.0	2.5	Estatico	1.686e+01	1.819e+01	1.106e-02	1.105e-03
25.0	5.0	2.5	Estatico	1.670e+01	1.781e+01	1.064e-02	1.106e-03
15.0	2.0	3.5	Estatico	1.686e+01	1.819e+01	1.104e-02	1.109e-03
20.0	3.0	3.5	Estatico	1.648e+01	1.754e+01	1.007e-02	1.111e-03
20.0	5.0	2.5	Estatico	1.648e+01	1.754e+01	1.011e-02	1.113e-03
30.0	5.0	3.5	Estatico	1.635e+01	1.727e+01	9.763e-03	1.113e-03
25.0	6.0	2.5	Estatico	1.668e+01	1.767e+01	1.060e-02	1.113e-03
25.0	4.0	3.5	Estatico	1.676e+01	1.777e+01	1.082e-02	1.115e-03
10.0	2.0	2.5	Estatico	1.681e+01	1.779e+01	1.095e-02	1.118e-03
20.0	6.0	2.5	Estatico	1.645e+01	1.747e+01	1.001e-02	1.120e-03
30.0	6.0	3.5	Estatico	1.622e+01	1.695e+01	9.545e-03	1.121e-03
20.0	4.0	3.5	Estatico	1.644e+01	1.742e+01	9.984e-03	1.122e-03
15.0	3.0	3.5	Estatico	1.676e+01	1.793e+01	1.079e-02	1.122e-03
20.0	5.0	3.5	Estatico	1.644e+01	1.733e+01	9.987e-03	1.133e-03
15.0	6.0	2.5	Estatico	1.691e+01	1.799e+01	1.118e-02	1.134e-03
10.0	3.0	2.5	Estatico	1.671e+01	1.738e+01	1.070e-02	1.134e-03
15.0	4.0	3.5	Estatico	1.671e+01	1.777e+01	1.067e-02	1.136e-03
10.0	2.0	3.5	Estatico	1.677e+01	1.755e+01	1.084e-02	1.141e-03
20.0	6.0	3.5	Estatico	1.644e+01	1.719e+01	1.002e-02	1.144e-03
15.0	6.0	3.5	Estatico	1.672e+01	1.759e+01	1.071e-02	1.162e-03
10.0	5.0	2.5	Estatico	1.630e+01	1.648e+01	9.677e-03	1.166e-03
10.0	6.0	2.5	Estatico	1.614e+01	1.609e+01	9.290e-03	1.182e-03
10.0	4.0	3.5	Estatico	1.626e+01	1.644e+01	9.583e-03	1.185e-03
10.0	5.0	3.5	Estatico	1.620e+01	1.616e+01	9.450e-03	1.208e-03
10.0	6.0	3.5	Estatico	1.595e+01	1.574e+01	8.804e-03	1.230e-03

Passaram nos ensaios Geometria mais leve e que passou os ensaios

Figura 263 – Tabela dos resultados do ensaio estático

Resumo dos resultados do estudo dos frisos

Largura(mm)	Altura(mm)	Esp.(mm)	Teste	Tens.(MPa)	Def.(mm)	Def.Pl	Peso(Ton)
30.0	2.0	2.5	Estatico	1.699e+01	1.856e+01	1.136e-02	1.082e-03
30.0	2.0	2.5	Ball Drop	2.180e+01	NC	1.025e-02	1.082e-03
30.0	3.0	2.5	Estatico	1.684e+01	1.828e+01	1.099e-02	1.087e-03
30.0	2.0	3.5	Estatico	1.695e+01	1.843e+01	1.127e-02	1.089e-03
30.0	4.0	2.5	Estatico	1.675e+01	1.804e+01	1.076e-02	1.093e-03
25.0	2.0	3.5	Estatico	1.694e+01	1.831e+01	1.126e-02	1.095e-03
15.0	2.0	2.5	Estatico	1.698e+01	1.844e+01	1.136e-02	1.095e-03
20.0	3.0	2.5	Estatico	1.654e+01	1.773e+01	1.023e-02	1.097e-03
30.0	5.0	2.5	Estatico	1.663e+01	1.779e+01	1.046e-02	1.098e-03
25.0	4.0	2.5	Estatico	1.669e+01	1.791e+01	1.061e-02	1.099e-03
20.0	2.0	3.5	Estatico	1.657e+01	1.779e+01	1.030e-02	1.100e-03
30.0	6.0	2.5	Estatico	1.642e+01	1.741e+01	1.053e-02	1.104e-03
20.0	4.0	2.5	Estatico	1.651e+01	1.761e+01	1.016e-02	1.105e-03
25.0	3.0	3.5	Estatico	1.687e+01	1.804e+01	1.109e-02	1.105e-03
15.0	3.0	2.5	Estatico	1.686e+01	1.819e+01	1.106e-02	1.105e-03
25.0	5.0	2.5	Estatico	1.670e+01	1.781e+01	1.064e-02	1.106e-03
15.0	2.0	3.5	Estatico	1.686e+01	1.819e+01	1.104e-02	1.109e-03
20.0	3.0	3.5	Estatico	1.648e+01	1.754e+01	1.007e-02	1.111e-03
20.0	5.0	2.5	Estatico	1.648e+01	1.754e+01	1.011e-02	1.113e-03
30.0	5.0	3.5	Estatico	1.635e+01	1.727e+01	9.763e-03	1.113e-03
25.0	6.0	2.5	Estatico	1.668e+01	1.767e+01	1.060e-02	1.113e-03
25.0	4.0	3.5	Estatico	1.676e+01	1.777e+01	1.082e-02	1.115e-03
10.0	2.0	2.5	Estatico	1.681e+01	1.779e+01	1.095e-02	1.118e-03
20.0	6.0	2.5	Estatico	1.645e+01	1.747e+01	1.001e-02	1.120e-03
30.0	6.0	3.5	Estatico	1.622e+01	1.695e+01	9.545e-03	1.121e-03
20.0	4.0	3.5	Estatico	1.644e+01	1.742e+01	9.984e-03	1.122e-03
15.0	3.0	3.5	Estatico	1.676e+01	1.793e+01	1.079e-02	1.122e-03
20.0	5.0	3.5	Estatico	1.644e+01	1.733e+01	9.987e-03	1.133e-03
15.0	6.0	2.5	Estatico	1.691e+01	1.799e+01	1.118e-02	1.134e-03
10.0	3.0	2.5	Estatico	1.671e+01	1.738e+01	1.070e-02	1.134e-03
15.0	4.0	3.5	Estatico	1.671e+01	1.777e+01	1.067e-02	1.136e-03
10.0	2.0	3.5	Estatico	1.677e+01	1.755e+01	1.084e-02	1.141e-03
20.0	6.0	3.5	Estatico	1.644e+01	1.719e+01	1.002e-02	1.144e-03
15.0	6.0	3.5	Estatico	1.672e+01	1.759e+01	1.071e-02	1.162e-03
10.0	5.0	2.5	Estatico	1.630e+01	1.648e+01	9.677e-03	1.166e-03
10.0	6.0	2.5	Estatico	1.614e+01	1.609e+01	9.290e-03	1.182e-03
10.0	4.0	3.5	Estatico	1.626e+01	1.644e+01	9.583e-03	1.185e-03
10.0	5.0	3.5	Estatico	1.620e+01	1.616e+01	9.450e-03	1.208e-03
10.0	6.0	3.5	Estatico	1.595e+01	1.574e+01	8.804e-03	1.230e-03

Passaram nos ensaios Geometria mais leve e que passou os ensaios

Figura 264 – Tabela dos resultados do ensaio estudo completo

```
# Read the report_card database
def Read_report_card(file):

    # Open the text file with the test data
    f=open(file,"r")
    lines=f.readlines()

    # Retrieve each of the variables
    # from the txt file
    my_path = (lines[0].split('       ')[7])
    size = len(my_path)
    my_path = my_path[:size - 2]

    # retrieve the values of the file as a dictionary
    final_report={}
    counter = 1
    for x in lines:
        final_report[counter] = {'size':x.split('     ')[0],
            'height':x.split(' ')[1],
            'thickness':x.split('     ')[2],
            'test':x.split('   ')[3],
            'stress':float(x.split('  ')[4]),
            'displacement':float(x.split('    ')[5]),
            'plastic_strain':float(x.split('  ')[6]),
            'weight':float(x.split('  ')[7]),
            'status':x.split(' ')[8]}
        counter +=1

    return final_report
```

Figura 265 – Leitura dos dados dos resultados finais (*Read_report_card*)

```
# Split the report-card HashMap
# into pages with a max of 40 values
def Split_dict(data, SIZE=40):
    itd = iter(data)
    for i in range(0, len(data), SIZE):
        yield {k:data[k] for k in islice(itd, SIZE)}
```

Figura 266 – Divisão da tabela em páginas com um máximo de 40 valores (*Split_dict*)

```
# Plot the table with the final results
def Final_report_card(final_report, iterator, my_path, file_name):

    # Data management
    size=[]
    height=[]
    thickness=[]
    test=[]
    stress=[]
    displacement=[]
    plastic_strain=[]
    weight=[]
    for x in final_report:
        size.append(final_report[x]['size'])
        height.append(final_report[x]['height'])
        thickness.append(final_report[x]['thickness'])
        test.append(final_report[x]['test'])
        stress.append("{:.3e}".format(final_report[x]['stress']))
        weight.append("{:.3e}".format(final_report[x]['weight']))
        plastic_strain.append("{:.3e}"\
            .format(final_report[x]['plastic_strain']))

        if final_report[x]['displacement']=='NC':
            displacement.append(final_report[x]['displacement'])
        else:
            displacement.append("{:.3e}"\
                .format(final_report[x]['displacement']))


    # Plot Table
    fig, ax =plt.subplots(1, 1, figsize=(8.3,11.7), dpi=80)
    fig.suptitle('Resumo dos resultados do estudo dos frisos',
        fontsize=14)
    data=[size,height,thickness,test,stress,
        displacement,plastic_strain,weight]
    table_data = np.transpose(data)
    column_labels=['Largura(mm)','Altura(mm)','Esp.(mm)',
        'Teste','Tens.(MPa)','Def.(mm)',
        'Def.Pl', 'Peso(Ton)']
    ax.axis('tight')
    ax.axis('off')
    my_table=ax.table(cellText=table_data,
        colLabels=column_labels,
        loc="center", cellLoc = "center")
```

Figura 267 – Plot da tabela dos resultados finais do estudo (*Final_report_card*)(1/3)

```
46      # Coloring of the geometries that passed the test,
47      # and the lightest one out of them
48      for x in final_report:
49          status = final_report[x]['status']
50          print(status[:-2])
51          if final_report[x]['status'] == 'passed':
52              face_color = "#daf7d4"
53              my_table[x-iterator*40,0].set_facecolor(face_color)
54              my_table[x-iterator*40,1].set_facecolor(face_color)
55              my_table[x-iterator*40,2].set_facecolor(face_color)
56              my_table[x-iterator*40,3].set_facecolor(face_color)
57              my_table[x-iterator*40,4].set_facecolor(face_color)
58              my_table[x-iterator*40,5].set_facecolor(face_color)
59              my_table[x-iterator*40,6].set_facecolor(face_color)
60              my_table[x-iterator*40,7].set_facecolor(face_color)
61
62          elif final_report[x]['status'] == 'best':
63              face_color = "#ffe700"
64              my_table[x-iterator*40,0].set_facecolor(face_color)
65              my_table[x-iterator*40,1].set_facecolor(face_color)
66              my_table[x-iterator*40,2].set_facecolor(face_color)
67              my_table[x-iterator*40,3].set_facecolor(face_color)
68              my_table[x-iterator*40,4].set_facecolor(face_color)
69              my_table[x-iterator*40,5].set_facecolor(face_color)
70              my_table[x-iterator*40,6].set_facecolor(face_color)
71              my_table[x-iterator*40,7].set_facecolor(face_color)
72
73          elif status[:-2] == 'best':
74              face_color = "#ffe700"
75              my_table[x-iterator*40,0].set_facecolor(face_color)
76              my_table[x-iterator*40,1].set_facecolor(face_color)
77              my_table[x-iterator*40,2].set_facecolor(face_color)
78              my_table[x-iterator*40,3].set_facecolor(face_color)
79              my_table[x-iterator*40,4].set_facecolor(face_color)
80              my_table[x-iterator*40,5].set_facecolor(face_color)
81              my_table[x-iterator*40,6].set_facecolor(face_color)
82              my_table[x-iterator*40,7].set_facecolor(face_color)
83
84
85              my_table[x-iterator*40-1,0].set_facecolor(face_color)
86              my_table[x-iterator*40-1,1].set_facecolor(face_color)
87              my_table[x-iterator*40-1,2].set_facecolor(face_color)
88              my_table[x-iterator*40-1,3].set_facecolor(face_color)
89              my_table[x-iterator*40-1,4].set_facecolor(face_color)
90              my_table[x-iterator*40-1,5].set_facecolor(face_color)
91              my_table[x-iterator*40-1,6].set_facecolor(face_color)
92              my_table[x-iterator*40-1,7].set_facecolor(face_color)
```

Figura 268 – Plot da tabela dos resultados finais do estudo (*Final_report_card*)(1/3)

```
    # Font size and scaling
    my_table.auto_set_font_size(False)
    my_table.set_fontsize(10)
    my_table.scale(1, 1)
    fig.tight_layout()

    # Legend handels and labels
    green_patch = mpatches.Patch(color="#daf7d4",
        label='Passaram nos ensaios')

    yellow_patch = mpatches.Patch(color="#ffe700",
        label='Geometria mais leve e que passou os ensaios')

    # Plot Legend
    plt.legend(handles = [green_patch, yellow_patch],
        bbox_to_anchor=(0.9, 0.1), loc='upper right',
        ncol= 2, labelspacing = 1.0, fontsize = 10)

    # Save the table
    fig.savefig(my_path + '\\'+ str(file_name) + \
        '_' + str(iterator+1) + '.pdf')
```

Figura 269 – *Plot* da tabela dos resultados finais do estudo (*Final_report_card*)(1/3)

3.5.5 Aplicabilidade do software

Como a Simoldes Plásticos® pretende que o software desenvolvido neste projeto possa ser aplicado a diversos tipos de peças diferentes, para além de este ter sido aplicado à peça do caso de estudo, foram também testados outros componentes com condições de fronteira, geometrias e zonas de frisos diferentes da pretendida.

- Uma das peças onde foi aplicado o software foi numa placa longa com 900x450 mm, em que duas das suas extremidades possuem extrusões encastradas, e a área de aplicação dos frisos se reduziu um a retângulo reto de 800 mm de largura e 100 mm de altura (Figura 270). Para este componente os parâmetros ideais para os frisos foram:
    - Largura-76.0mm Altura-3.0mm Espessura-2.5mm.

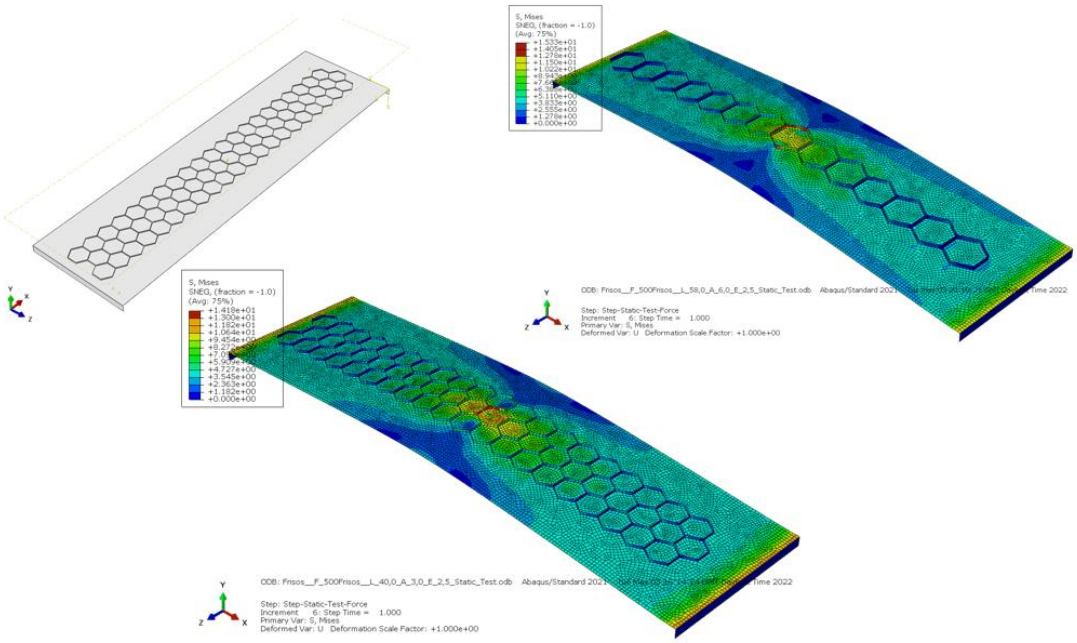


Figura 270 – Programa aplicado a uma placa dobrada nos pontos de encastramento

- Também foi aplicado a uma placa de pequenas dimensões (500x200 mm), encastrada em todo o seu bordo, e uma área de frisos retangular (450x100 mm) (Figura 271). Neste estudo obteve-se a geometria de:
    - Largura-66.0mm Altura-1.0mm Espessura-1.0mm.

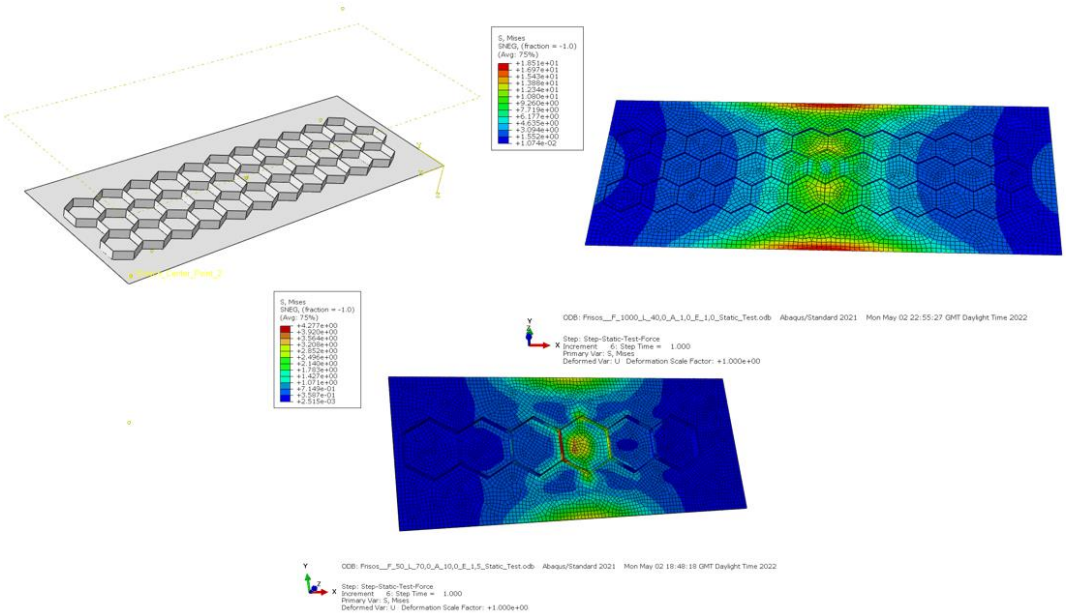


Figura 271 – Programa aplicado a uma placa de dimensões reduzidas, e encastrada a toda a sua volta

- De maneira a ter-se uma ideia de como os frisos influenciam as propriedades mecânicas do componente, foi também testada a peça da Figura 272, que possui uma curvatura da face onde se aplicam os frisos semelhante à da peça do caso de estudo da dissertação. Por sua vez as dimensões da zona de aplicação dos frisos foram as mesmas, mas aqui, a carga foi aplicada na zona com maior densidade de frisos, como demonstrado na Figura 272. Aqui as dimensões obtidas para os frisos foram as seguintes:
    - Largura-20mm Altura-2mm Espessura-2.5mm.

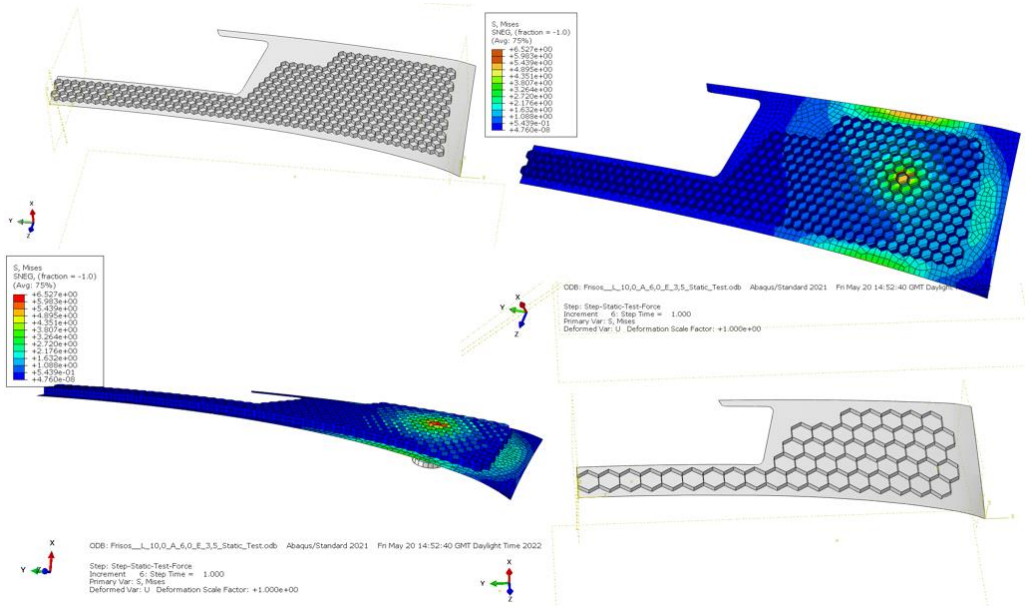


Figura 272 – Programa aplicado a um componente com uma face irregular de aplicação de frisos

- Por fim foi testada a peça a peça do caso de estudo da Simoldes® (Figura 273), onde todo o processo de análise, e resultados obtidos, encontra-se mais detalhado no capítulo 3.5.6.

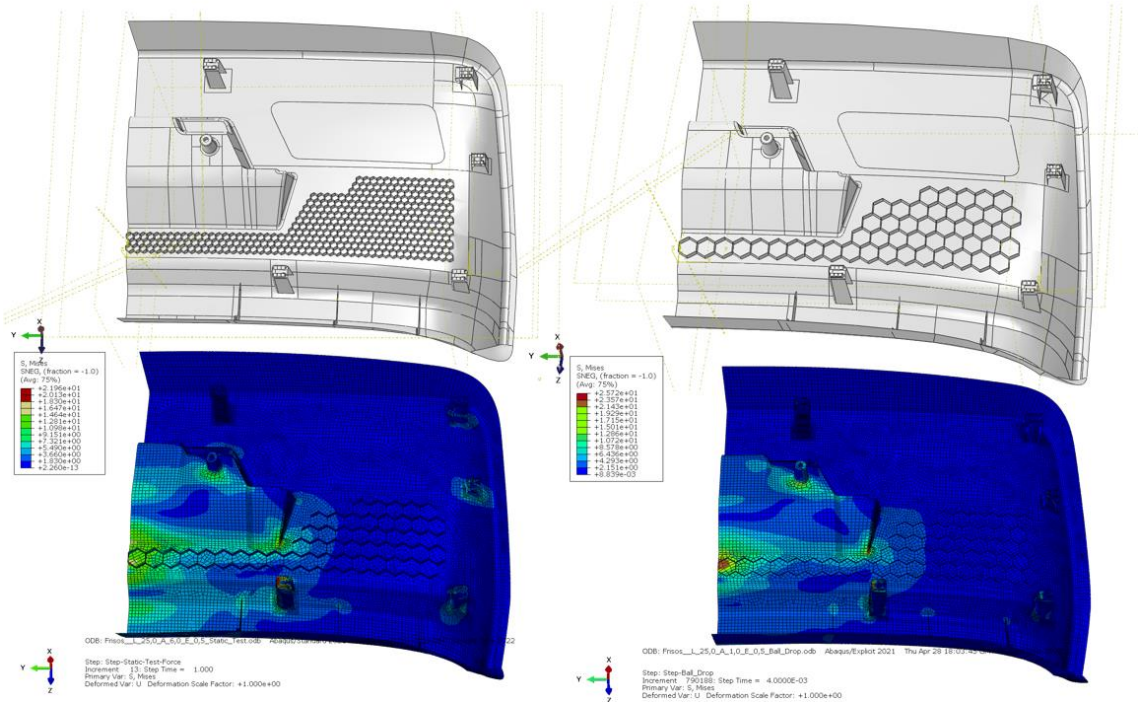


Figura 273 – Programa aplicado à peça da Simoldes®

Com estas análises foi também capaz de se ter uma primeira ideia de como cada um dos parâmetros dos frisos afeta os resultados da peça. Observando-se os resultados da tabela de resultados da peça da Figura 272 (Tabela 9), pode-se ver que as três variáveis do estudo melhoram o desempenho mecânico da peça, e analisando melhor os gráficos da Figura 274 à Figura 282, pode-se ainda inferir que dentro das três, a largura dos frisos é o parâmetro que mais influencia a tensão instalada e a deformação plástica da peça, sendo que a altura dos frisos é a que mais afeta o deslocamento do impactor durante o ensaio. A espessura dos frisos também tem uma contribuição positiva para a melhoria das propriedades mecânicas das peças, mas não é tão acentuado como o efeito das outras duas variáveis. É importante notar que no referido caso de estudo, permitiu-se que fosse feito o ensaio do *Ball Drop* para todas as geometrias, em vez de permitir que este parasse quando encontrasse a geometria ideal.

Tabela 9 – Resultados da peça exemplo

Largura (mm)	Altura (mm)	Espessura (mm)	Teste	Tensão v. Mises (MPa)	Deslocamento do impactor (mm)	Deformação Plástica	Massa (Kg)
30	2	2.5	Estático	17.769	12.126	1.005E-02	0.164
30	2	2.5	*Ball Drop*	29.859	NC	3.233E-02	0.164
20	2	2.5	Estático	14.059	12.095	4.179E-03	0.172
20	2	2.5	*Ball Drop*	28.057	NC	1.206E-02	0.172
30	4	2.5	Estático	16.949	11.796	7.860E-03	0.176
30	4	2.5	*Ball Drop*	27.744	NC	2.890E-02	0.176
20	2	3.5	Estático	13.221	12.003	3.563E-03	0.183
20	2	3.5	*Ball Drop*	28.088	NC	9.026E-03	0.183
30	6	2.5	Estático	15.668	11.303	5.551E-03	0.187
30	6	2.5	*Ball Drop*	29.843	NC	2.336E-02	0.187
20	4	2.5	Estático	12.812	11.693	3.988E-03	0.188
20	4	2.5	*Ball Drop*	26.582	NC	6.707E-03	0.188
30	4	3.5	Estático	14.998	11.552	4.913E-03	0.188
30	4	3.5	*Ball Drop*	29.736	NC	2.114E-02	0.188
10	2	2.5	Estático	12.585	11.648	2.175E-03	0.201
10	2	2.5	*Ball Drop*	25.083	NC	4.231E-03	0.201
20	6	2.5	Estático	13.789	11.120	5.667E-03	0.203
20	6	2.5	*Ball Drop*	26.898	NC	1.241E-02	0.203
30	6	3.5	Estático	13.175	10.954	4.940E-03	0.204

30	6	3.5	*Ball Drop*	29.904	NC	2.094E-02	0.204
20	4	3.5	Estático	11.882	11.487	2.616E-03	0.205
20	4	3.5	*Ball Drop*	25.021	NC	3.769E-03	0.205
10	2	3.5	Estático	12.201	11.497	1.203E-03	0.223
10	2	3.5	*Ball Drop*	24.215	NC	2.849E-03	0.223
20	6	3.5	Estático	12.145	10.848	3.313E-03	0.227
20	6	3.5	*Ball Drop*	23.540	NC	8.035E-03	0.227
10	4	2.5	Estático	10.957	11.025	0.000E+00	0.233
10	4	2.5	*Ball Drop*	23.777	NC	3.003E-03	0.233
10	6	2.5	Estático	8.784	10.372	0.000E+00	0.265
10	6	2.5	*Ball Drop*	23.710	NC	3.905E-03	0.265
10	4	3.5	Estático	10.466	10.829	0.000E+00	0.268
10	4	3.5	*Ball Drop*	22.667	NC	1.162E-03	0.268
10	6	3.5	Estático	6.527	10.207	0.000E+00	0.313
10	6	3.5	*Ball Drop*	22.122	NC	1.878E-03	0.313

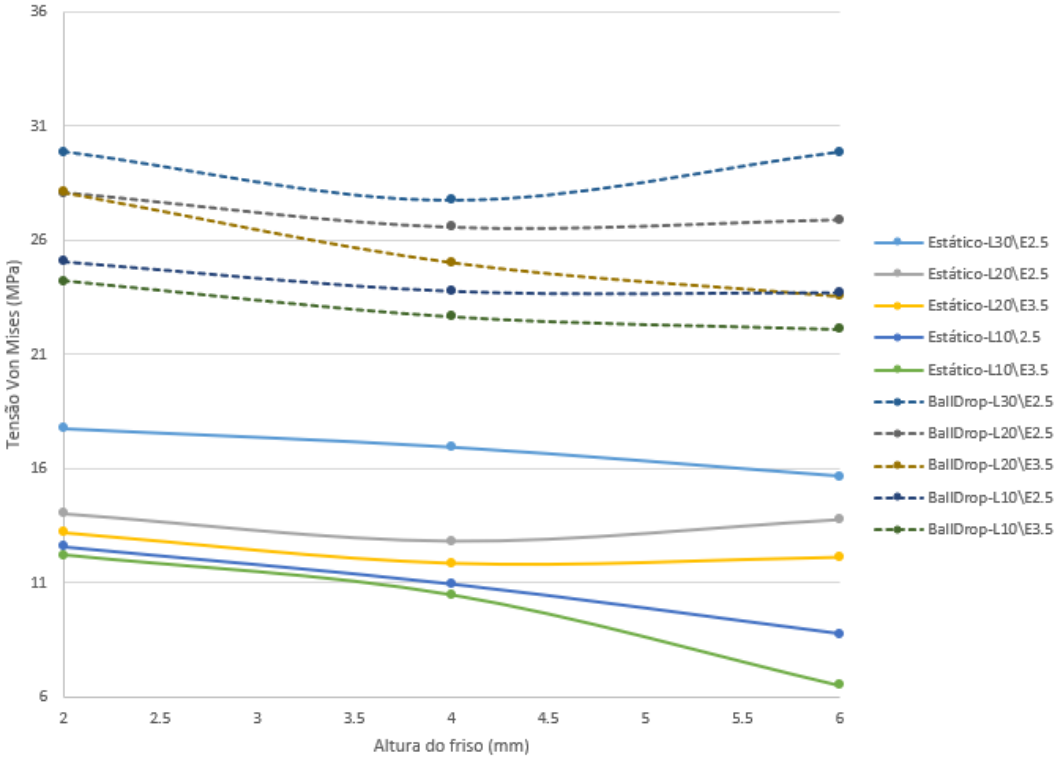


Figura 274 – Variação da tensão com a altura dos frisos

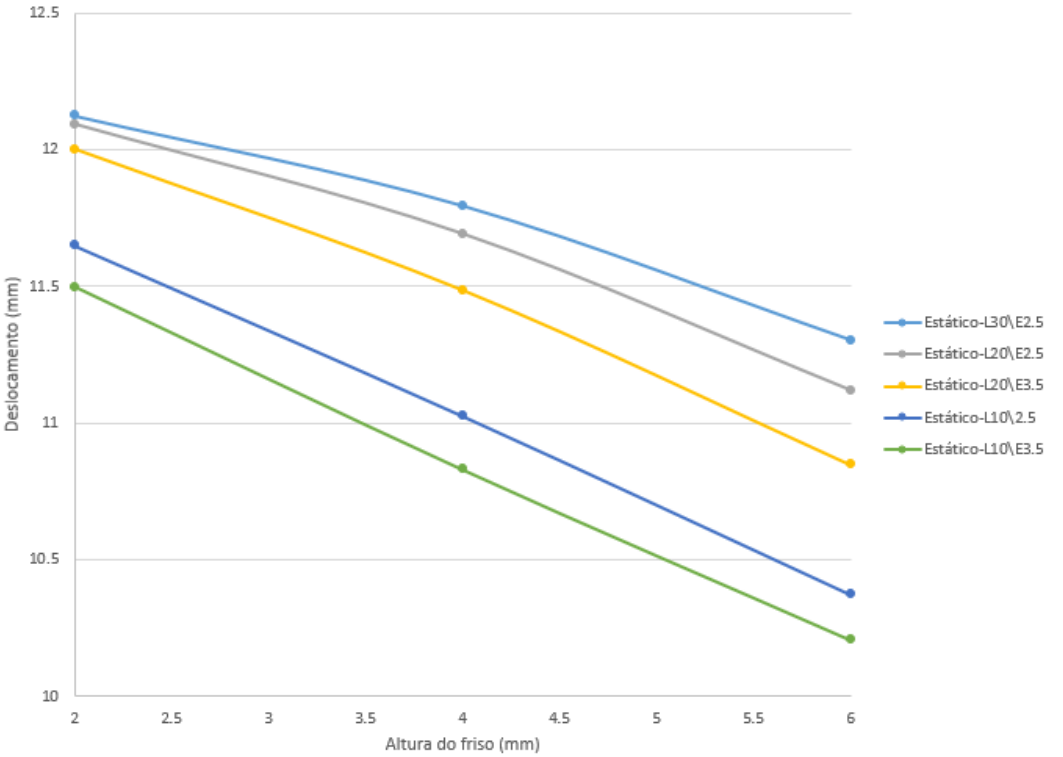


Figura 275 - Variação do deslocamento do impactor com a altura dos frisos

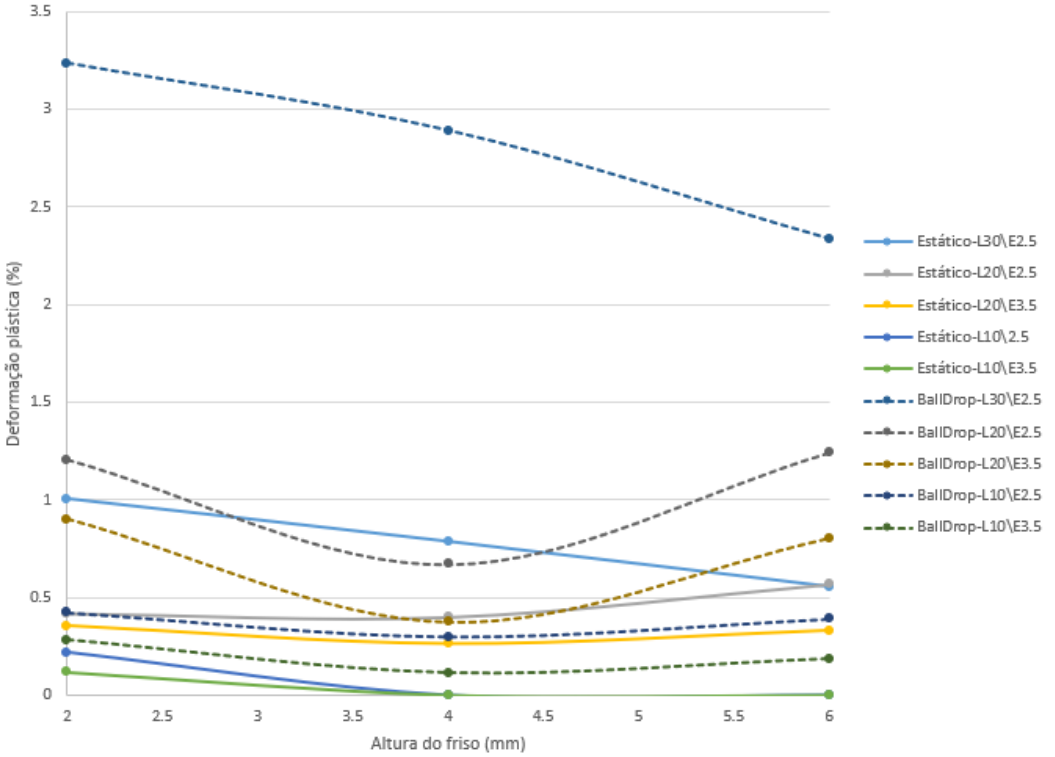


Figura 276 - Variação da deformação plástica com a altura dos frisos

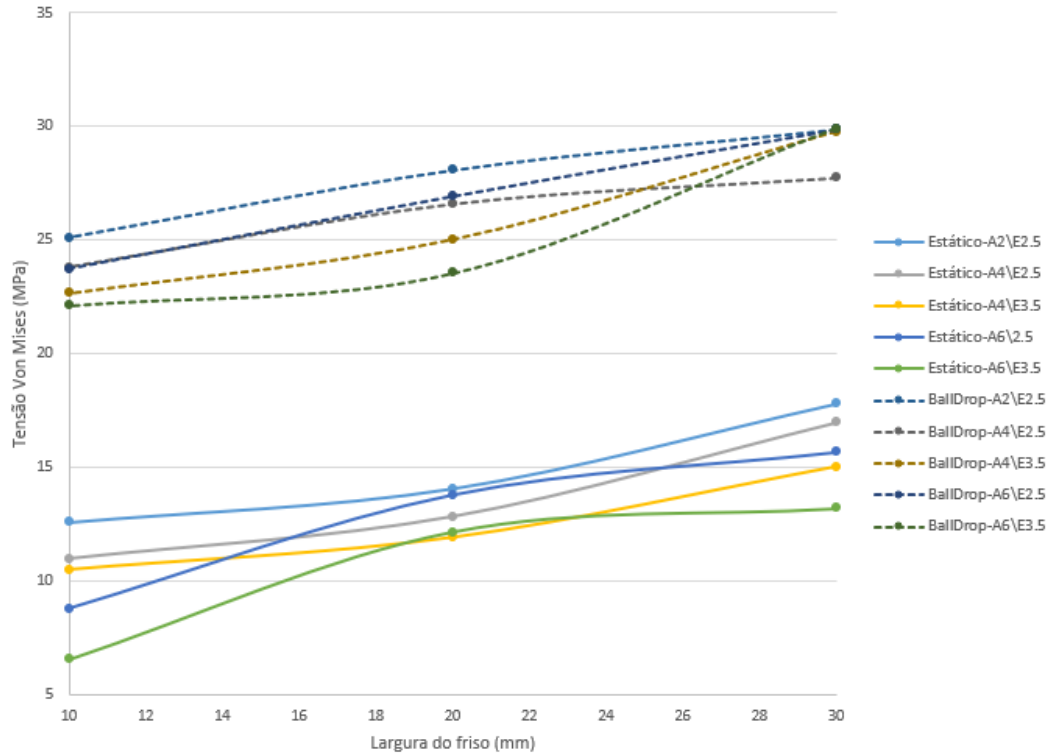


Figura 277 - Variação da tensão com a largura dos frisos

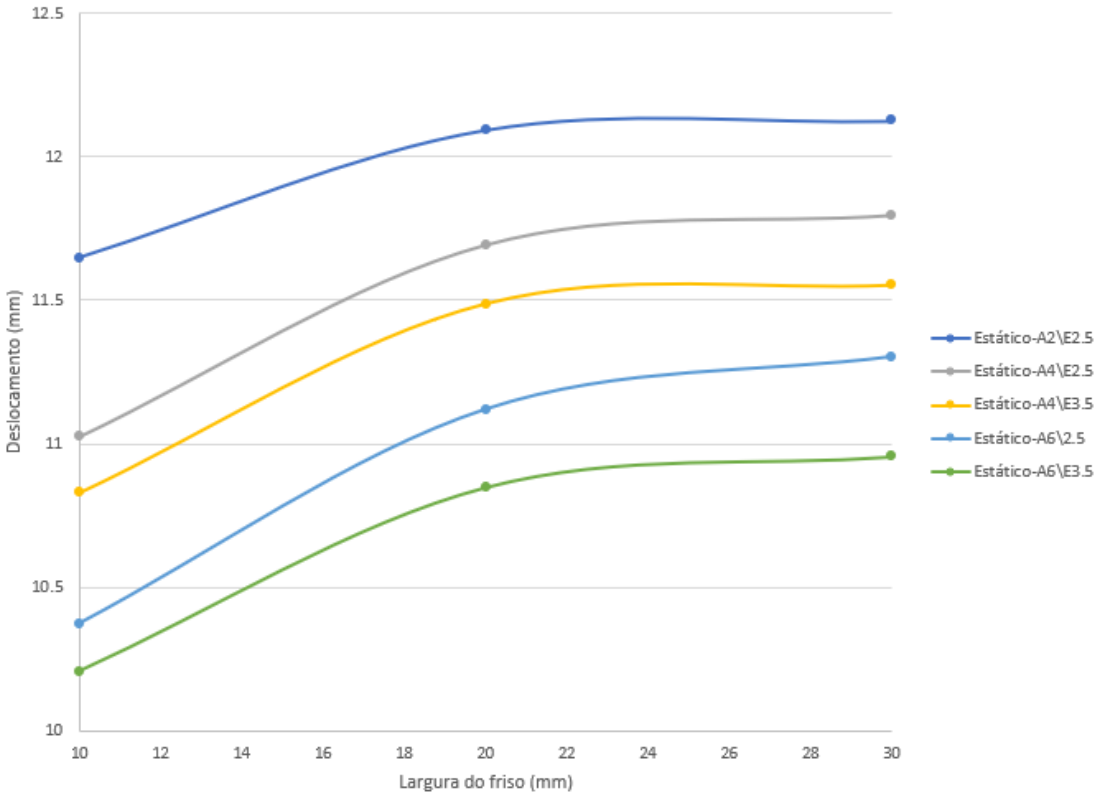


Figura 278 - Variação do deslocamento do impactor com a largura dos frisos

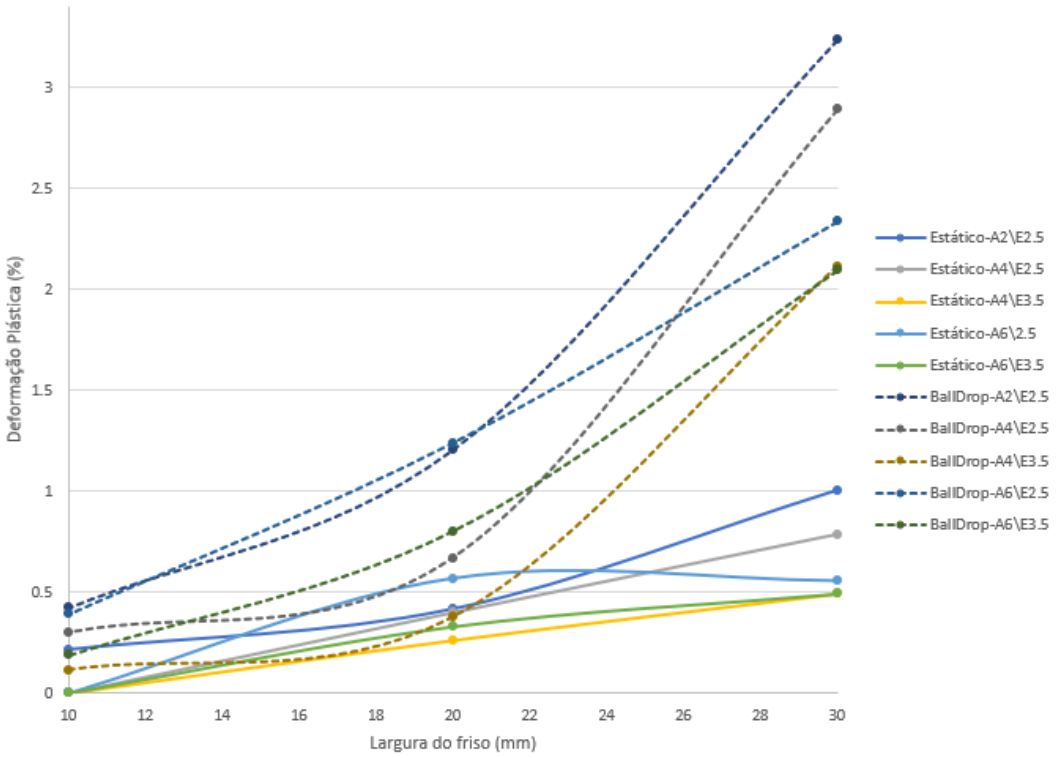


Figura 279 - Variação da deformação plástica com a largura dos frisos

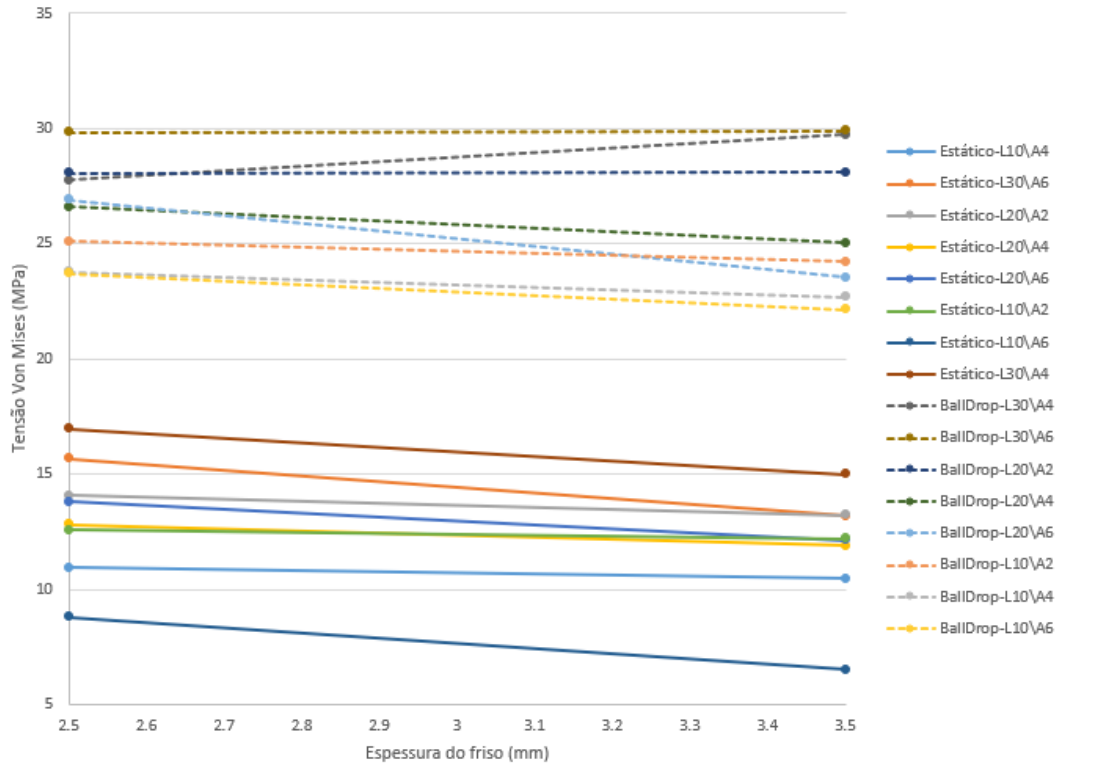


Figura 280 - Variação da tensão com a espessura dos frisos

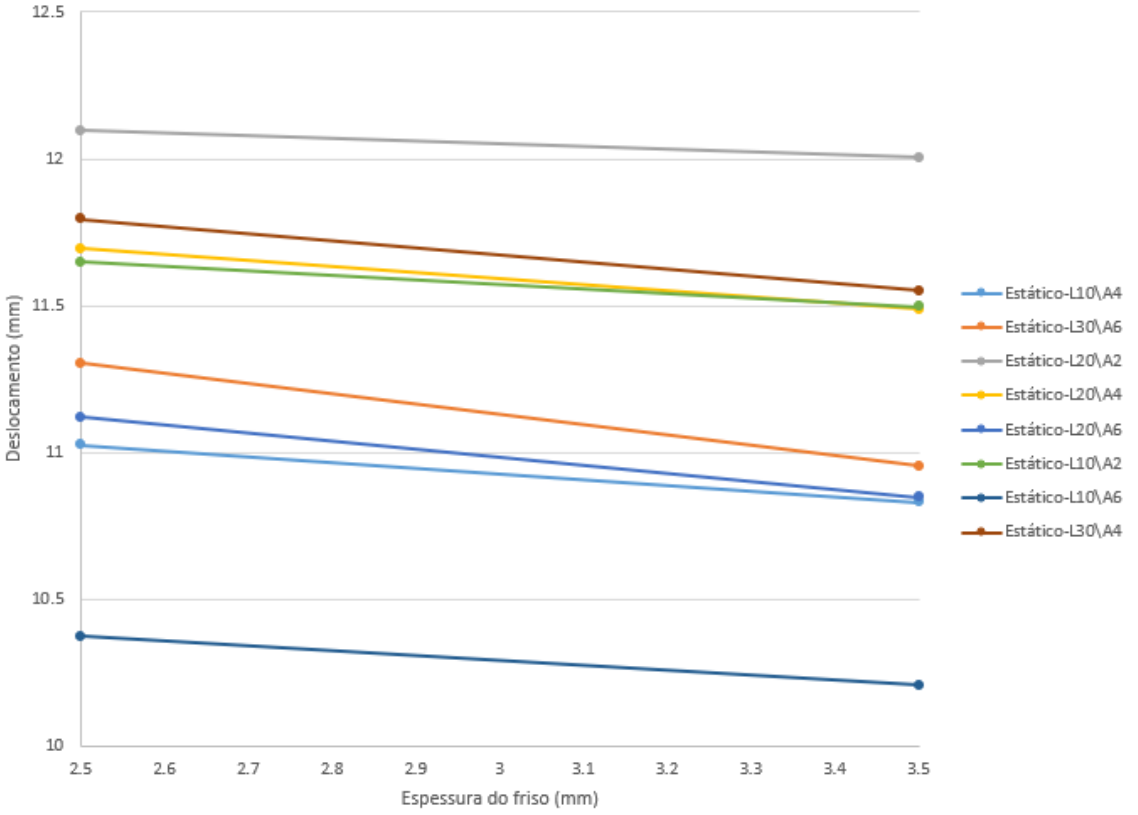


Figura 281 - Variação do deslocamento do impactor com a espessura dos frisos

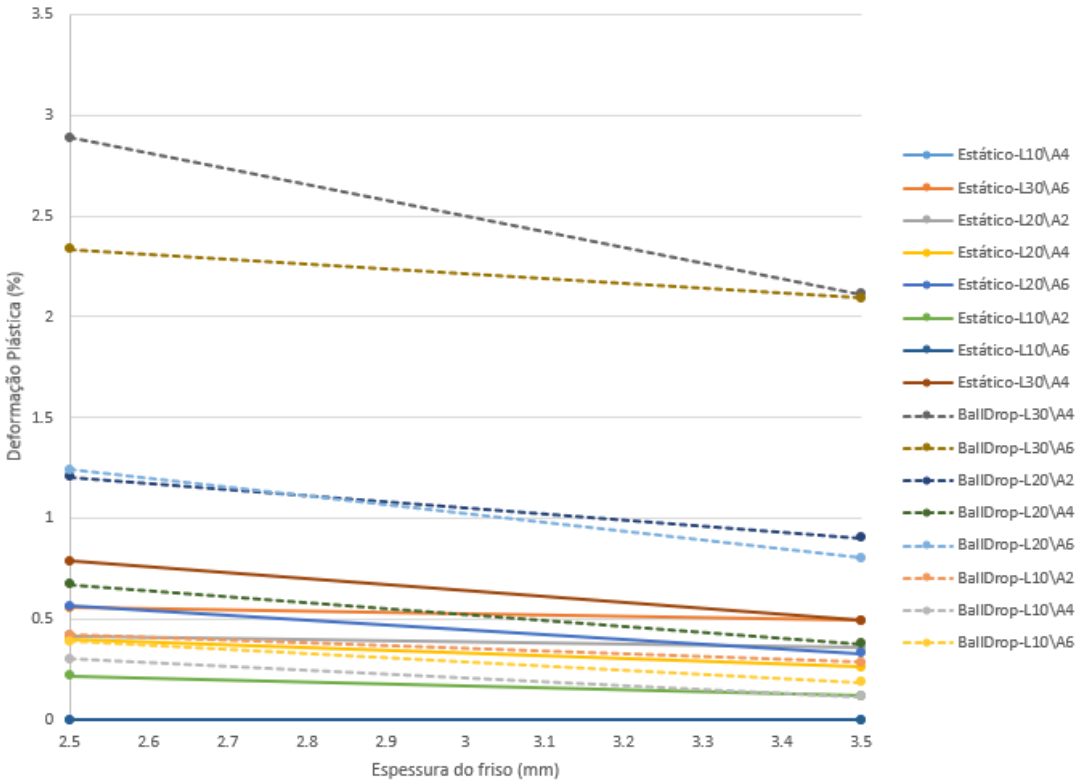


Figura 282 - Variação da deformação plástica com a espessura dos frisos

3.5.6 Análise do caso de estudo

Com o software desenvolvido, analisou-se o componente pretendido pela Simoldes®. Para tal geraram-se então dois modelos diferentes, um com o impactor do ensaio estático (Figura 283), e outro com a esfera do *Ball Drop* (Figura 284). Estes dois modelos são acedidos à vez pelo programa, consoante o ensaio que este pretenda realizar.

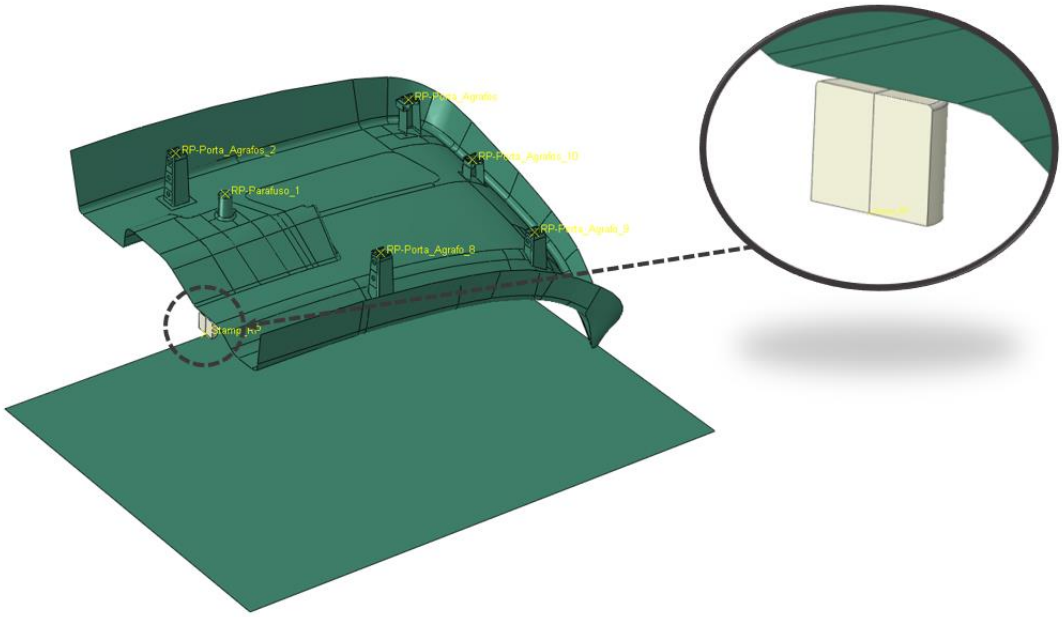


Figura 283 – Modelo do ensaio estático

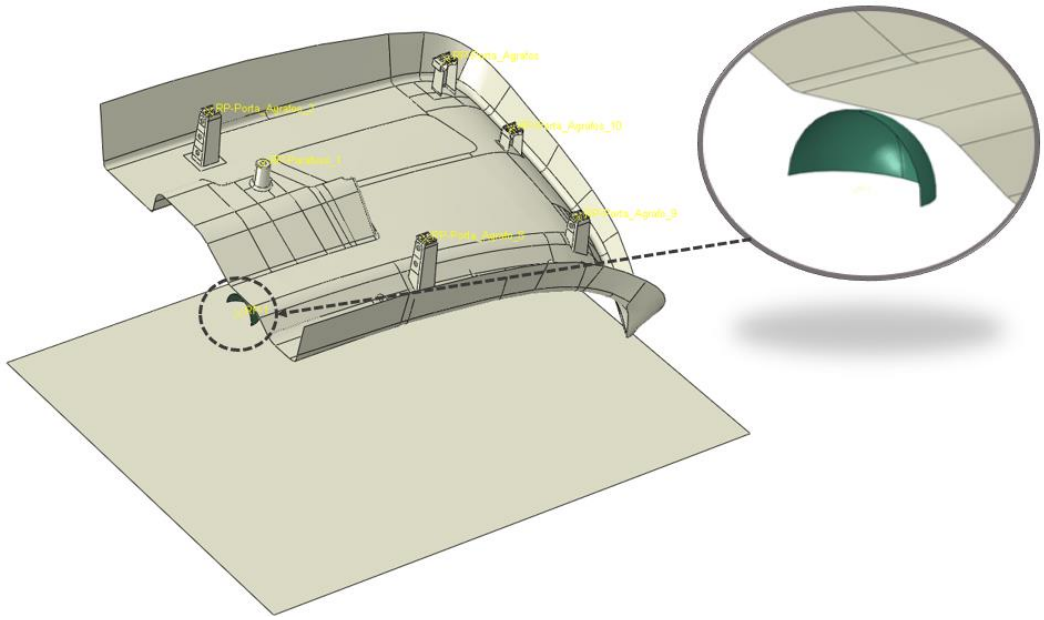


Figura 284 – Modelo do *Ball Drop*

Quanto aos parâmetros a serem impostos no software, foi definido pela Simoldes® que os parâmetros fossem os seguintes:

- Largura dos frisos mínima fosse de 10mm e a máxima de 30mm, sendo também analisadas as larguras de 15, 20 e 25 mm.
- Os frisos para a respetiva peça só podem possuir uma de duas espessuras 2.5mm ou 3.5mm.
- A altura dos frisos não pode ser menor que 2mm nem maior que 6mm, e, portanto, estudaram-se os valores de 2, 3, 4, 5 e 6 mm.

Com o realizar do ensaio reparou-se que, independentemente dos parâmetros dos frisos, uma das fixações da peça (porta agrafos) não conseguiria suportar os esforços nos dois ensaios, e, portanto, este teria de ser alterado para tornar a peça funcional (Figura 285). Como a análise dos porta agrafos se encontra a ser desenvolvida numa outra tese em paralelo a esta, alterou-se o programa de forma a que, na parte de análise dos resultados obtidos, não tomasse em conta os dados dos elementos dos porta agrafos, e apenas analisasse o corpo da peça.

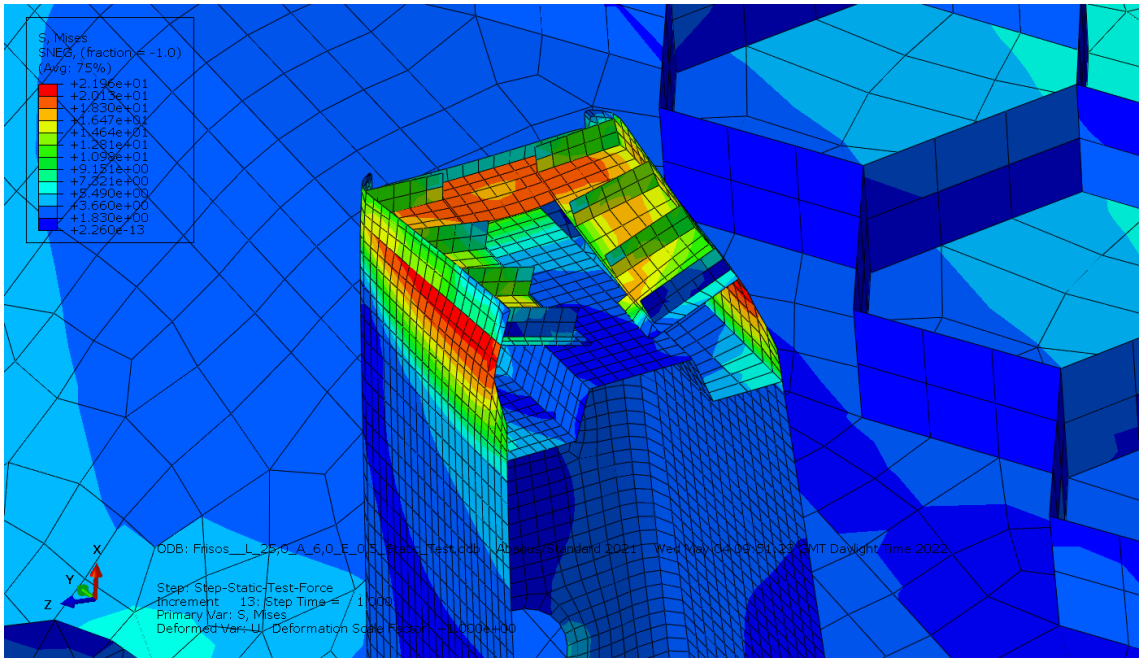


Figura 285 – Tensão no respetivo porta agrafo

Para esta parte desenvolveram-se as funções *Exclusion_Area* e *Exclusion_elements*. A *Exclusion_elements* simplesmente recebe um nome de um *set*, e trata de ir ao modelo buscar todas as *labels* dos elementos que estão inseridos dentro do respetivo *set*, e guarda-as numa lista que devolve ao programa (Figura 288). A *Exclusion_Area* envia a informação à *Exclusion_elements* sobre que *sets* é que devem ser analisados (Figura 287). A lista de *labels* gerada pela *Exclusion_Area*, é depois enviada para o programa

principal, e deste, para as funções de exportações de dados, e estas só analisam os elementos cujas *labels* não constam na lista gerada pela *Exclusion_Area*. Assim, nenhum dos elementos dos porta agrafos é tido em conta para a validação da peça base.

Para além dos porta agrafos, verificou-se ainda que, os elementos da zona dos frisos que se encontram ao longo do eixo de simetria (Figura 286), devido ao pequeno espaço existente para a geração da malha, não são fiáveis para a análise, pois muitas vezes estes elementos encontram-se distorcidos/deformados. Sendo assim, estes elementos também não foram tidos em consideração para a validação da peça, sendo que a função *Exclusion_Area* os seleciona através de uma *bounding box* localizada nessa mesma zona (Figura 287 Linhas 25-41).

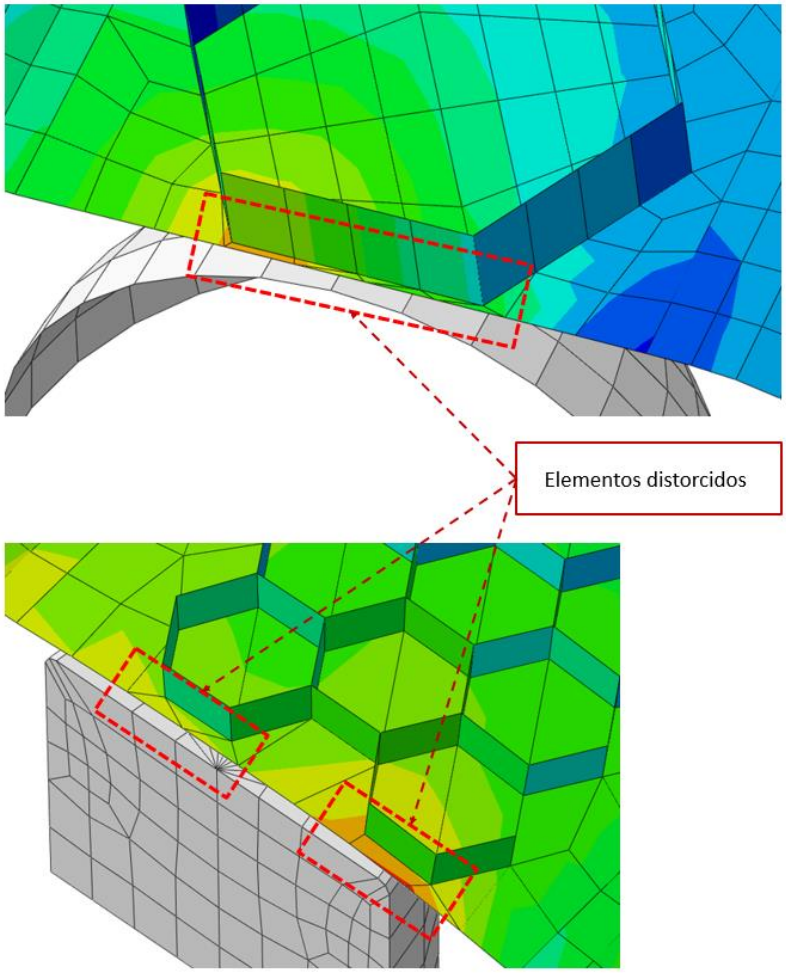


Figura 286 – Elementos segundo o eixo de simetria

```
# Get the element labels from the fixture elements
def Exclusion_Area(job_name):
    exclusion_element_list = [1]
    exclusion_set_name = 'Porta_Agrafos'
    exclusion_element_list = Exclusion_elements(exclusion_element_list,
        exclusion_set_name, job_name)
    exclusion_set_name = 'Porta_Agrafos_2'
    exclusion_element_list = Exclusion_elements(exclusion_element_list,
        exclusion_set_name, job_name)
    exclusion_set_name = 'Porta_Agrafos_8'
    exclusion_element_list = Exclusion_elements(exclusion_element_list,
        exclusion_set_name, job_name)
    exclusion_set_name = 'Porta_Agrafos_9'
    exclusion_element_list = Exclusion_elements(exclusion_element_list,
        exclusion_set_name, job_name)
    exclusion_set_name = 'Porta_Agrafos_10'
    exclusion_element_list = Exclusion_elements(exclusion_element_list,
        exclusion_set_name, job_name)

    # Internal variables
    p = mymodel.parts['Base']
    d = p.datums

    # Boundary box Coordinates
    bounding_box_point_1_id = p.features['Exclusion_Bounding_Point_1'].id
    bounding_box_point_2_id = p.features['Exclusion_Bounding_Point_2'].id
    bounding_box_point_1 = p.getCoordinates(d[bounding_box_point_1_id])
    bounding_box_point_2 = p.getCoordinates(d[bounding_box_point_2_id])

    # Reordering of the boundary box coordinates
    min_x_top = min(bounding_box_point_1[0],bounding_box_point_2[0])
    min_y_top = min(bounding_box_point_1[1],bounding_box_point_2[1])
    min_z_top = min(bounding_box_point_1[2],bounding_box_point_2[2])

    max_x_top = max(bounding_box_point_1[0],bounding_box_point_2[0])
    max_y_top = max(bounding_box_point_1[1],bounding_box_point_2[1])
    max_z_top = max(bounding_box_point_1[2],bounding_box_point_2[2])

    # Get the meshelement objects from the bounding box
    exclusion_elements_from_box = p.elements.getByBoundingBox(
        min_x_top,min_y_top, min_z_top,max_x_top,max_y_top,max_z_top)

    # Get the element label from all elements from the bounding box
    for element in exclusion_elements_from_box:
        exclusion_element_list.append(element.label)

    return exclusion_element_list
```

Figura 287 – Função de obtenção dos sets dos elementos a excluir (*Exclusion_Area*)

```python
# Get the element labels from a set
def Exclusion_elements(exclusion_element_list,
    exclusion_set_name, odb_name):

    # Open the odb
    odb = session.openOdb(name=odb_name + '.odb')
    myview =session.viewports[session.currentViewportName]
    myview.setValues(displayedObject=odb)

    # Internal Variable
    p = mymodel.parts['Base']

    # Get all element labels inside the set
    exclusion_set = p.sets[exclusion_set_name]
    test= []
    for exclusion_element in exclusion_set.elements:
        exclusion_element_list.append(exclusion_element.label)
        test.append(exclusion_element)

    # return the list of element labels
    return exclusion_element_list
```

Figura 288 - Função de obtenção das labels dos elementos a excluir (*Exclusion_elements*)

Com estes casos tratados, voltou-se a executar o programa para a peça do caso de estudo, e agora já se obtiveram resultados mais fidedignos, como se pode ver na Tabela 10. Os gráficos de cada uma das geometrias encontram-se no anexo 6.1.

Tabela 10 – Resultados do estudo dos frisos da peça do caso de estudo

Largura (mm)	Altura (mm)	Espessura (mm)	Teste	Tensão v. Mises (MPa)	Deslocamento do impactor (mm)	Deformação Plástica	Massa (Kg)
30	2	2.5	Estático	10.879	9.041	8.312E-04	1.082
30	2	2.5	*Ball Drop*	21.799	NC	1.025E-02	1.082
25	2	2.5	Estático	10.897	9.028	8.563E-04	1.083
30	3	2.5	Estático	10.809	8.885	7.373E-04	1.084
20	2	2.5	Estático	10.886	9.027	8.414E-04	1.086
30	2	3.5	Estático	10.842	8.940	7.826E-04	1.086
25	3	2.5	Estático	10.830	8.876	7.668E-04	1.090
30	4	2.5	Estático	10.744	8.736	6.511E-04	1.090
25	2	3.5	Estático	10.852	8.922	7.954E-04	1.092
15	2	2.5	Estático	10.933	9.088	9.041E-04	1.092
20	3	2.5	Estático	10.831	8.900	7.681E-04	1.094
30	3	3.5	Estático	10.752	8.743	6.630E-04	1.094
30	5	2.5	Estático	10.684	8.604	5.715E-04	1.095
25	4	2.5	Estático	10.769	8.738	6.852E-04	1.096
20	2	3.5	Estático	10.848	8.938	7.903E-04	1.097
30	6	2.5	Estático	10.646	8.402	4.545E-04	1.101
20	4	2.5	Estático	10.786	8.795	7.082E-04	1.102

25	3	3.5	Estático	10.770	8.736	6.864E-04	1.102
15	3	2.5	Estático	10.896	8.997	8.548E-04	1.102
30	4	3.5	Estático	10.660	8.539	5.399E-04	1.102
25	5	2.5	Estático	10.718	8.633	6.174E-04	1.103
15	2	3.5	Estático	10.904	9.019	8.660E-04	1.106
20	3	3.5	Estático	10.785	8.780	7.071E-04	1.108
20	5	2.5	Estático	10.753	8.716	6.638E-04	1.110
30	5	3.5	Estático	10.637	8.383	4.498E-04	1.110
25	6	2.5	Estático	10.651	8.482	5.275E-04	1.110
25	4	3.5	Estático	10.698	8.579	5.911E-04	1.112
15	4	2.5	Estático	10.870	8.928	8.208E-04	1.112
10	2	2.5	Estático	10.731	8.650	6.337E-04	1.115
20	6	2.5	Estático	10.709	8.607	6.053E-04	1.117
30	6	3.5	Estático	10.630	8.157	4.121E-04	1.118
20	4	3.5	Estático	10.731	8.645	6.357E-04	1.119
15	3	3.5	Estático	10.860	8.904	8.076E-04	1.119
25	5	3.5	Estático	10.651	8.448	5.193E-04	1.121
15	5	2.5	Estático	10.853	8.877	7.978E-04	1.121
20	5	3.5	Estático	10.691	8.541	5.813E-04	1.130
25	6	3.5	Estático	10.629	8.261	4.123E-04	1.131
15	6	2.5	Estático	10.822	8.783	7.561E-04	1.131
10	3	2.5	Estático	10.661	8.419	5.042E-04	1.131

15	4	3.5	Estático	10.828	8.811	7.651E-04	1.133
10	2	3.5	Estático	10.665	8.500	5.444E-04	1.137
20	6	3.5	Estático	10.653	8.403	5.013E-04	1.141
15	5	3.5	Estático	10.805	8.737	7.342E-04	1.146
10	4	2.5	Estático	10.655	8.210	4.082E-04	1.147
15	6	3.5	Estático	10.762	8.620	6.763E-04	1.159
10	3	3.5	Estático	10.652	8.244	4.125E-04	1.160
10	5	2.5	Estático	10.660	8.026	3.364E-04	1.163
10	6	2.5	Estático	10.665	7.780	2.305E-04	1.179
10	4	3.5	Estático	10.645	8.007	3.025E-04	1.182
10	5	3.5	Estático	10.675	7.827	2.556E-04	1.205
10	6	3.5	Estático	10.638	7.572	1.007E-04	1.227

Analisando-se estes resultados, pode-se concluir que a geometria que conseguiu satisfazer os requisitos do cliente, gastando a menor quantidade de material possível, foi a de 30mm de largura, 2mm de altura e 2.5mm de espessura, que corresponde aos parâmetros mínimos que podem ser utilizados na produção de frisos para a referida peca. Tendo obtido:

- Ensaio estático:
  - Tensão máxima – 10.88 MPa < 19.77MPa;
  - Deformação plástica – 0.83% < 5.45%;
  - Deslocamento do impactor – 9.04mm;
- *Ball Drop*:
  - Tensão máxima – 21.80 MPa < 27.89 MPa;
  - Deformação plástica - 1.02% < 2.48%;

Como o resultado obtido para os frisos corresponde aos parâmetros mínimos possíveis, voltou-se a testar a peça, mas desta vez sem frisos nenhuns, tendo obtido os seguintes resultados:

Tabela 11 – Resultados do estudo sem frisos da peça do caso de estudo

Largura (mm)	Altura (mm)	Espessura (mm)	Teste	Tensão v. Mises (MPa)	Deslocamento do impactor (mm)	Deformação Plástica	Massa (Kg)
0	0	0	Estático	11.046	9.372	1.052E-03	1.062
0	0	0	*Ball Drop*	26.914	NC	7.393E-03	1.062

Aqui pode-se presenciar, que mesmo sem frisos a peça é capaz de suportar os esforços dos ensaios, porém este encontra-se no limite das suas capacidades, estando muito próximo de falhar o ensaio do *Ball Drop*. Por esta razão, aconselha-se que devem ser colocados os frisos com os parâmetros mínimos, levando assim a que a peça possua um fator de segurança de 1.28.

Face aos resultados obtidos, foi pedido pela Simoldes® que fosse realizado mais um estudo para os frisos, mas desta vez, analisando apenas uma espessura de 0.95mm, e uma altura 5 ou 8mm. Correndo-se novamente o software, mas com estes novos parâmetros obtiveram-se os resultados da Tabela 12, e os gráficos do anexo 6.2.

Tabela 12– Resultados do segundo estudo dos frisos da peça do caso de estudo

Largura (mm)	Altura (mm)	Espessura (mm)	Teste	Tensão v. Mises (MPa)	Deslocamento do impactor (mm)	Deformação Plástica	Massa (Kg)
30	5	0.95	Estático	10.877	9.045	8.283E-04	1.075
30	5	0.95	*Ball Drop*	20.279	NC	1.048E-03	1.075
25	5	0.95	Estático	10.890	9.049	8.463E-04	1.076
20	5	0.95	Estático	10.921	9.101	8.879E-04	1.078
30	8	0.95	Estático	10.778	8.860	6.964E-04	1.079
15	5	0.95	Estático	10.957	9.200	9.341E-04	1.083
25	8	0.95	Estático	10.809	8.900	7.383E-04	1.084

20	8	0.95	Estático	10.853	8.980	7.968E-04	1.087
15	8	0.95	Estático	10.916	9.105	8.798E-04	1.093
10	5	0.95	Estático	10.723	8.609	6.246E-04	1.099
10	8	0.95	Estático	10.666	8.169	4.400E-04	1.118

Observando-se estes novos dados pode-se concluir que a melhor geometria neste caso, apresenta os seguintes parâmetros: largura 30 mm, altura 5 mm e espessura 0.95 mm, sendo que para esta geometria se obtiveram os valores de:

- Ensaio estático:
  - Tensão máxima – 10.88 MPa < 19.77 MPa;
  - Deformação plástica – 0.83 % < 5.45 %;
  - Deslocamento do impactor – 9.05 mm;
- *Ball Drop*:
  - Tensão máxima – 20.28 MPa < 27.89 MPa;
  - Deformação plástica - 1.04 % < 2.48 %;

Estes valores são próximos dos do primeiro estudo, contudo, esta última geometria possui um melhor desempenho mecânico, durante o ensaio de *Ball Drop*, tendo uma tensão máxima de 20.28 MPa, enquanto que a primeira obteve 21.80 MPa, e necessita de uma menor quantidade de material para ser produzida. Sendo assim, tendo em conta estes dois estudos, a melhor geometria de frisos para a peça é largura 30 mm, altura 5mm e espessura 0.95 mm (Figura 289).

Comparando a geometria obtida pelo programa, com a que tinha sido previamente implementada pela Simoldes®, que possui as dimensões de:

- Geometria previamente implementada pela Simoldes® (Figura 290):
  - Largura – 24 mm;
  - Altura – 5 mm;
  - Espessura – 0.95mm;
  - Massa – 1.076 Kg.

Pode-se comprovar que a largura dos frisos pode ser maior do que a previamente implementada e, portanto, poupar-se algum material na produção de respetivo componente.

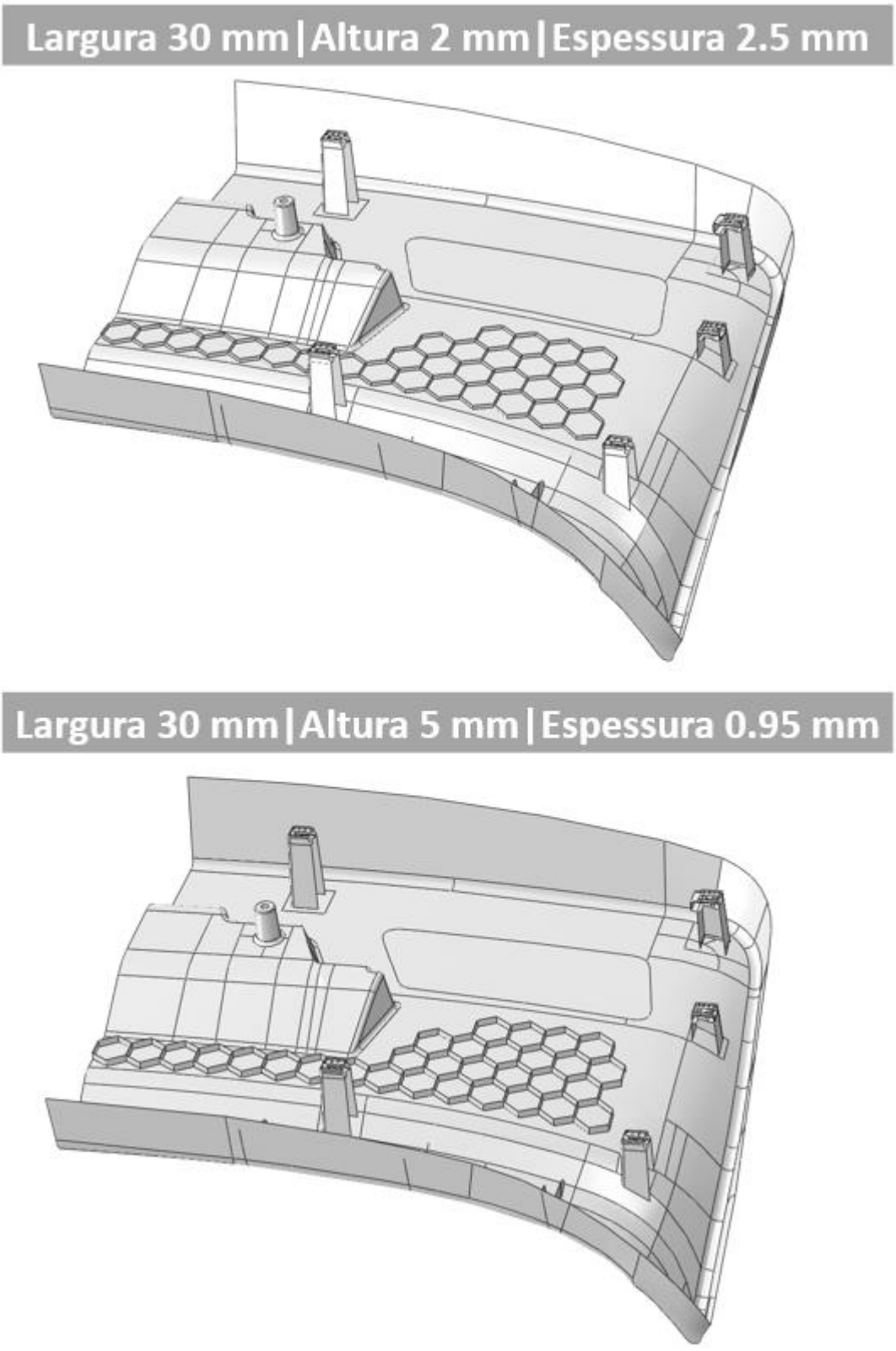


Figura 289 – Peças resultantes dos dois estudos

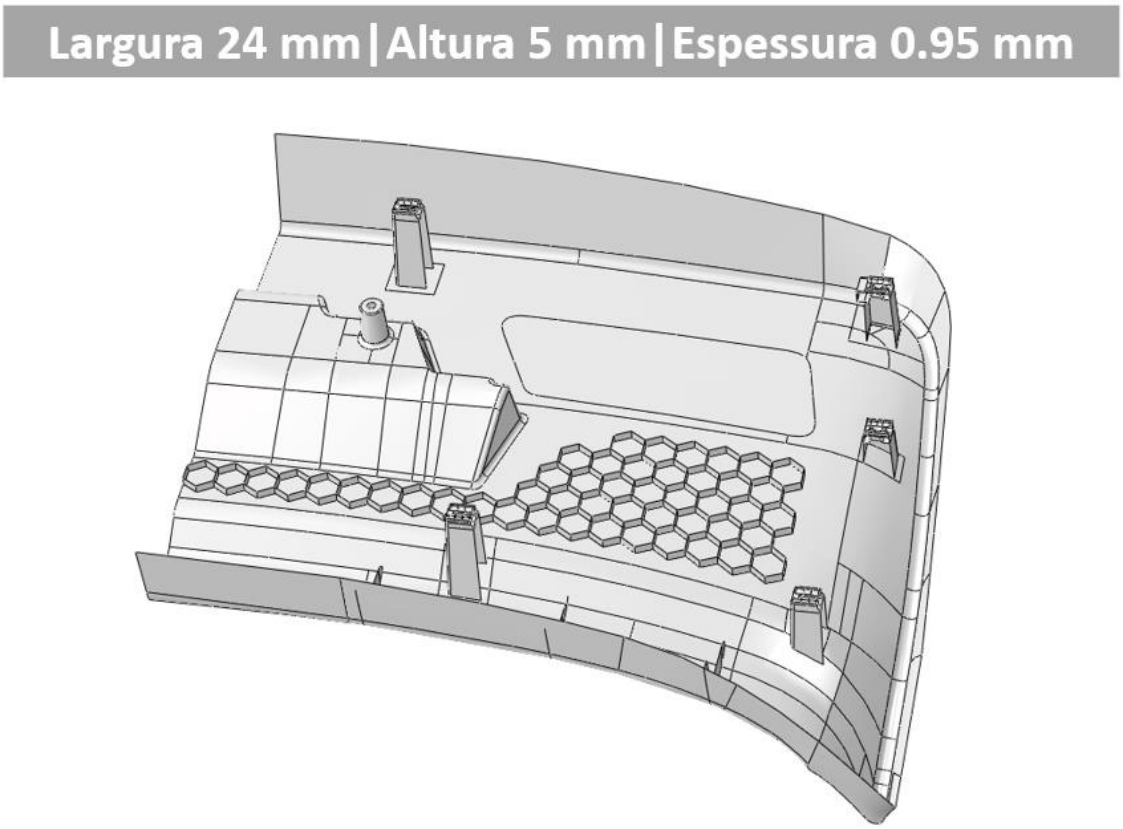


Figura 290 – Geometria de frisos previamente implementada pela Simoldes®

CONCLUSÕES E PROPOSTAS DE TRABALHOS FUTUROS

4 CONCLUSÕES E PROPOSTAS DE TRABALHOS FUTUROS

4.1 Conclusões

Com o desenvolvimento desta dissertação esperava-se que fosse desenvolvido um software que utilizasse as capacidades de simulação numérica do Abaqus®, de forma a poder-se a analisar quais são as melhores dimensões para o reforço de favo de abelha de um qualquer componente que a empresa decidesse analisar, e depois de forma a comprovar o seu funcionamento, utilizá-lo para estudar uma dada peça projetada pela empresa.

Face aos requisitos da empresa, primeiro desenvolveu-se um algoritmo de desenho que fosse capaz de desenhar diferentes tipos de reforço de favo de abelha, com diferentes parâmetros dimensionais, tendo-se imposto parâmetros adicionais, que permitiam que o desenho fosse desenvolvido da forma mais rápida possível, como a supressão de hexágonos, e a utilização de diversas estruturas de dados adequadas para as situações necessárias.

Com o algoritmo de desenho concluído, tratou-se então de formular uma metodologia de extrusão dos hexágonos, que permitisse que estes fossem extrudidos, tanto para superfícies simples, como para irregulares, tendo-se conseguido desenvolver uma metodologia capaz de trabalhar tanto com superfícies planas, como curvas.

Com a parte da alteração da geometria tratada, formularam-se então as condições de ensaio impostas pelo respetivo cliente, bem como os critérios de falha, que permitem avaliar se uma geometria passou ou não nos respetivos testes. Para além da imposição dos ensaios, criaram-se metodologias de monitorização do cálculo, de forma a confirmar-se se este está a ser feito corretamente, e formas de extração e organização dos dados resultantes do estudo, para mais tarde serem tratados pelo programa.

Com os dados devidamente armazenados, foram desenvolvidas metodologias de análise que permitissem que o estudo fosse feito o mais rápido possível, como por exemplo a organização das geometrias por ordem crescente de peso após o ensaio estático, de maneira que não fosse necessário executar o ensaio de *Ball Drop* para todas elas.

Com os dados obtidos, e para que o utilizador possa analisar de forma fácil os resultados obtidos, foram incluídas no programa funções para processar os resultados e apresentá-los de forma gráfica, de forma totalmente automática.

Com todo este processo de análise automática desenvolvido, estudaram-se então três peças como exemplo, onde se pode verificar como cada parâmetro dos frisos influência diferentes propriedades mecânicas de um componente. Por fim estudou-se o componente proposto pela Simoldes®, tendo-se chegado à conclusão de que este apenas necessita de alterações geométricas mínimas, para satisfazer as condições impostas pelo cliente.

4.2 Propostas de trabalhos futuros

Com a realização da presente dissertação verifica-se que existem algumas partes do software desenvolvido, que podem ser melhoradas, de forma a torná-lo mais abrangível para outros casos de estudo. São então sugeridas algumas propostas para trabalhos futuros:

- Alterar o algoritmo de deteção de limites, de forma a permitir um maior número de limites da zona de reforço - De momento o algoritmo do capítulo 3.5.3.6.4 apenas permite utilizar até 4 limites de área, o que é suficiente para a análise do componente pretendido pela Simoldes®. Porém existem peças que necessitam de um maior número de limites para serem resolvidas, e, portanto, o passo seguinte envolveria reformular o algoritmo de forma a poder-se impor uma infinidade de limites, cujo número seria imposto pelo utilizador;
- Desenvolver o cálculo matemático para uma transição de um limite superior para um inferior- Como no caso de estudo deste trabalho não existia uma transição de um limite com uma altura superior ao do seguinte, não foi desenvolvida a metodologia matemática necessária para serem tratados estes casos e, portanto, este seria um caso que poderia ser tratado no futuro. De momento, nos casos mencionados o algoritmo gera o erro presente na Figura 291;

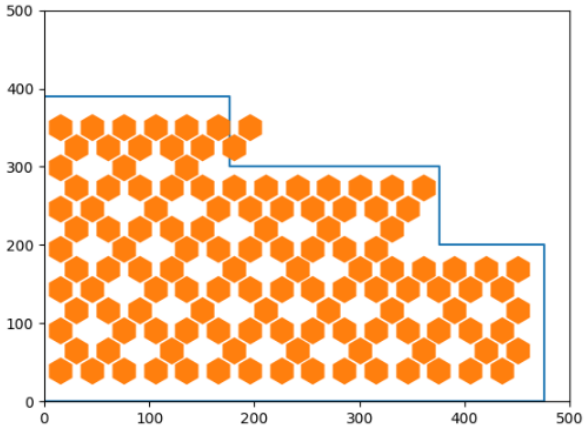


Figura 291 – Erro gerado na transição para um limite inferior

- Adicionar outros estudos referidos no caderno de encargos, de outros clientes, e permitir ao utilizador escolher quais pretende utilizar. Neste caso de estudo são realizados dois ensaios: um estático com 200 N de força aplicada na peça, e um *Ball Drop* no qual uma esfera de 500 g é largada sobre a respetiva peça. Apesar destes serem dois ensaios que são requeridos pelo cliente da peça que foi estudada, outros clientes possuem ensaios diferentes, e seria interessante fazer com que o programa conseguisse executar outro tipo de ensaios, e permitir ao utilizador escolher quais os que pretende utilizar num respetivo estudo;
- Desenvolver algoritmos de desenho para diferentes geometrias de frisos – no âmbito desta dissertação, apenas se pretendeu estudar frisos com uma geometria de favo de abelha, contudo, estes não são as únicas geometrias que existem, e, portanto, uma das propostas futuras seria a de implementar algoritmos de desenho para essas mesmas geometrias;
- Permitir que algumas das variáveis internas do sistema possam ser alteradas pelo utilizador – dentro do programa existem variáveis, como a força a aplicar no ensaio estático, ou a velocidade da bola no *Ball Drop*, que de momento são variáveis que o utilizador não consegue alterar sem mexer no código interno do programa, e, portanto, o que mais tarde poderia ser feito, é tornar estas mesma variáveis, acessíveis ao utilizador, através do GUI. Aqui poderia também ser implementado o *Fox Toolkit*, que é uma ferramenta mais complexa para criação de GUI's dentro do Abaqus®, que permite criar GUI's mais complexos.
- Desenvolver uma base de dados para todos os materiais a utilizar – De momento a base de dados dos materiais, consiste na utilização de ficheiros de texto, em que cada um possui os dados de cada um dos respetivos materiais. Isto foi feito, pois o ambiente do Python do Abaqus®, é um ambiente fechado, e, portanto, não possui as livrarias convencionais para a utilização de bases de dados mais correntes. Porém, é possível incorporar estas funcionalidades, se

existir um ficheiro externo ao ambiente do Abaqus®, que controle e envie informação ao ficheiro interno, que foi desenvolvido nesta dissertação. Assim, o ficheiro externo pode aceder a bases de dados, como por exemplo em SQL, e enviar a informação ao ficheiro interno. Assim a informação das bases de dados fica mais organizada e rápida de aceder.

- Permitir a análise de peças com elementos sólidos – dependendo das dimensões da peça base, nem sempre é correta a aproximação da geometria através de planos médios e, portanto, nestes componentes a análise dos frisos deve ser feita com elementos de malha 3D. Para que o software possa ser utilizado neste tipo de componentes este tem de ser alterado de forma a permitir que os frisos sejam desenhados na sua forma sólida, ou, caso estes sejam desenhados em plano médio, permitir a congruência entre os graus de liberdade dos diferentes tipos de elementos existentes no modelo resultante.

BIBLIOGRAFIA E OUTRAS FONTES DE INFORMAÇÃO

5 BIBLIOGRAFIA E OUTRAS FONTES DE INFORMAÇÃO

[1] M. Freitas. "SIMOLDES Plásticos." COMPETE 2020. https://www.compete2020.gov.pt/noticias/detalhe/NL_SIMOLDES_Plasticos (accessed Outubro 12, 2021).

[2] S. Plastics. "Simoldes Plastics." https://www.simoldes.com/plastics/empresa/ (accessed Outubro 12, 2021).

[3] *2016 PRODUCTION STATISTICS*, International Organization of Motor Vehicle Manufacturers, Nov. 2021. [Online]. Available: https://www.oica.net/category/production-statistics/2016-statistics/

[4] *2018 PRODUCTION STATISTICS*, International Organization of Motor Vehicle Manufacturers, Nov. 2021. [Online]. Available: https://www.oica.net/category/production-statistics/2018-statistics/

[5] *2020 PRODUCTION STATISTICS*, International Organization of Motor Vehicle Manufacturers, Nov. 2021. [Online]. Available: https://www.oica.net/category/production-statistics/2020-statistics/

[6] *World motor vehicle production*, European Automobile Manufacturers Association, Nov. 2021. [Online]. Available: https://www.acea.auto/figure/world-motor-vehicle-production/

[7] *Motor vehicle production, by world region*, European Automobile Manufacturers Association, Nov. 2021. [Online]. Available: https://www.acea.auto/figure/motor-vehicle-production-by-world-region/

[8] *Employment trends in the EU automotive sector*, European Automobile Manufacturers Association, Nov. 2021. [Online]. Available: https://www.acea.auto/figure/employment-trends-in-eu-automotive-sector/

[9] *EU manufacturing employment*, European Automobile Manufacturers Association, Nov. 2021. [Online]. Available: https://www.acea.auto/figure/eu-manufacturing-employment/

[10] AFIA. "AFIA." https://afia.pt/) (accessed Outubro 23, 2021).

[11] (2016) Indústria automóvel e Componentes. *Portugal Global*. Available: https://portugalglobal.pt/PT/RevistaPortugalglobal/2016/Documents/Portugalglobal_n87.pdf

[12] B. Zabala *et al.*, "Mechanism-based wear models for plastic injection moulds," *Wear,* vol. 440-441, p. 203105, 2019/12/15/ 2019, doi: https://doi.org/10.1016/j.wear.2019.203105.

[13] C. Capela, "Processamento de plásticos e materiais compósitos: comportamento mecânico de componentes em serviço," Dissertação de Especialista, Departamento de Engenharia Mecânica, Instituto Politécnico de Leiria, Leiria, 2010. [Online]. Available: https://iconline.ipleiria.pt/bitstream/10400.8/2690/1/PROCESSAMENTO%20DE

%20PL%C3%81STICOS%20E%20MATERIAIS%20COMP%C3%93SITOS_COMPORTAME.pdf

[14] M. F. de S. F. de Moura, A. M. B. de Morais, and A. G. de Magalhães, *Materiais compósitos: materiais, fabrico e comportamento mecânico*. Publindústria, 2005.

[15] S. Kulkarni, *Robust Process Development and Scientific Molding: Theory and Practice*. Carl Hanser Verlag GmbH & Company KG, 2017.

[16] D. V. Rosato and M. G. Rosato, *Injection Molding Handbook*. Springer US, 2012.

[17] H. Kuhn, D. Medlin, and A. I. H. Committee, *Mechanical Testing and Evaluation*. ASM International, 2000.

[18] J. R. Davis, *Tensile Testing, 2nd Edition*. ASM International, 2004.

[19] J. P. Davim and A. G. Magalhães, *Ensaios mecânicos e tecnológicos: inclui exercícios resolvidos e propostos*. PUBLINDUSTRIA, 2010.

[20] Á. F. M. Azevedo, *Método dos Elementos Finitos*, 1ª Edição ed. Porto: Faculdade de Engenharia do Porto, 2003.

[21] D. L. Logan, *A First Course in the Finite Element Method: Enhanced Version*. Cengage Learning, 2022.

[22] R. D. S. G. Campilho, *Método de Elementos Finitos: Ferramentas para Análise Estrutural*. Porto, Portugal: Publindústria, 2012.

[23] S. S. Rao, *The Finite Element Method in Engineering: Pergamon International Library of Science, Technology, Engineering and Social Studies*, 6th ed. Elsevier Science, 2017.

[24] O. C. Zienkiewicz, R. L. Taylor, R. L. Taylor, and J. Z. Zhu, *The Finite Element Method: Its Basis and Fundamentals* (no. vol. 1). Butterworth-Heinemann, 2013.

[25] K. J. Bathe, *Finite Element Procedures*. Klaus-Jürgen Bathe, 2014.

[26] D. Hutton, *Fundamentals Of Finite Element Analysis*. McGraw-Hill Education (India) Pvt Limited, 2005.

[27] J. Sarrate and M. Staten, *Proceedings of the 22nd International Meshing Roundtable*. Springer International Publishing, 2016.

[28] D. Lo, *Finite Element Mesh Generation*. Taylor & Francis Group, 2017.

[29] C. Rodrigues, "Fundamentos de Análise Estrutural - Método dos elementos finitos - Introdução," Pós-graduação em cálculo assisitido de estruturas, Departamento de Engenharia Mecânica, Instituto Superior de Engenharia do Porto, Porto, Portugal, 2016.

[30] S. P. Sastry, "A 2D advancing-front Delaunay mesh refinement algorithm," *Computational Geometry,* vol. 97, p. 101772, 2021/08/01/ 2021, doi: https://doi.org/10.1016/j.comgeo.2021.101772.

[31] V. Koncz, F. Izsák, Z. Noszticzius, and K. Kály-Kullai, "Adaptive moving mesh algorithm based on local reaction rate," *Heliyon,* vol. 7, no. 1, p. e05842, 2021/01/01/ 2021, doi: https://doi.org/10.1016/j.heliyon.2020.e05842.

[32] L. Qiao, Y. Zheng, H. Wu, Y. Wang, and X. Du, "Improved block-Jacobi parallel algorithm for the SN nodal method with unstructured mesh," *Progress in Nuclear Energy,* vol. 133, p. 103629, 2021/03/01/ 2021, doi: https://doi.org/10.1016/j.pnucene.2021.103629.

[33] D. Simson and K. Zając, "Applications of mesh algorithms and self-dual mesh geometries of root Coxeter orbits to a Horn-Sergeichuk type problem," *Linear Algebra and its Applications,* vol. 632, pp. 79-152, 2022/01/01/ 2022, doi: https://doi.org/10.1016/j.laa.2021.09.005.

[34] Y. A. Abdel-Nasser, "Frontal crash simulation of vehicles against lighting columns using FEM," *Alexandria Engineering Journal,* vol. 52, no. 3, pp. 295-299, 2013/09/01/ 2013, doi: https://doi.org/10.1016/j.aej.2013.01.005.

[35] Y. Peng, J. Yang, C. Deck, and R. Willinger, "Finite element modeling of crash test behavior for windshield laminated glass," *International Journal of Impact Engineering,* vol. 57, pp. 27-35, 2013/07/01/ 2013, doi: https://doi.org/10.1016/j.ijimpeng.2013.01.010.

[36] K. Hollstein, X. Yang, and K. Weide-Zaage, "Thermal analysis of the design parameters of a QFN package soldered on a PCB using a simulation approach," *Microelectronics Reliability,* vol. 120, p. 114118, 2021/05/01/ 2021, doi: https://doi.org/10.1016/j.microrel.2021.114118.

[37] W. Kuntjoro, A. M. H. A. Jalil, and J. Mahmud, "Wing Structure Static Analysis using Superelement," *Procedia Engineering,* vol. 41, pp. 1600-1606, 2012/01/01/ 2012, doi: https://doi.org/10.1016/j.proeng.2012.07.356.

[38] C. Suresh, K. Ramesh, and V. Paramaguru, "Aerodynamic performance analysis of a non-planar C-wing using CFD," *Aerospace Science and Technology,* vol. 40, pp. 56-61, 2015/01/01/ 2015, doi: https://doi.org/10.1016/j.ast.2014.10.014.

[39] M. A. Hariri-Ardebili, S. M. Seyed-Kolbadi, V. E. Saouma, J. Salamon, and B. Rajagopalan, "Random finite element method for the seismic analysis of gravity dams," *Engineering Structures,* vol. 171, pp. 405-420, 2018/09/15/ 2018, doi: https://doi.org/10.1016/j.engstruct.2018.05.096.

[40] M. G. Filippi, P. Kuo-Peng, and M. G. Vanti, "Electromagnetic device modeling using a new adaptive wavelet finite element method," *Mathematics and Computers in Simulation,* vol. 172, pp. 111-133, 2020/06/01/ 2020, doi: https://doi.org/10.1016/j.matcom.2019.12.016.

[41] U. Vignesh, D. Mehrotra, S. M. Bhave, R. Katrolia, and S. Sharma, "Finite element analysis of patient-specific TMJ implants to replace bilateral joints with simultaneous correction of facial deformity," *Journal of Oral Biology and Craniofacial Research,* vol. 10, no. 4, pp. 674-679, 2020/10/01/ 2020, doi: https://doi.org/10.1016/j.jobcr.2020.07.013.

[42] Q. Gui, G. Zhang, Y. Chai, and W. Li, "A finite element method with cover functions for underwater acoustic propagation problems," *Ocean Engineering,* p. 110174, 2021/11/22/ 2021, doi: https://doi.org/10.1016/j.oceaneng.2021.110174.

[43] Q. Wang, Z. Li, B. Qin, R. Zhong, and Z. Zhai, "Vibration characteristics of functionally graded corrugated plates by using differential quadrature finite element method," *Composite Structures,* vol. 274, p. 114344, 2021/10/15/ 2021, doi: https://doi.org/10.1016/j.compstruct.2021.114344.

[44] M. Driscoll, "The Impact of the Finite Element Method on Medical Device Design," *Journal of Medical and Biological Engineering,* vol. 39, no. 2, pp. 171-172, 2019/04/01 2019, doi: 10.1007/s40846-018-0428-4.

[45] P. A. Tipler and G. Mosca, *Physics for Scientists and Engineers*. W. H. Freeman, 2007.

[46] José Brandão, José Fecheira, *Estruturas Reticuladas Hiperstáticas* Porto, Portugal: Instituto Superior de Engenharia do Porto, 2011.

[47] H. P. Langtangen and K. A. Mardal, *Introduction to Numerical Methods for Variational Problems*. Springer International Publishing, 2020.

[48] S. Bournival, J.-C. Cuillière, and V. François, "A mesh-geometry based method for coupling 1D and 3D elements," *Advances in Engineering Software,* vol. 41, no. 6, pp. 838-858, 2010/06/01/ 2010, doi: https://doi.org/10.1016/j.advengsoft.2010.02.004.

[49] F. Flager, G. Soremekun, A. Adya, K. Shea, J. Haymaker, and M. Fischer, "Fully Constrained Design: A general and scalable method for discrete member sizing optimization of steel truss structures," *Computers & Structures,* vol. 140, pp. 55-65, 2014/07/30/ 2014, doi: https://doi.org/10.1016/j.compstruc.2014.05.002.

[50] P. Boeraeve, *Introduction to the Finite Element Method (FEM)*. Institut Gramme - Liege: Bélgica: D.I.P, 2010.

[51] A. Andrade, D. Camotim, and P. B. Dinis, "Lateral-torsional buckling of singly symmetric web-tapered thin-walled I-beams: 1D model vs. shell FEA," *Computers & Structures,* vol. 85, no. 17, pp. 1343-1359, 2007/09/01/ 2007, doi: https://doi.org/10.1016/j.compstruc.2006.08.079.

[52] F. Boussuge, J.-C. Léon, S. Hahmann, and L. Fine, "Idealized models for FEA derived from generative modeling processes based on extrusion primitives," *Engineering with Computers,* vol. 31, no. 3, pp. 513-527, 2015/07/01 2015, doi: 10.1007/s00366-014-0382-x.

[53] J. Srinivas, *Stress Analysis and Experimental Techniques: An Introduction*. Alpha Science International, 2011.

[54] R. D. Cook, *CONCEPTS AND APPLICATIONS OF FINITE ELEMENT ANALYSIS, 4TH ED*. Wiley India Pvt. Limited, 2007.

[55] M. A. Bhatti, *Fundamentals Of Finite Element Analysis*. McGraw-Hill Education (India) Pvt Limited, 2005.

[56] I. Koutromanos, *Fundamentals of Finite Element Analysis: Linear Finite Element Analysis*. Wiley, 2018.

[57] W. Chu, P. S. Ho, and W. Li, "An Adaptive Machine Learning Method Based on Finite Element Analysis for Ultra Low-k Chip Package Design," *IEEE Transactions on Components, Packaging and Manufacturing Technology,* vol. 11, no. 9, pp. 1435-1441, 2021, doi: 10.1109/TCPMT.2021.3102891.

[58] G. M. Puri, *Python Scripts for Abaqus: Learn by Example*. Gautam Puri, 2011.

[59] S. Abaqus. "Abaqus Scripting Reference Guide." Simulia Abaqus. http://130.149.89.49:2080/v6.13/books/ker/default.htm (accessed Dezembro 11, 2021).

[60] M. Mallick, A. Chakrabarty, and N. Khutia, "Genetic algorithm based design optimization of crashworthy honeycomb sandwiched panels of AA7075-T651 aluminium alloy for aerospace applications," *Materials Today: Proceedings,* 2021/11/10/ 2021, doi: https://doi.org/10.1016/j.matpr.2021.10.388.

[61] D. S. S. Corp, *Getting Started with ABAQUS: Interactive Edition ; Version 6.8*. Dassault Systèmes, 2008.

[62] I. Abaqus, "ABAQUS Scripting Reference Manual," *Requesting multiple inputs from the user*, 2006. [Online]. Available:

https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/cmd/default.htm?startat=pt02ch06s07.html

[63] M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, *Data Structures and Algorithms in Python*. John Wiley & Sons, Incorporated, 2013.

[64] D. Systèmes. "Mdb object." https://abaqus-docs.mit.edu/2017/English/SIMACAEKERRefMap/simaker-c-mdbpyc.htm (accessed February 2022).

[65] I. Abaqus, "ABAQUS Scripting Reference Manual," *Mdb object*, 2006. [Online]. Available: https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/ker/default.htm?startat=pt01ch27pyo01.html

[66] D. Systèmes. "Repositories." https://abaqus-docs.mit.edu/2017/English/SIMACAECMDRefMap/simacmd-c-intacltypesrepositories.htm (accessed February 2022).

[67] D. Systèmes. "Repository object." https://abaqus-docs.mit.edu/2017/English/SIMACAEKERRefMap/simaker-c-utlrepositorypyc.htm (accessed February 2022).

[68] I. Abaqus, "ABAQUS Scripting Reference Manual," *ConstrainedSketchGeometry object*, 2006. [Online]. Available: https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/ker/default.htm?startat=pt01ch44pyo04.html

[69] I. Abaqus, "ABAQUS Scripting Reference Manual," 2006. [Online]. Available: https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/ker/default.htm

[70] I. Abaqus, "ABAQUS Scripting Reference Manual," *Transform object*, 2006. [Online]. Available: https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/ker/default.htm?startat=pt01ch06pyo10.html

[71] M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, *Data Structures and Algorithms in Java*. Wiley, 2014.

[72] D. Systèmes, *Abaqus 6.12 Scripting Reference Manual*, 2012. [Online]. Available: http://dsk-016-1.fsid.cvut.cz:2080/v6.12/books/cmd/default.htm.

[73] I. Abaqus, "ABAQUS Scripting Reference Manual," *Summary of ABAQUS Scripting Interface changes between Version 6.4 and Version 6.5*, 2006. [Online]. Available: https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/ker/default.htm?startat=pyi03.html

[74] I. Abaqus, "ABAQUS Scripting Reference Manual," *MeshNodeArray object*, 2006. [Online]. Available: https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/ker/default.htm?startat=pt01ch28pyo09.html

[75] I. Abaqus, "ABAQUS Scripting Reference Manual," *Edge object*, 2006. [Online]. Available: https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/ker/default.htm?startat=pt01ch06pyo03.html

[76] I. Abaqus, "ABAQUS Scripting Reference Manual," *Face object*, 2006. [Online]. Available:

https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/ker/default.htm?startat=pt01ch06pyo05.html

[77] I. Abaqus, "ABAQUS Scripting Reference Manual," *Feature object*, 2006. [Online]. Available: https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/ker/default.htm?startat=pt01ch18pyo01.html

[78] F. Alghadari, T. Turmudi, and T. Herman, "The application of vector concepts on two skew lines," *Journal of Physics: Conference Series,* vol. 948, p. 012030, 01/01 2018, doi: 10.1088/1742-6596/948/1/012030.

[79] *Plastics — Determination of tensile properties — Part 1: General principles*, EN ISO 527-1, 1993.

[80] *Plastics — Determination of tensile properties — Part 2: Test conditions for moulding and extrusion plastics*, EN ISO 527-2, 1993.

[81] I. Abaqus, "ABAQUS Scripting Reference Manual," *SurfaceToSurfaceContactStd object*, 2006. [Online]. Available: https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/ker/default.htm?startat=pt01ch22pyo48.html

[82] I. Abaqus, "ABAQUS Scripting Reference Manual," *ContactExp object*, 2006. [Online]. Available: https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/ker/default.htm?startat=pt01ch22pyo16.html

[83] I. Abaqus, "ABAQUS Scripting Reference Manual," *ModelJob object*, 2006. [Online]. Available: https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/ker/default.htm?startat=pt01ch23pyo04.html

[84] P. Hazel, *The Exim SMTP Mail Server: Official Guide to Release 4*. UIT Cambridge, 2007.

[85] D. Systèmes. "Saving a user-defined view." https://abaqus-docs.mit.edu/2017/English/SIMACAECAERefMap/simacae-t-viwuserviewsavebtn.htm (accessed February 2022).

[86] I. Abaqus, "ABAQUS Scripting Reference Manual," *Session object*, 2006. [Online]. Available: https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/ker/default.htm?startat=pt01ch43pyo01.html

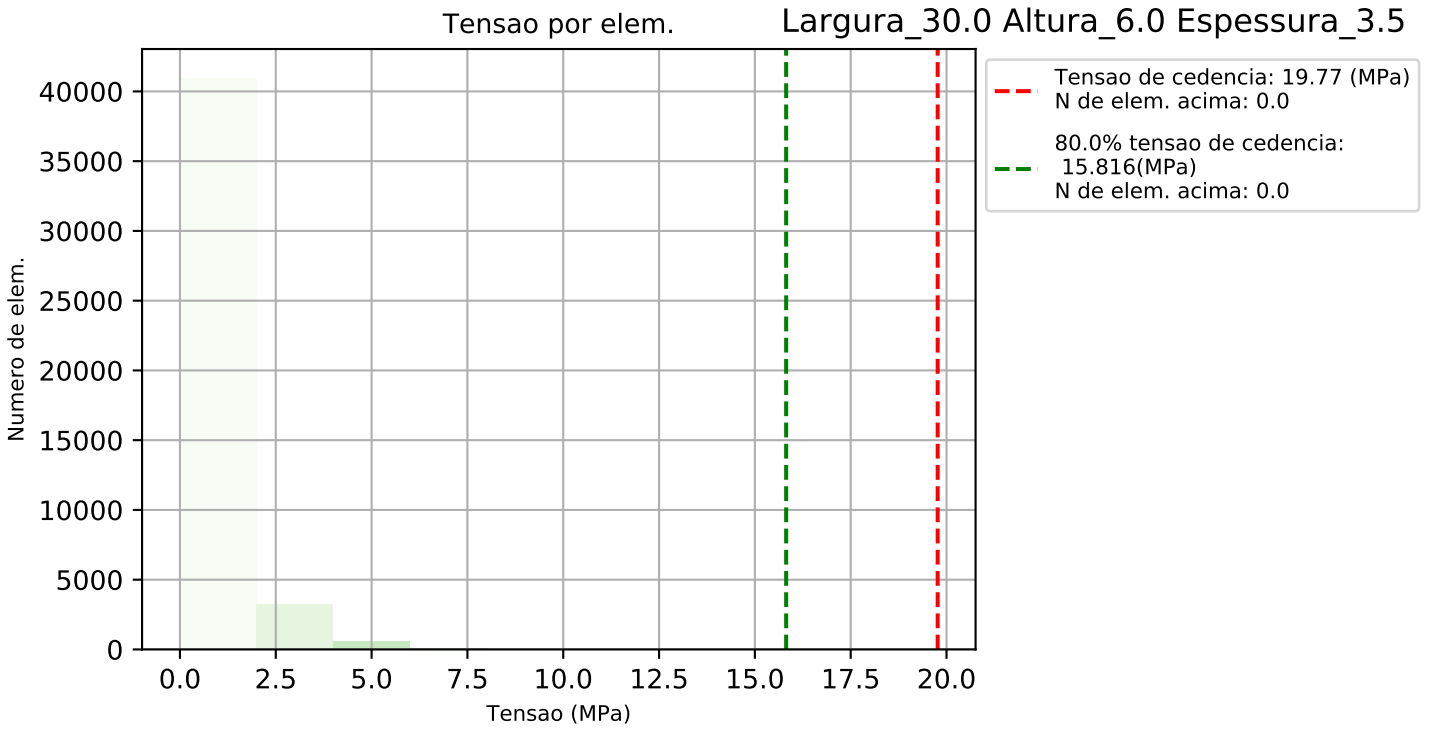
[87] I. Abaqus, "ABAQUS Scripting Reference Manual," *SurfaceToSurfaceContactExp object*, 2006. [Online]. Available: https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/ker/default.htm?startat=pt01ch22pyo47.html

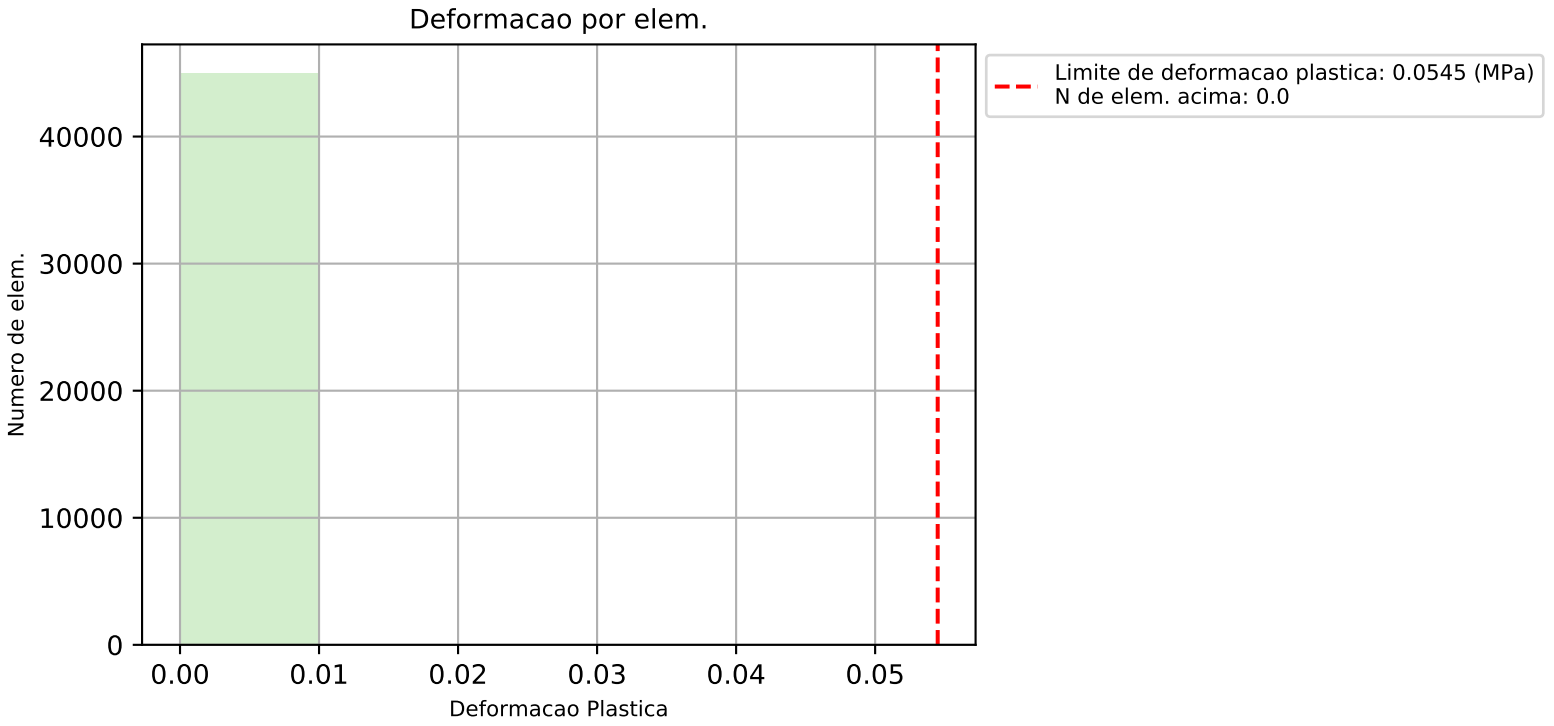
[88] I. Abaqus, "ABAQUS Scripting Reference Manual," *MeshElement object*, 2006. [Online]. Available: https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/ker/default.htm?startat=pt01ch28pyo05.html

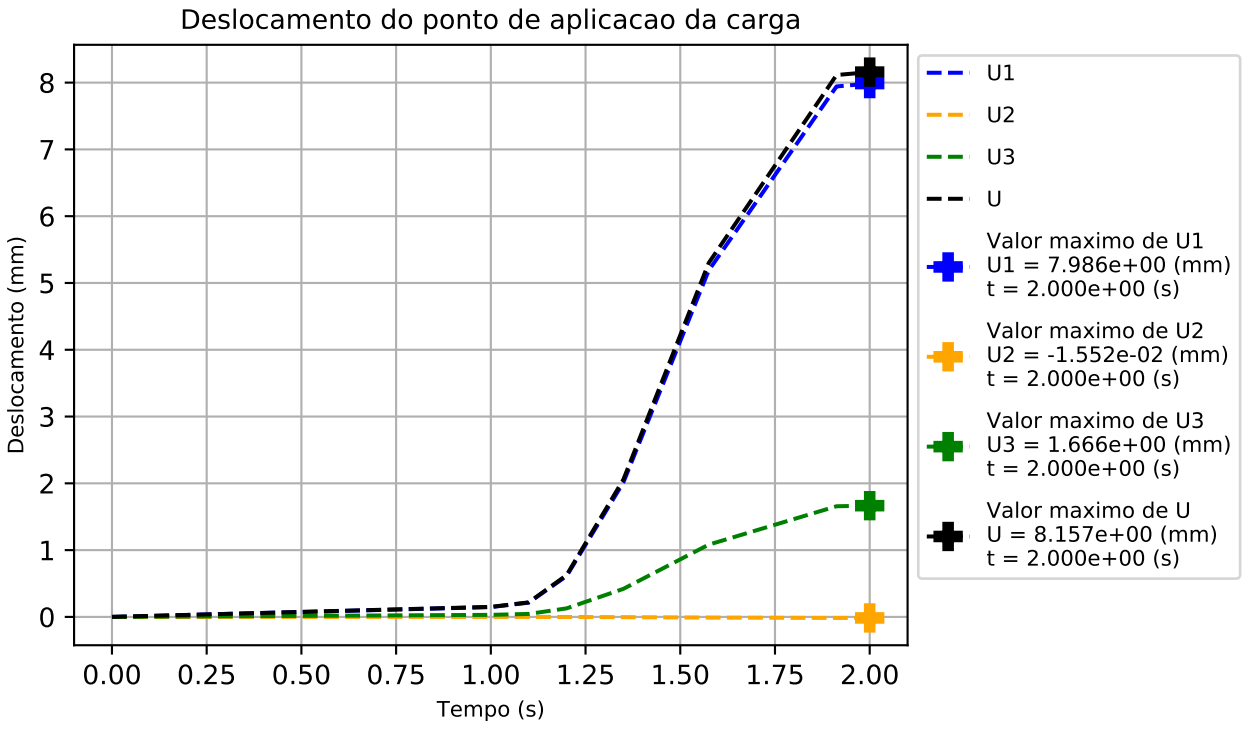
ANEXOS

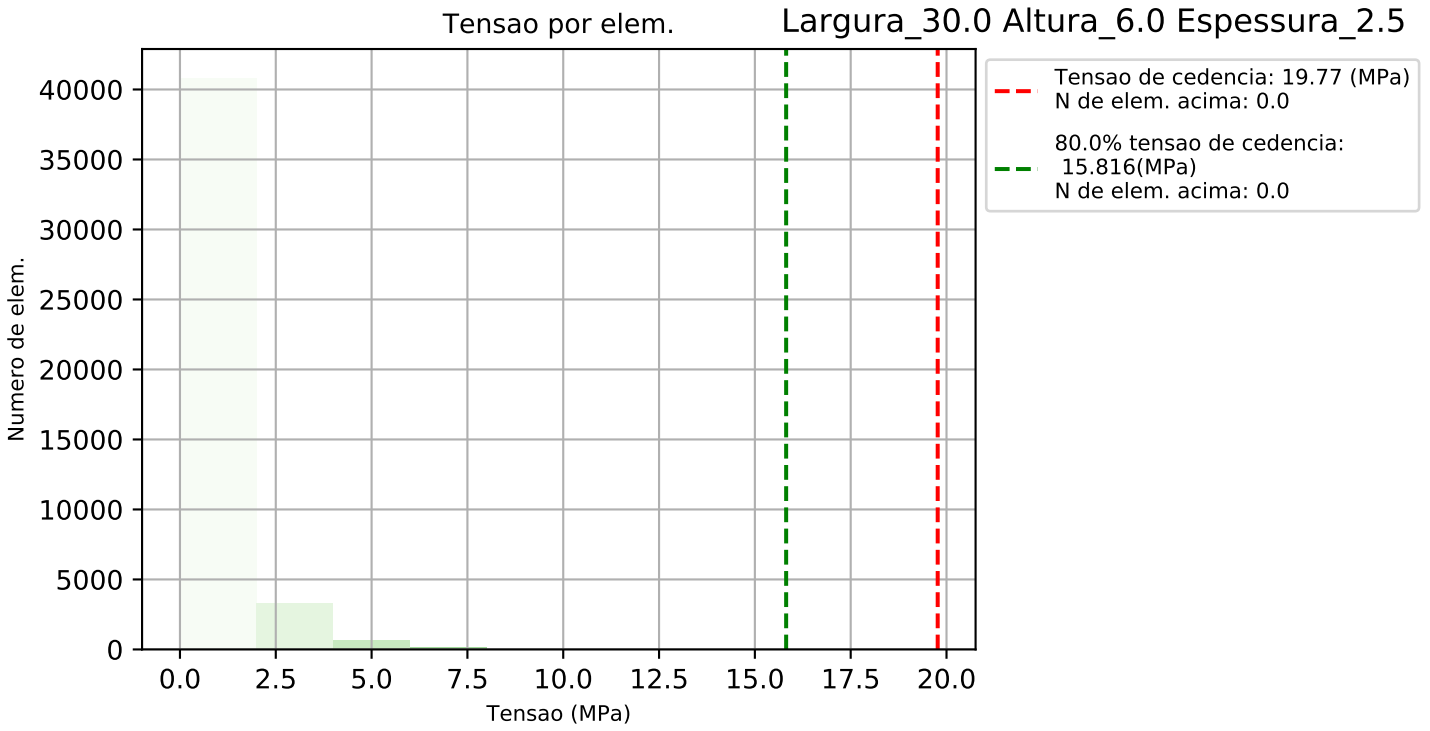
6 ANEXOS

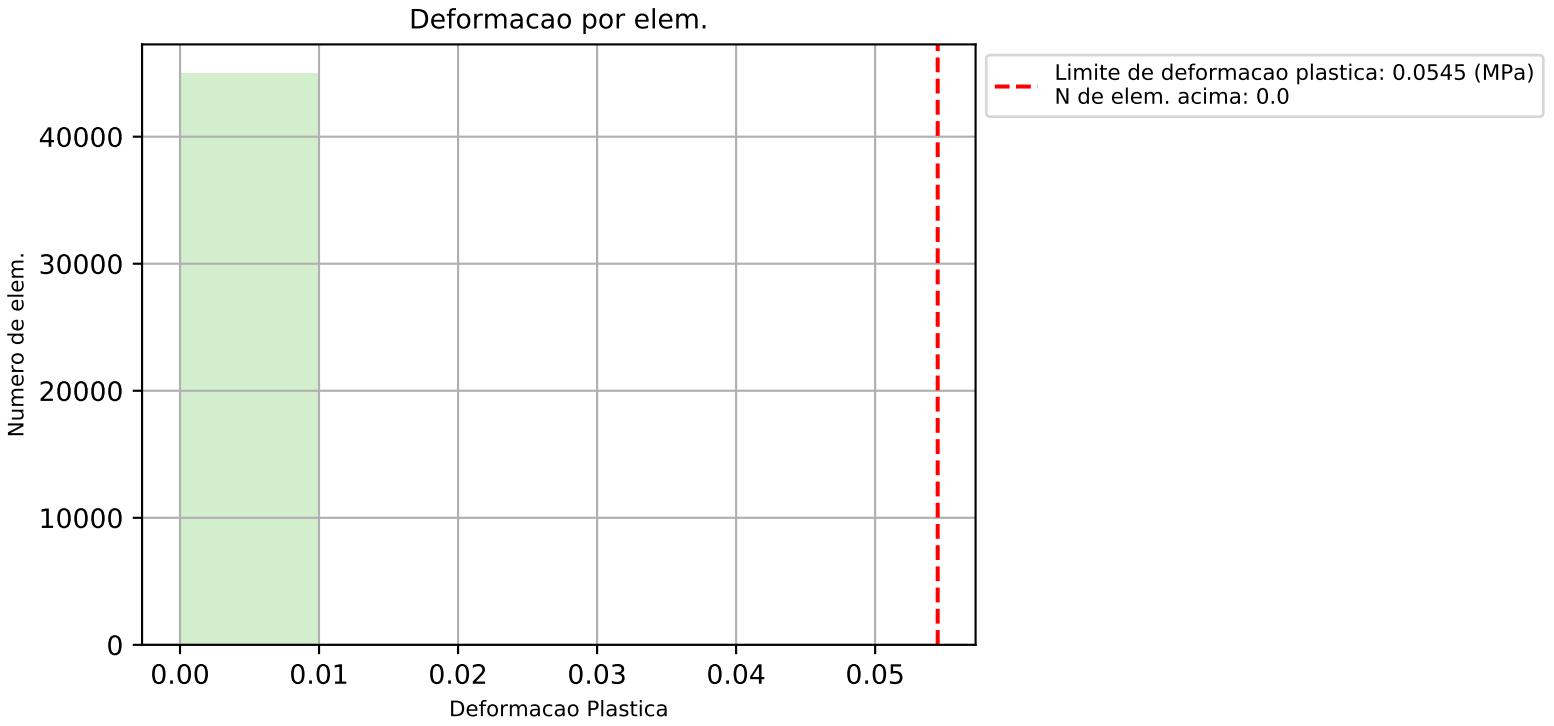
6.1 Livro de resultados do primeiro estudo dos frisos

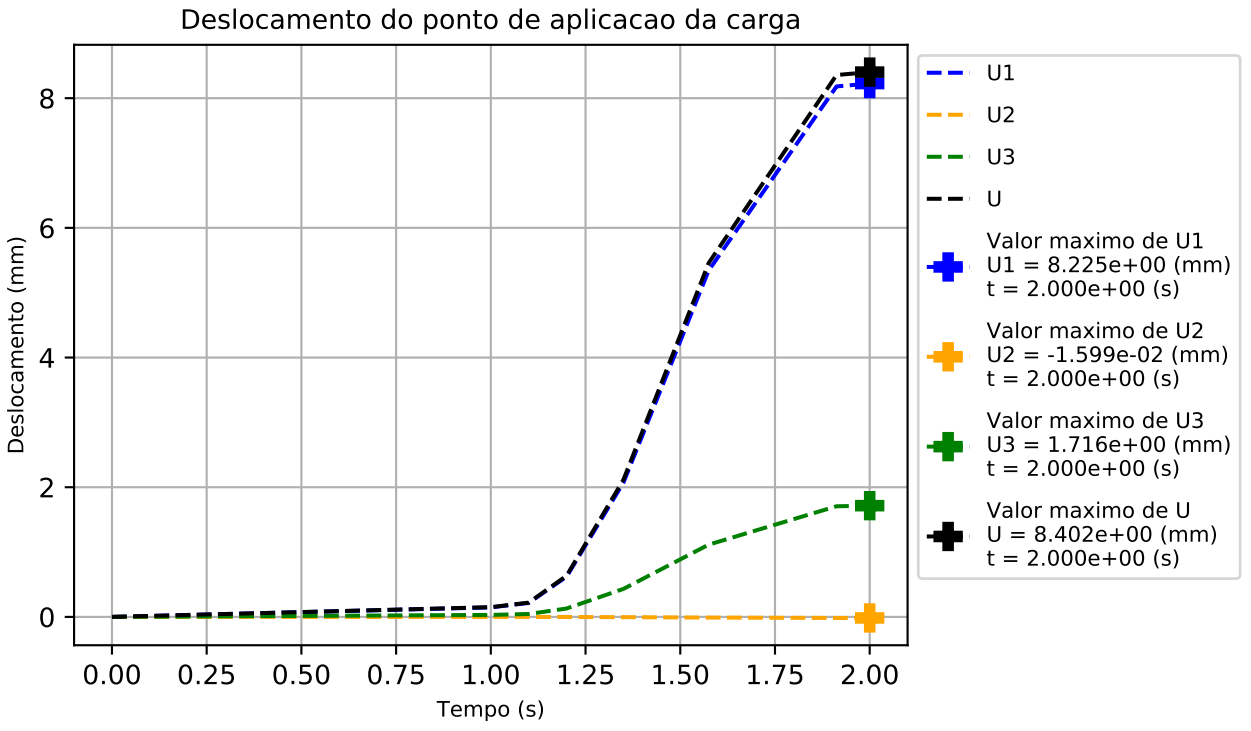

Tensao por elem.
Largura_30.0 Altura_6.0 Espessura_3.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

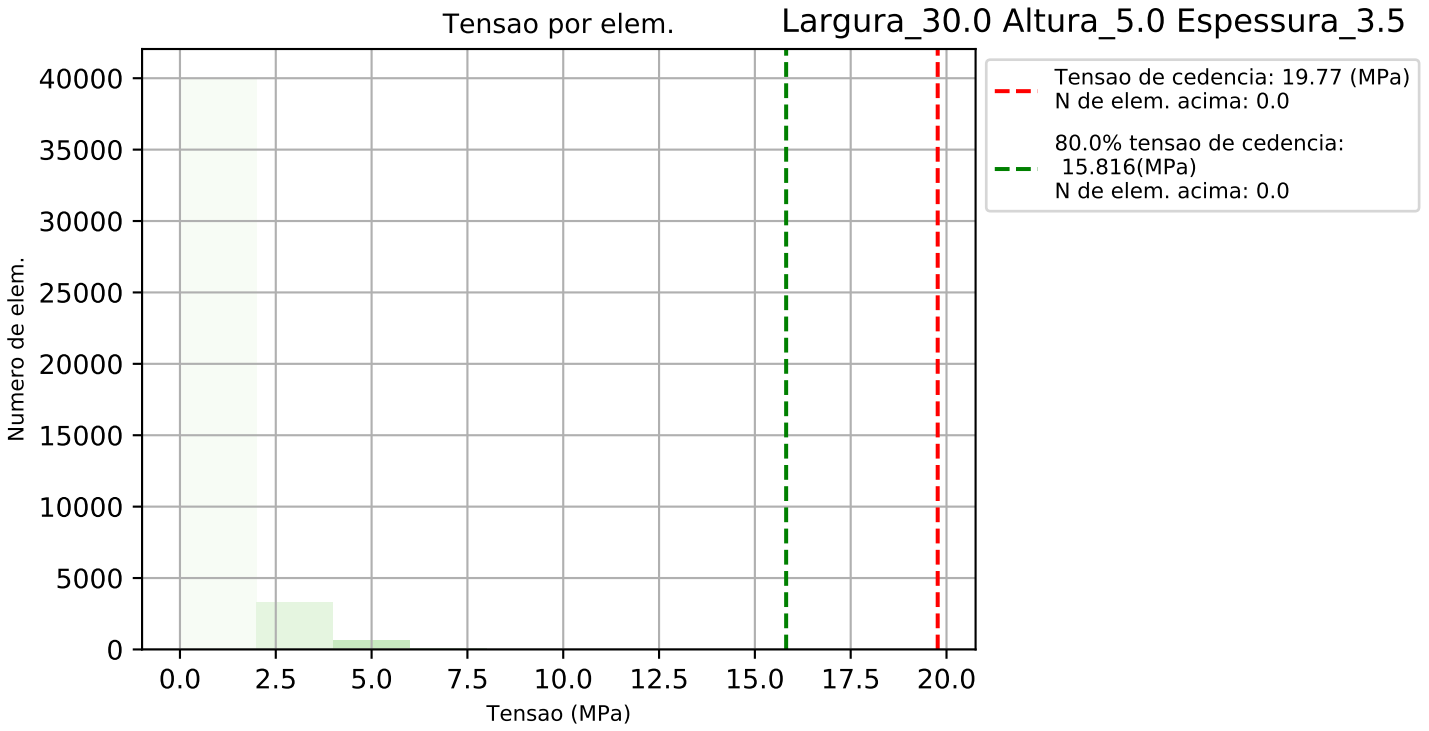

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

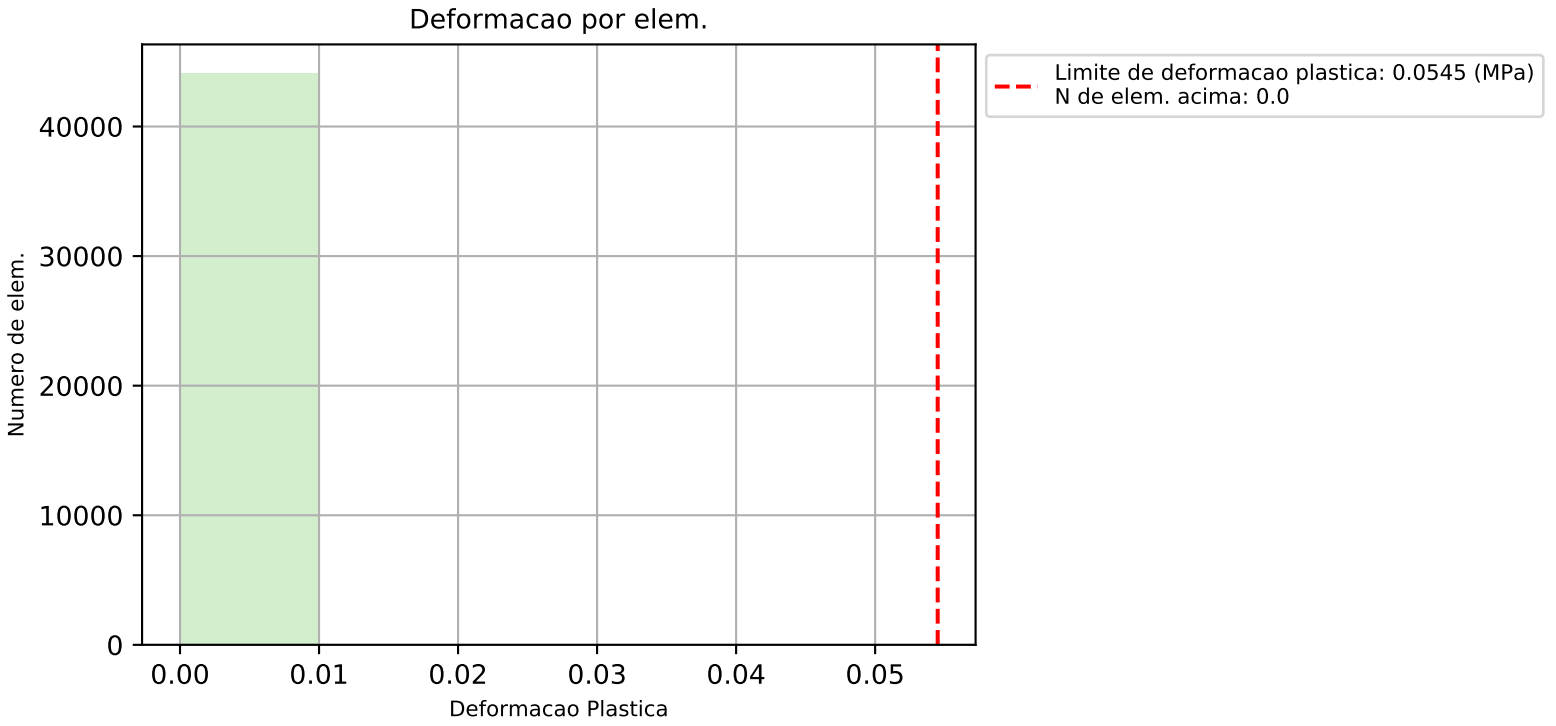

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 7.986e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.552e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.666e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.157e+00 (mm)
t = 2.000e+00 (s)

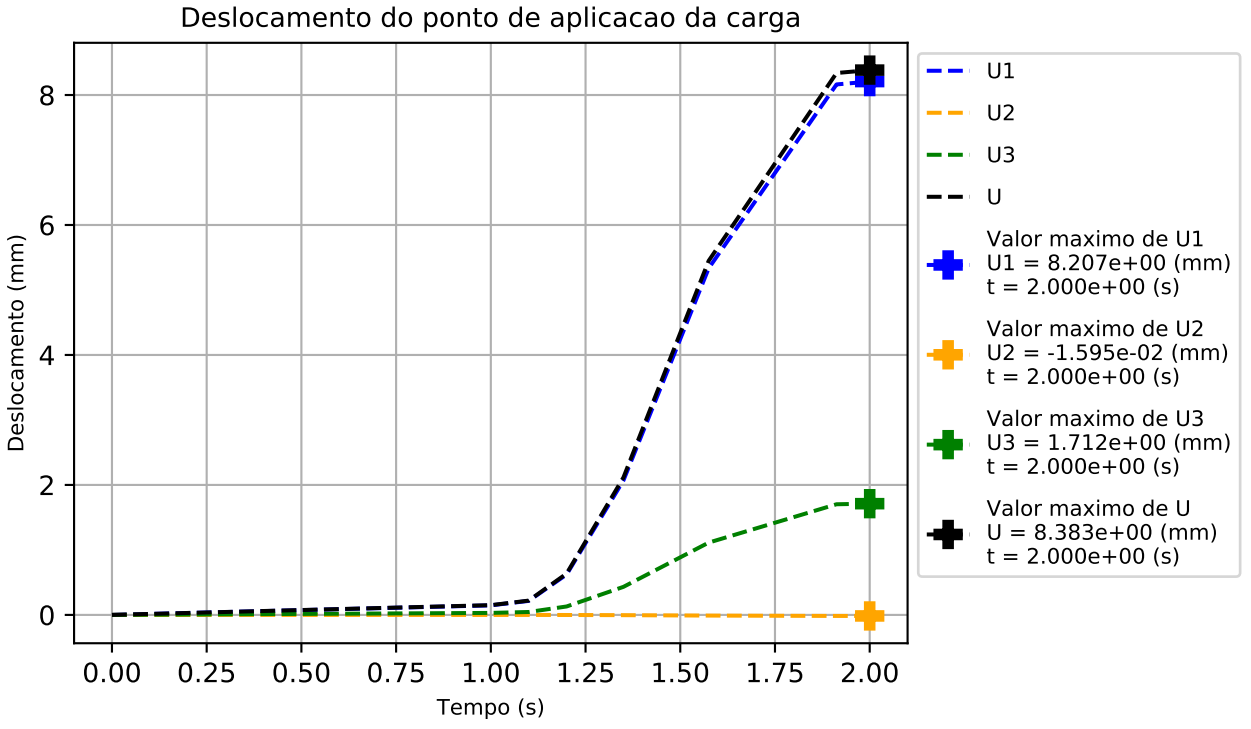

Tensao por elem.
Largura_30.0 Altura_6.0 Espessura_2.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

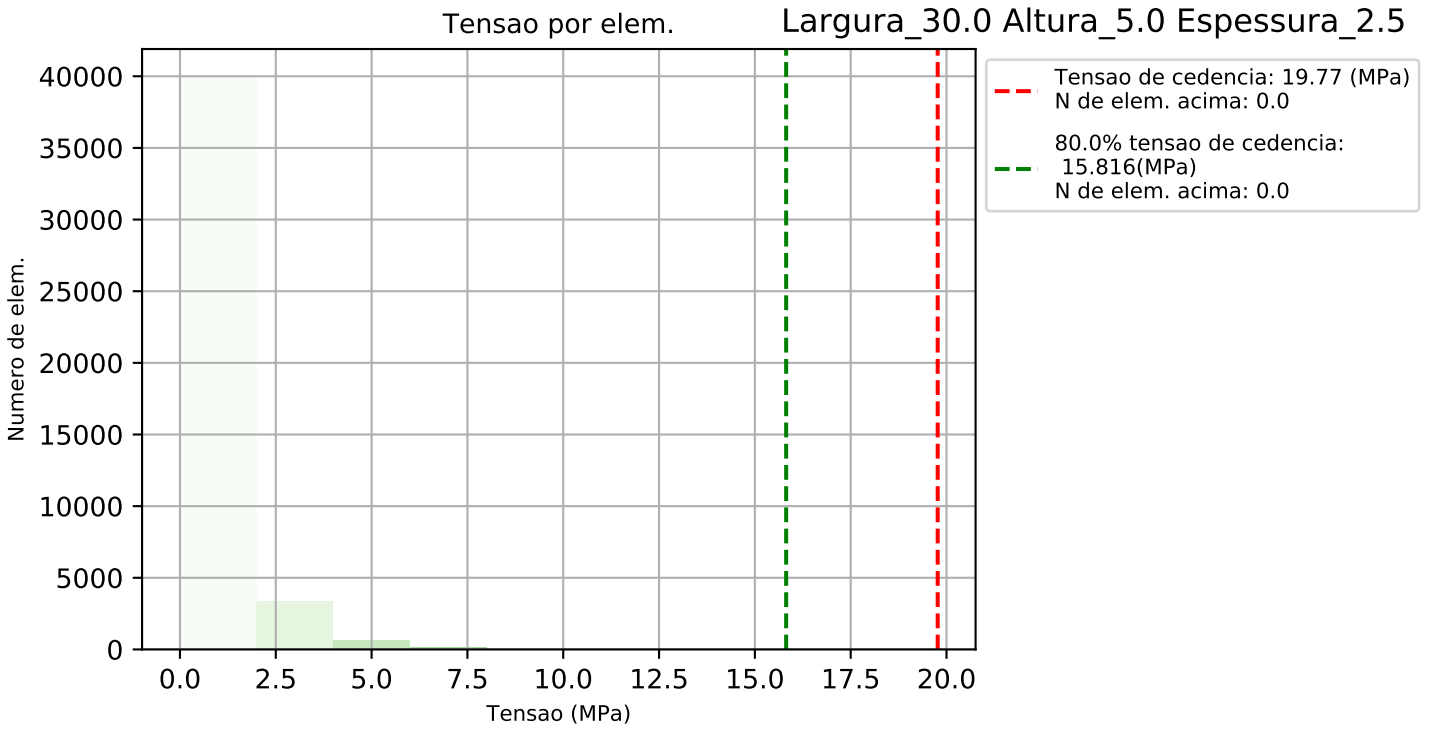

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

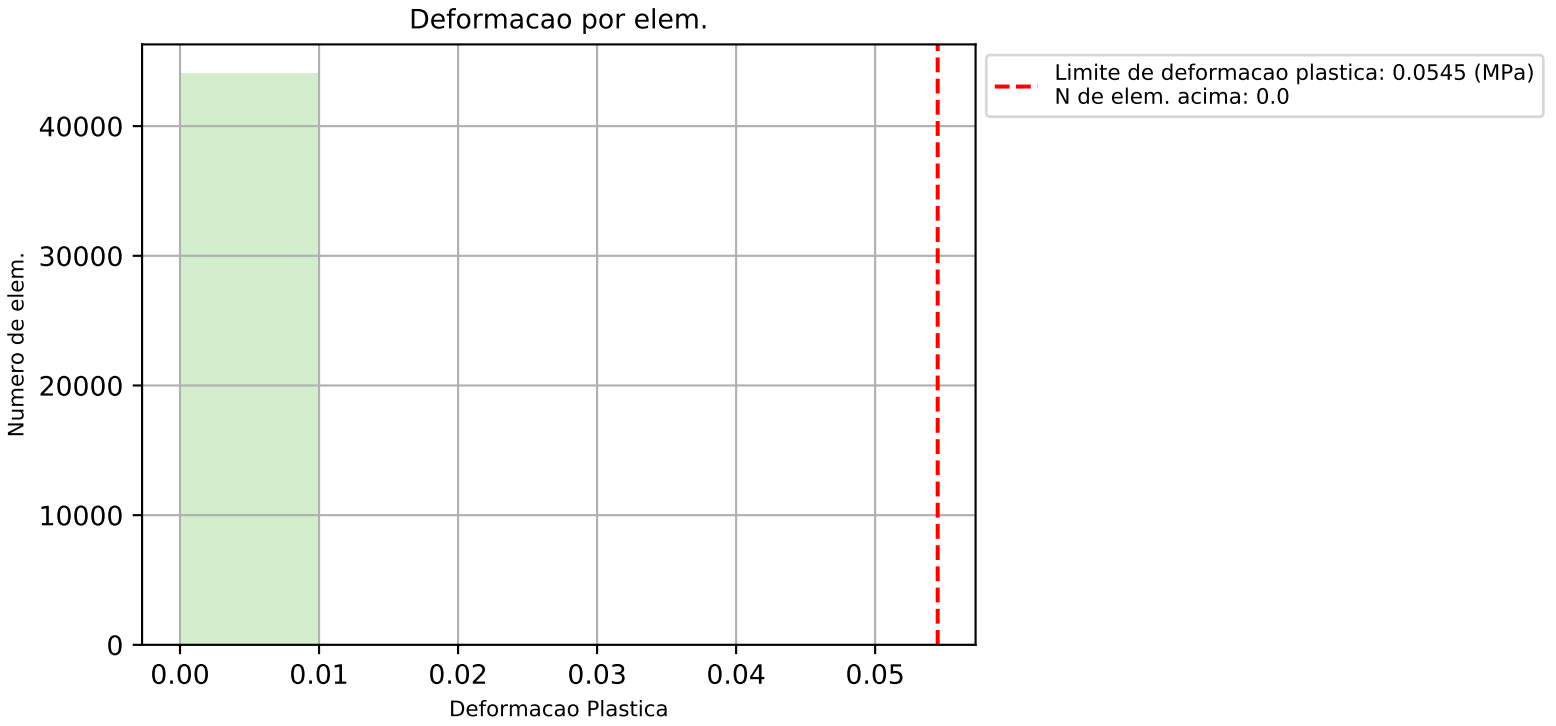

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.225e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.599e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.716e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.402e+00 (mm)
t = 2.000e+00 (s)

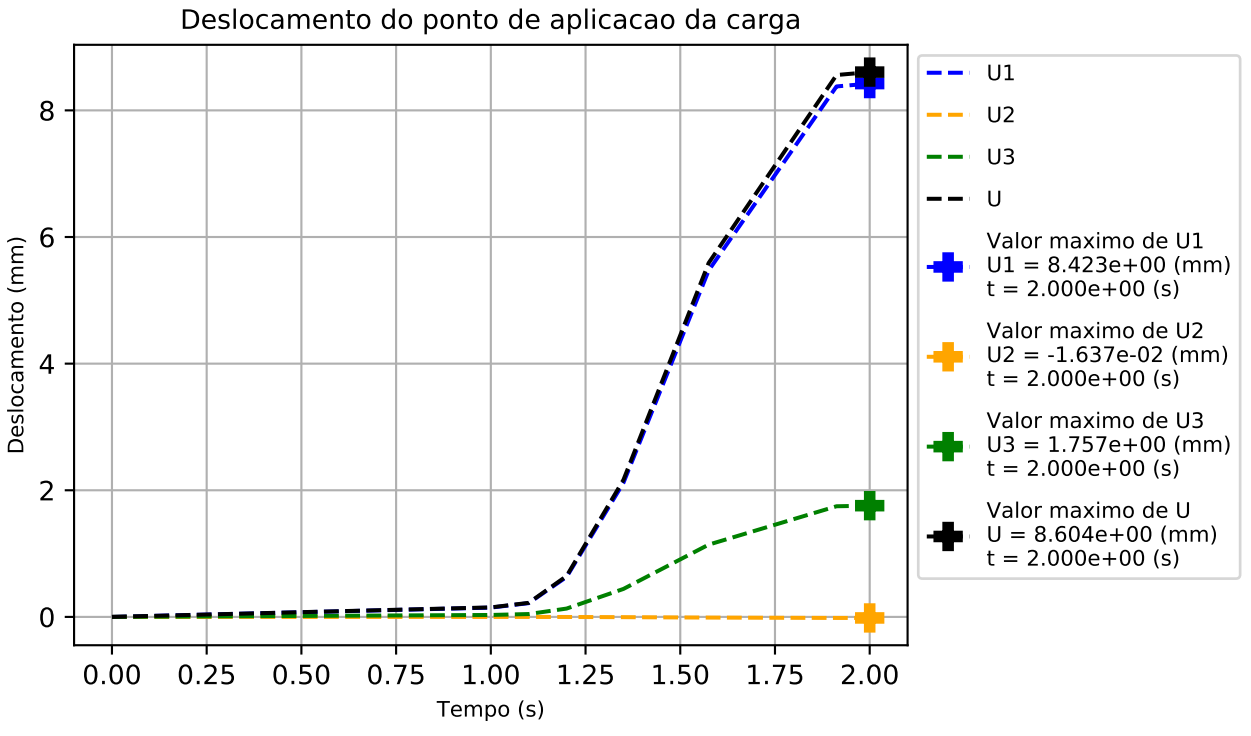

Tensao por elem.
Largura_30.0 Altura_5.0 Espessura_3.5
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

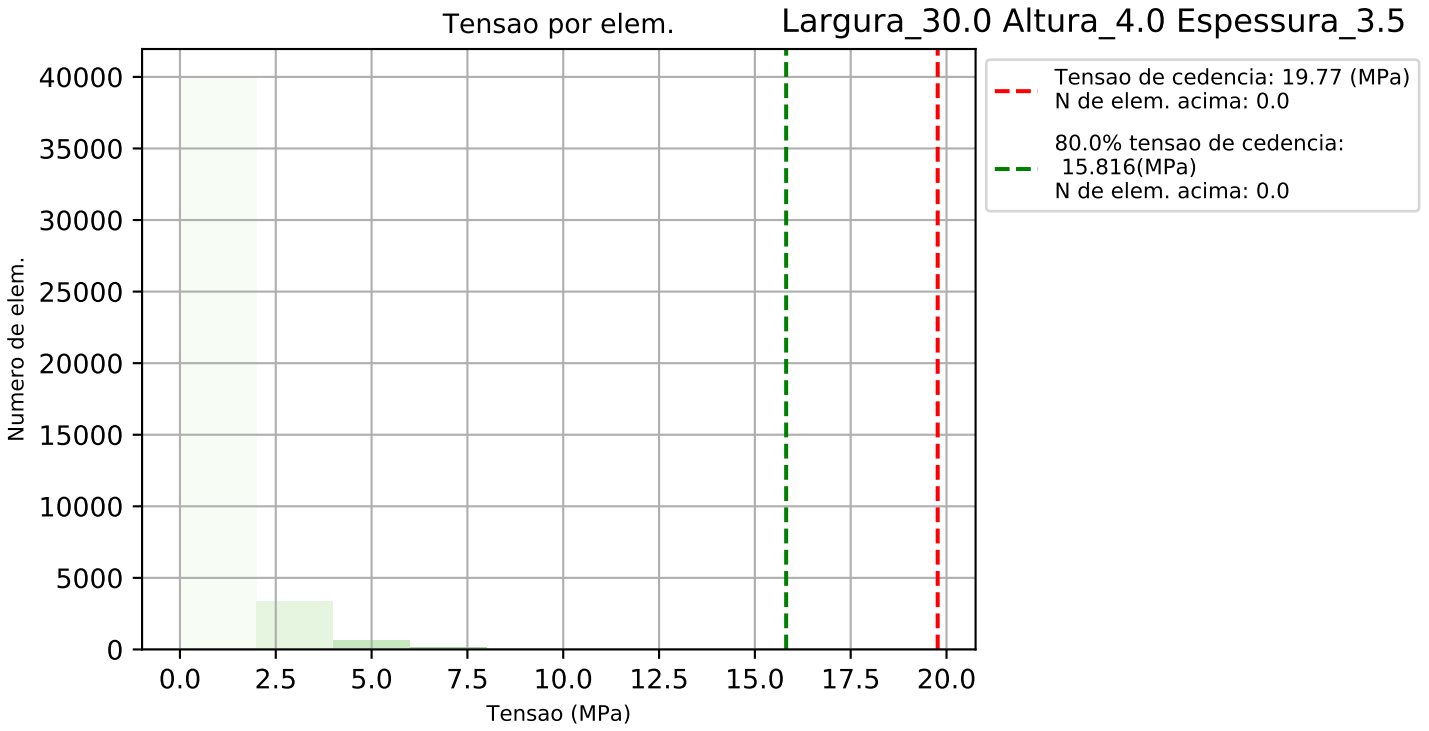

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

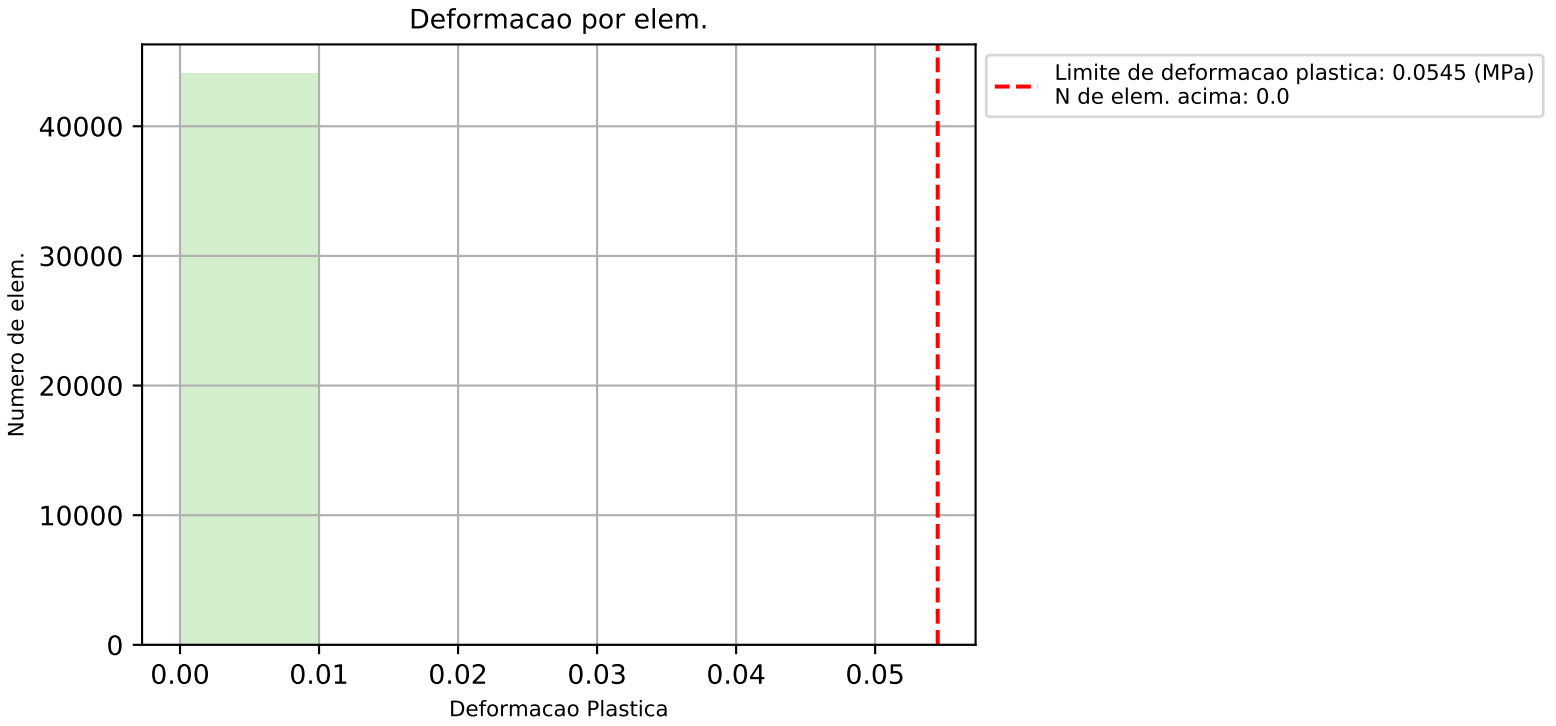

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.207e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.595e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.712e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.383e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

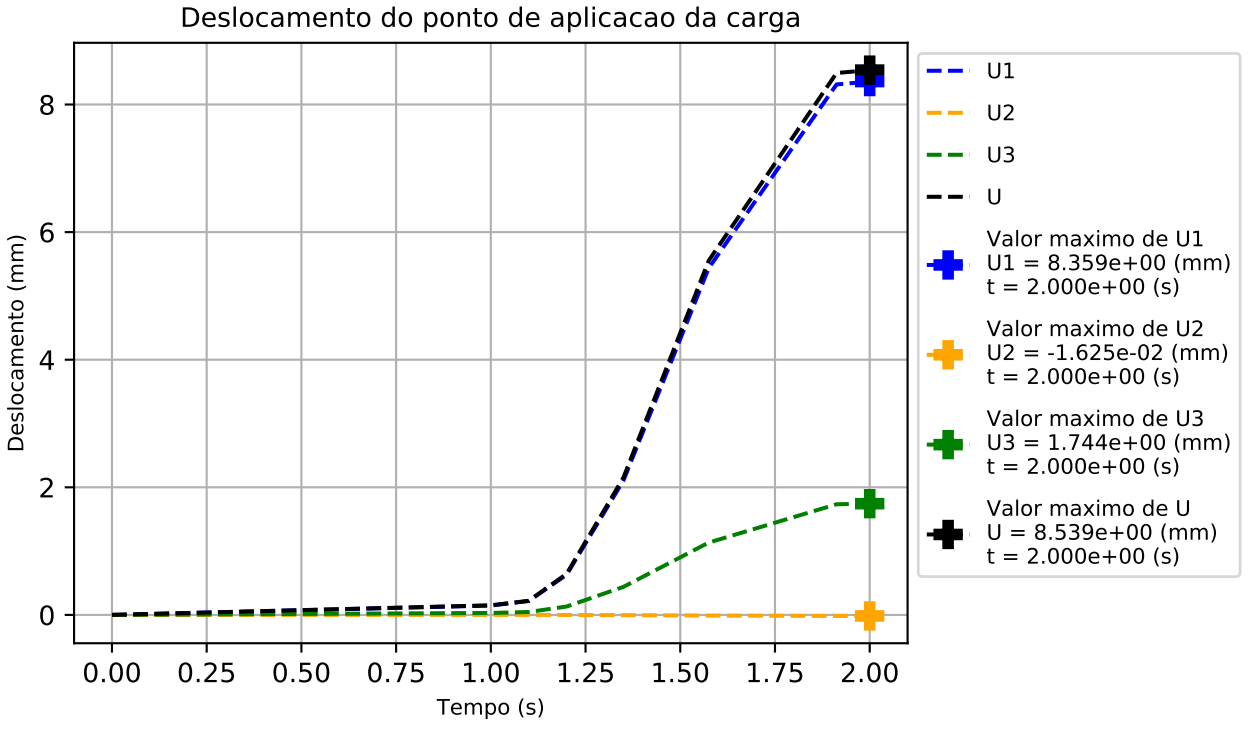

Tensao por elem.
Largura_30.0 Altura_5.0 Espessura_2.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

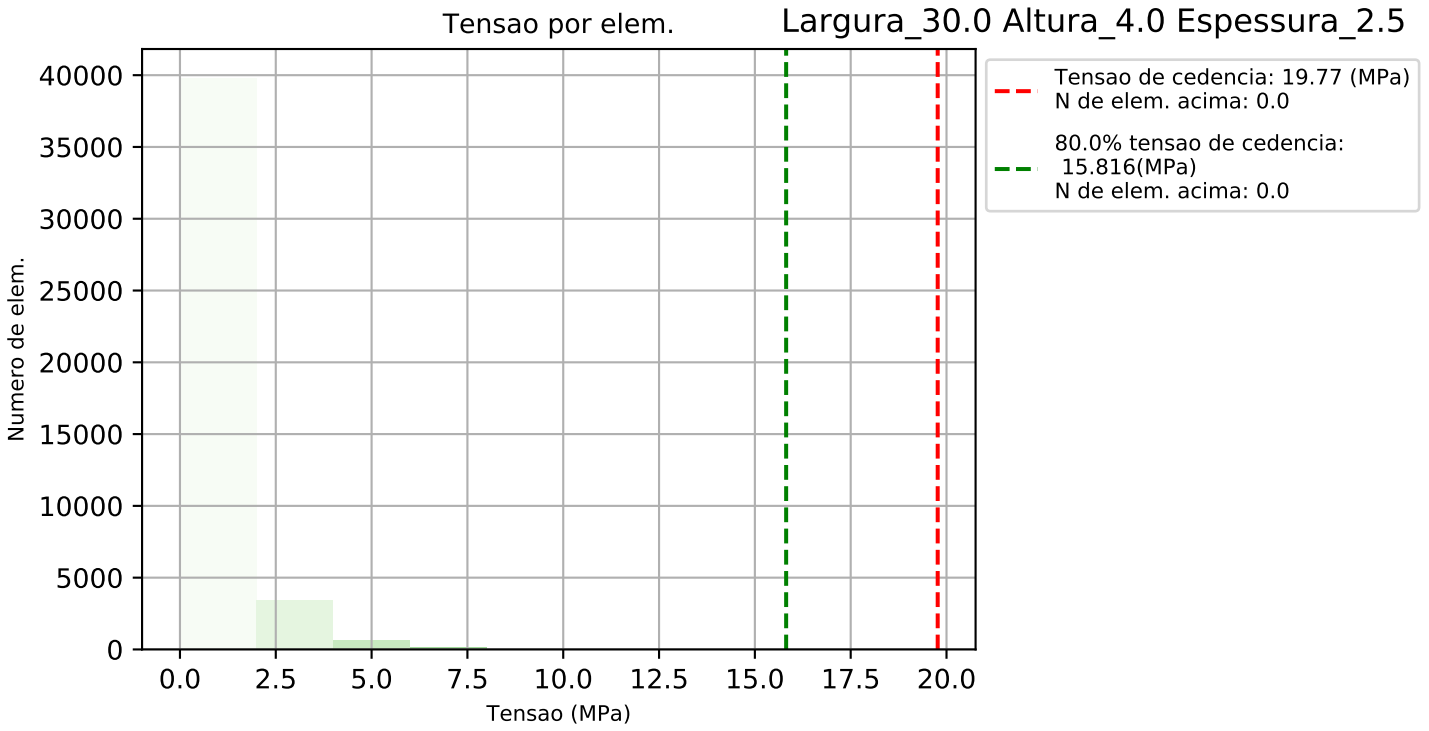

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

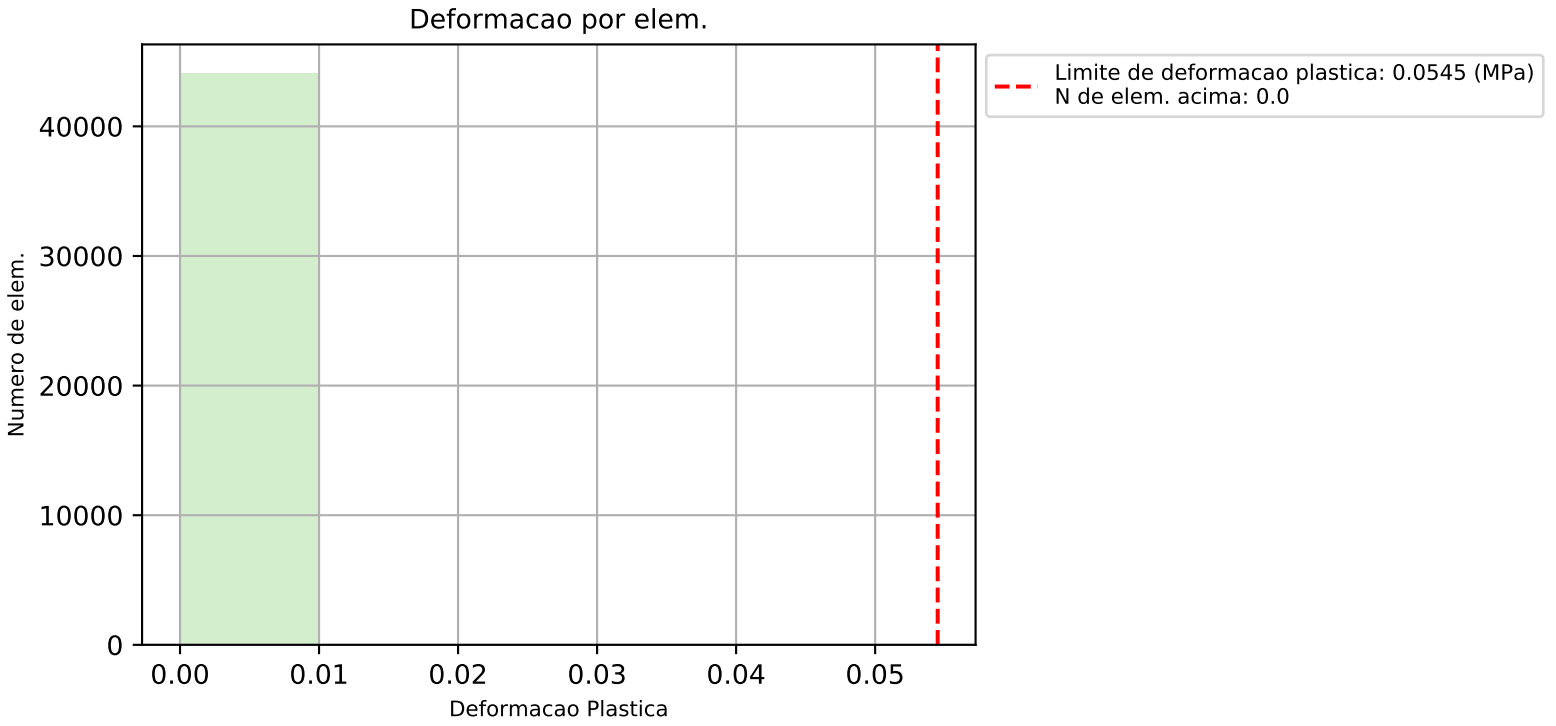

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.423e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.637e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.757e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.604e+00 (mm)
t = 2.000e+00 (s)

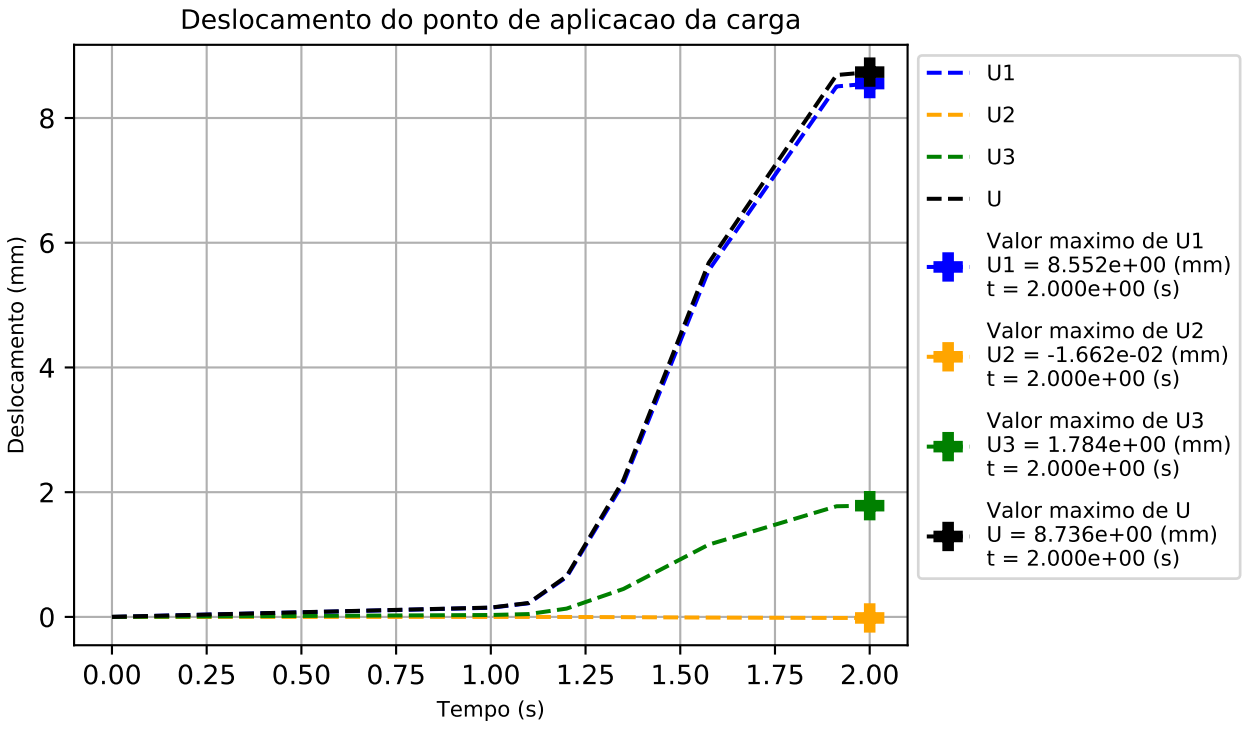

Tensao por elem.
Largura_30.0 Altura_4.0 Espessura_3.5
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

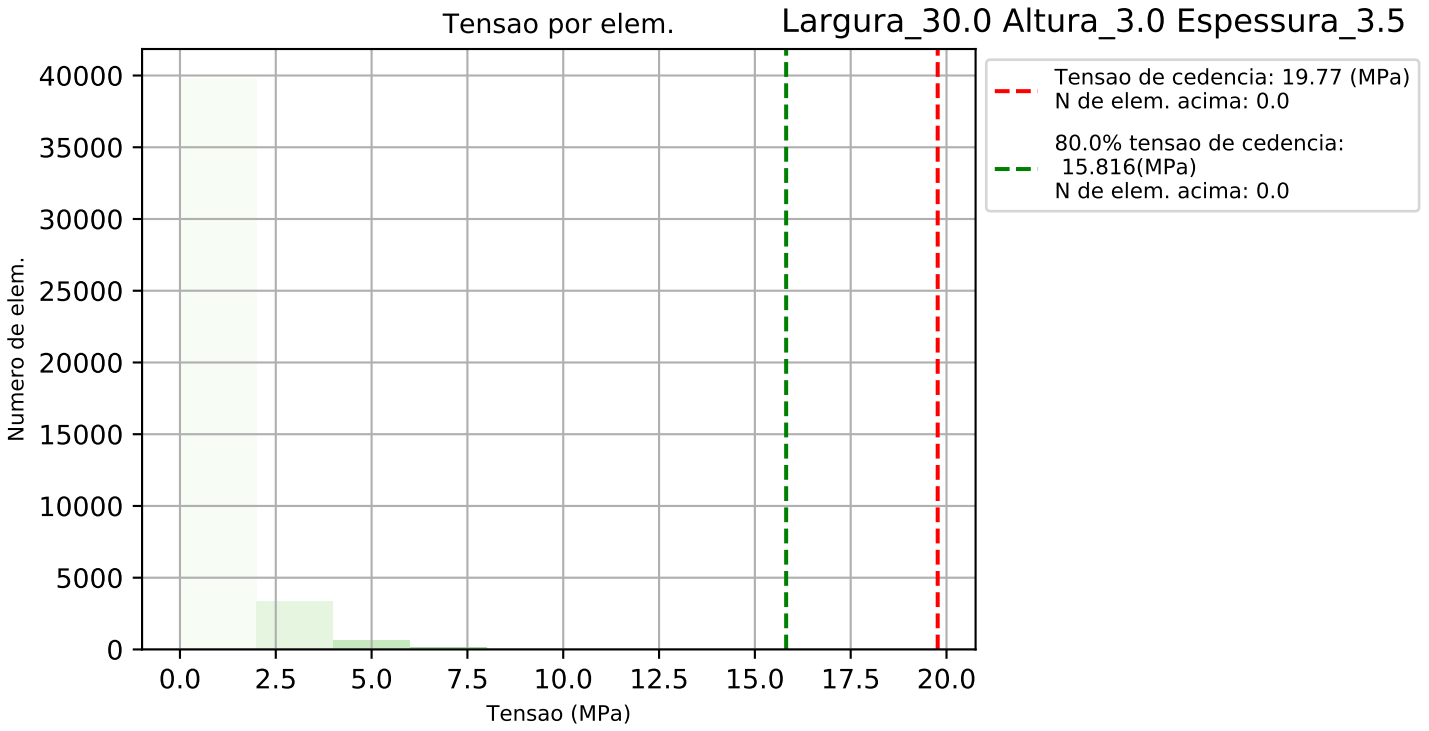

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

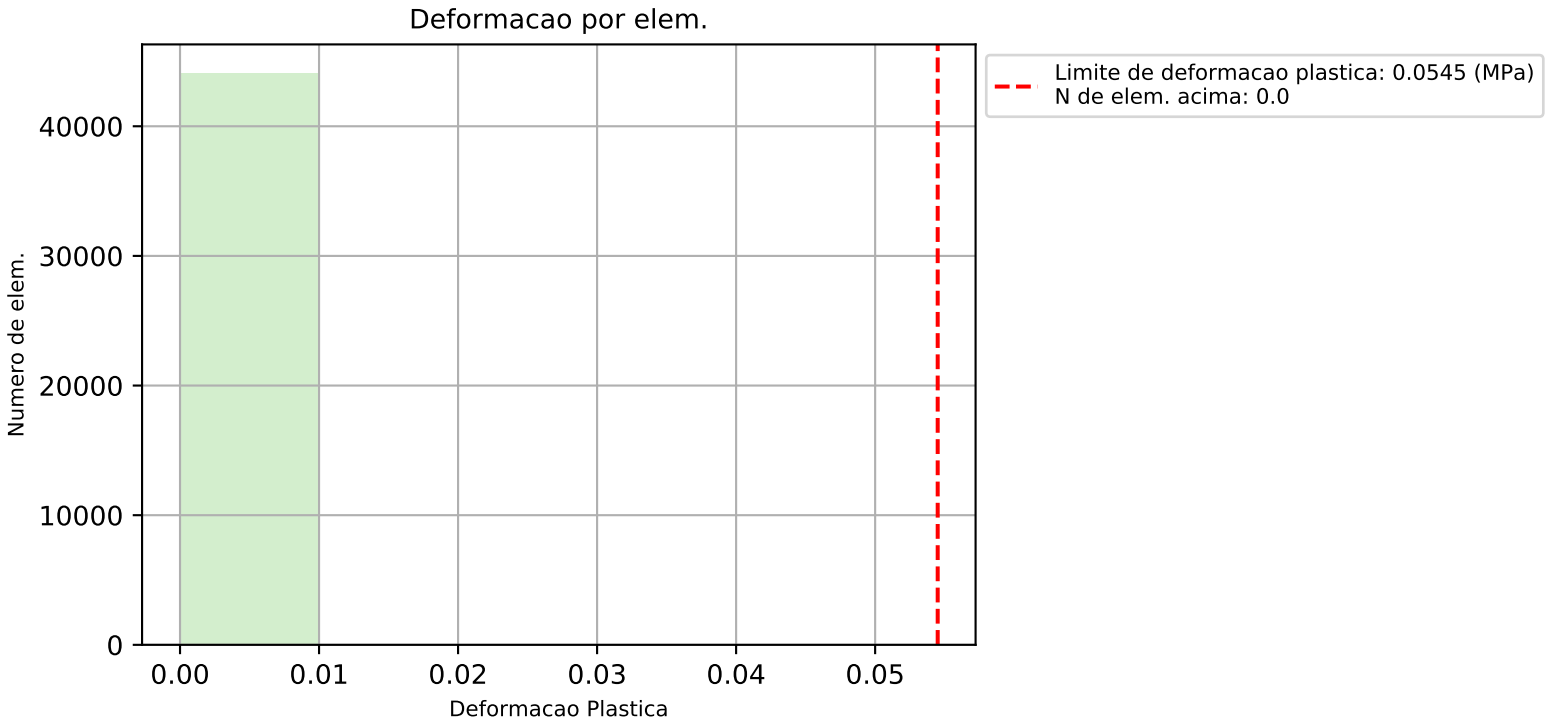

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.359e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.625e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.744e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.539e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

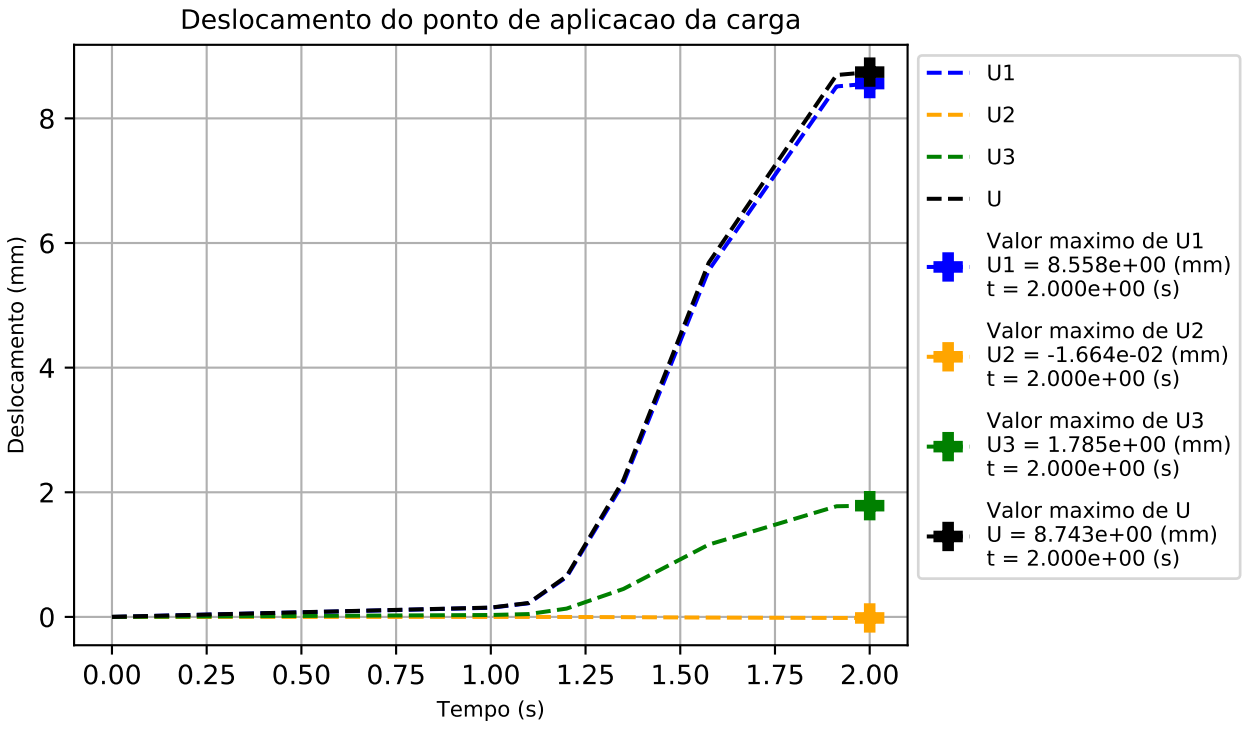

Tensao por elem.
Largura_30.0 Altura_4.0 Espessura_2.5
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

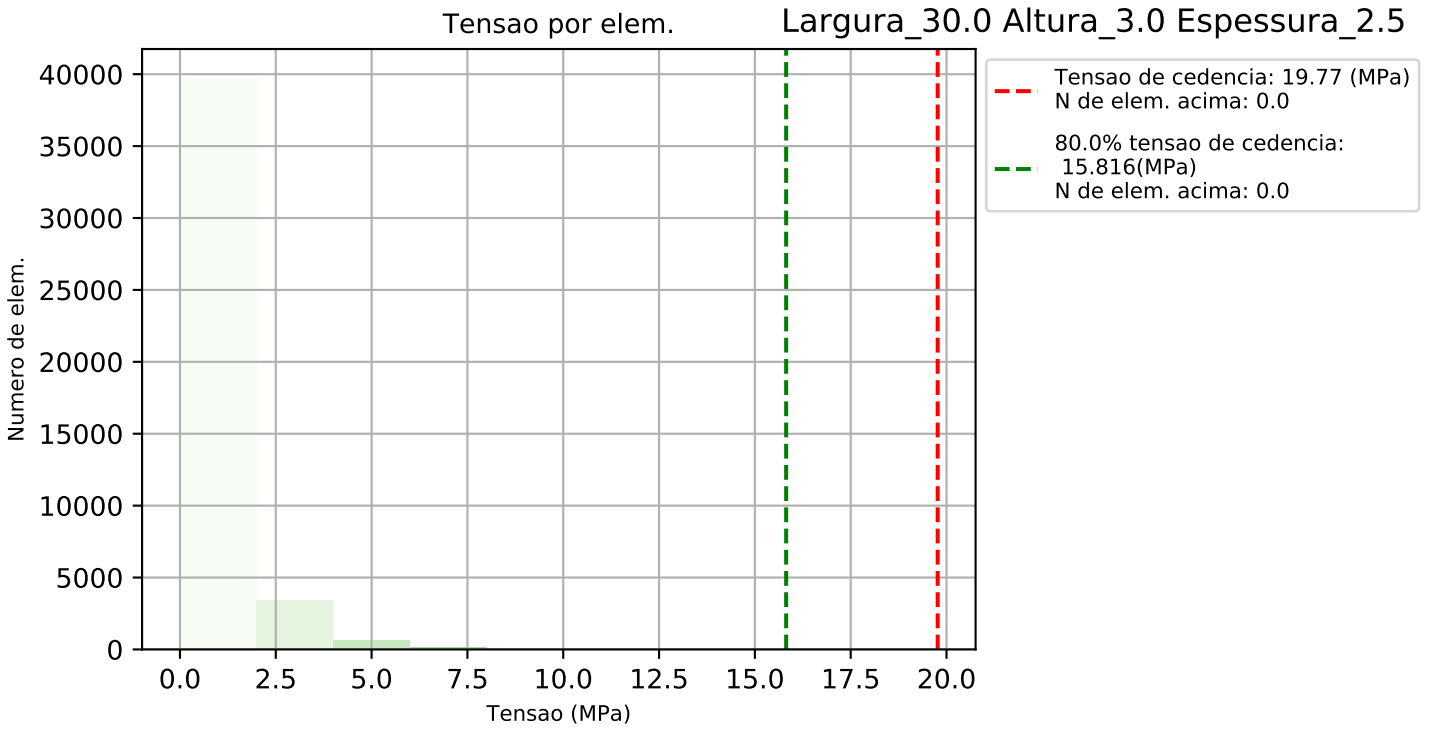

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

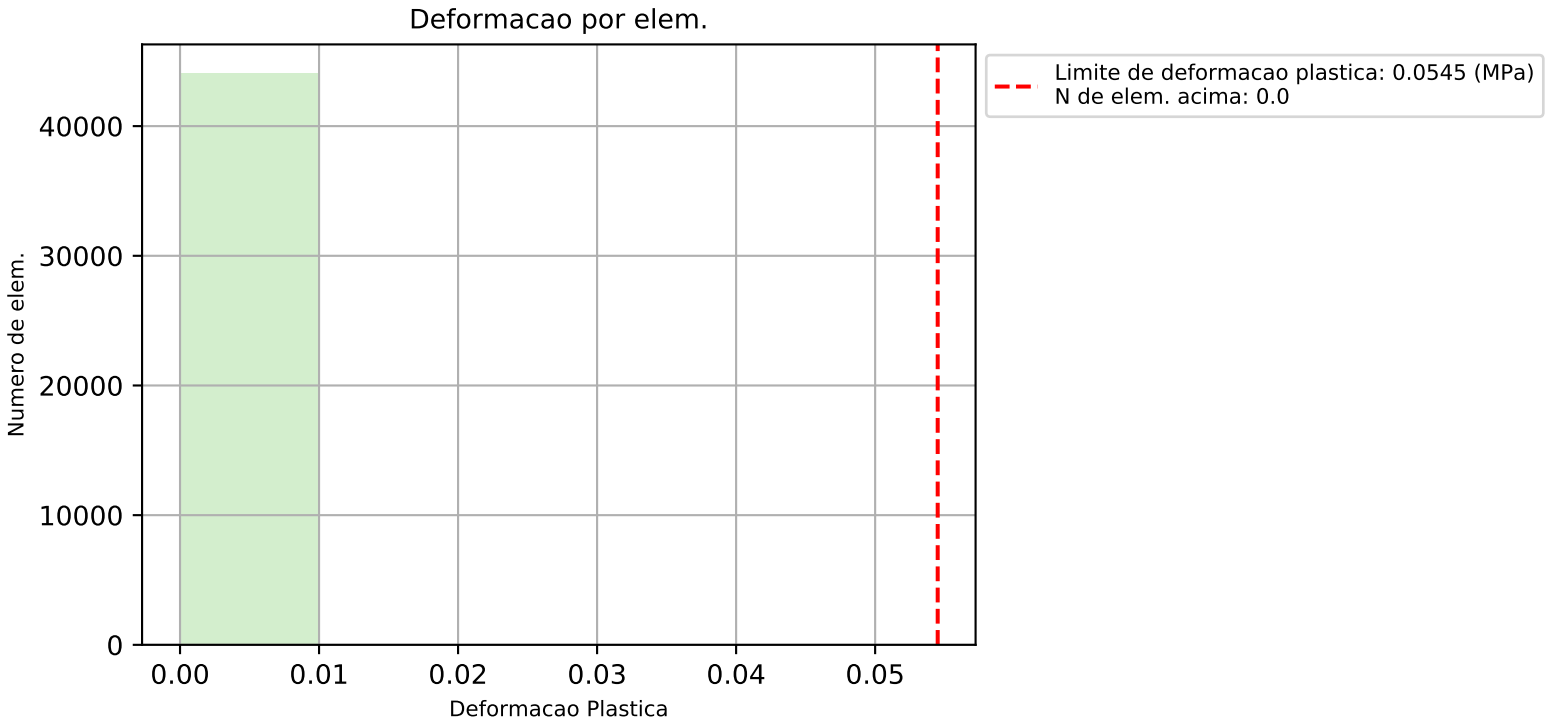

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.552e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.662e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.784e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.736e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

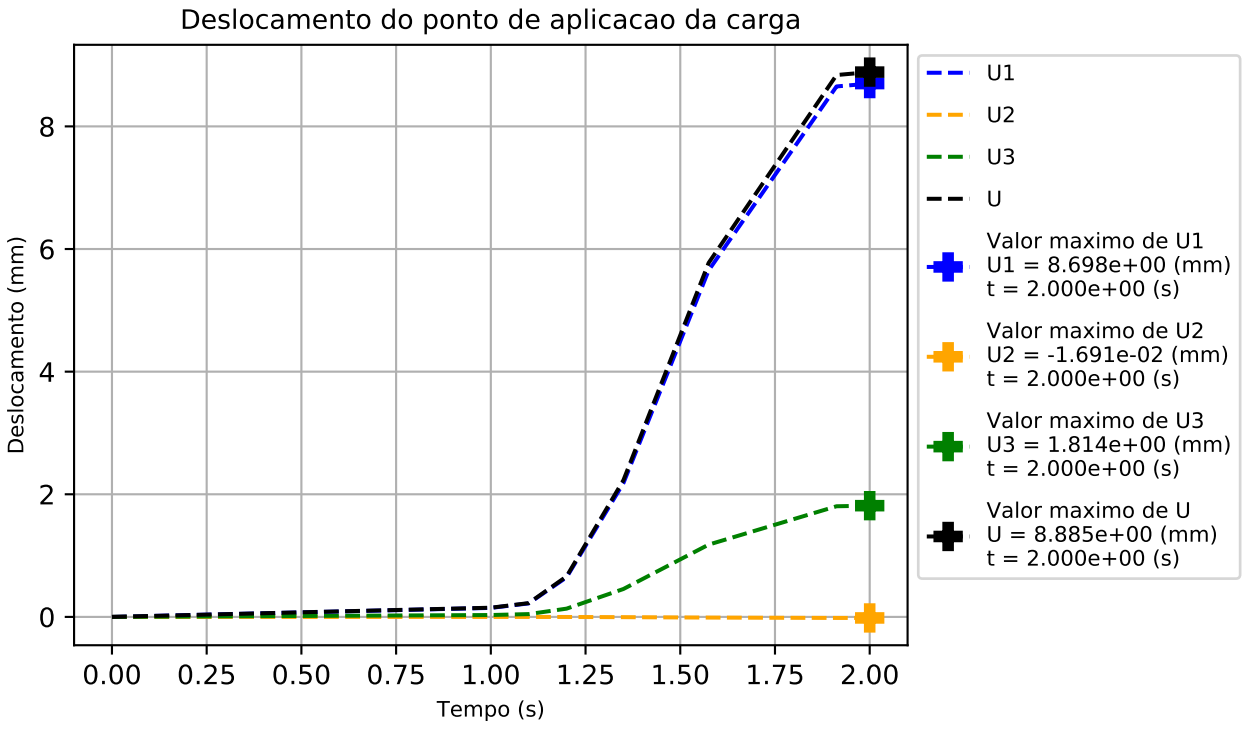

Tensao por elem.
Largura_30.0 Altura_3.0 Espessura_3.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

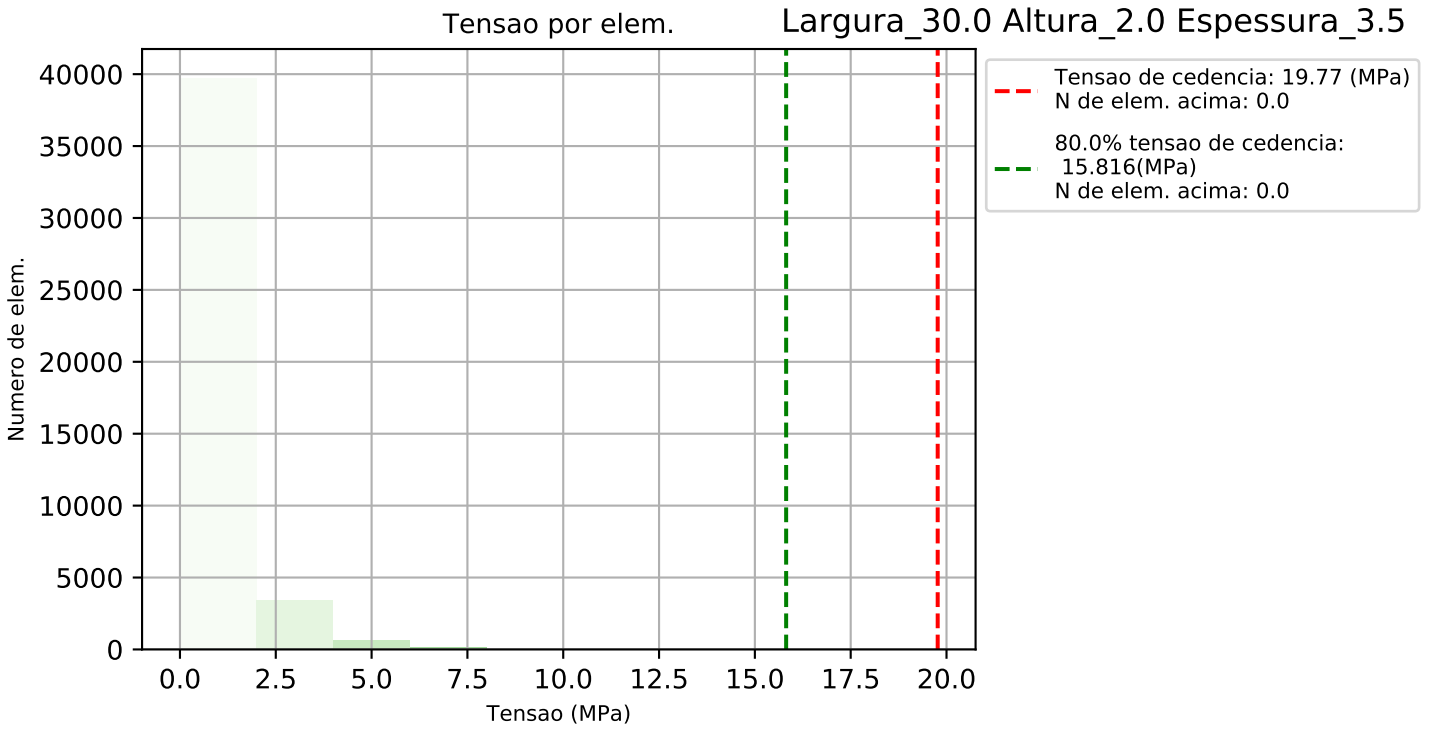

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

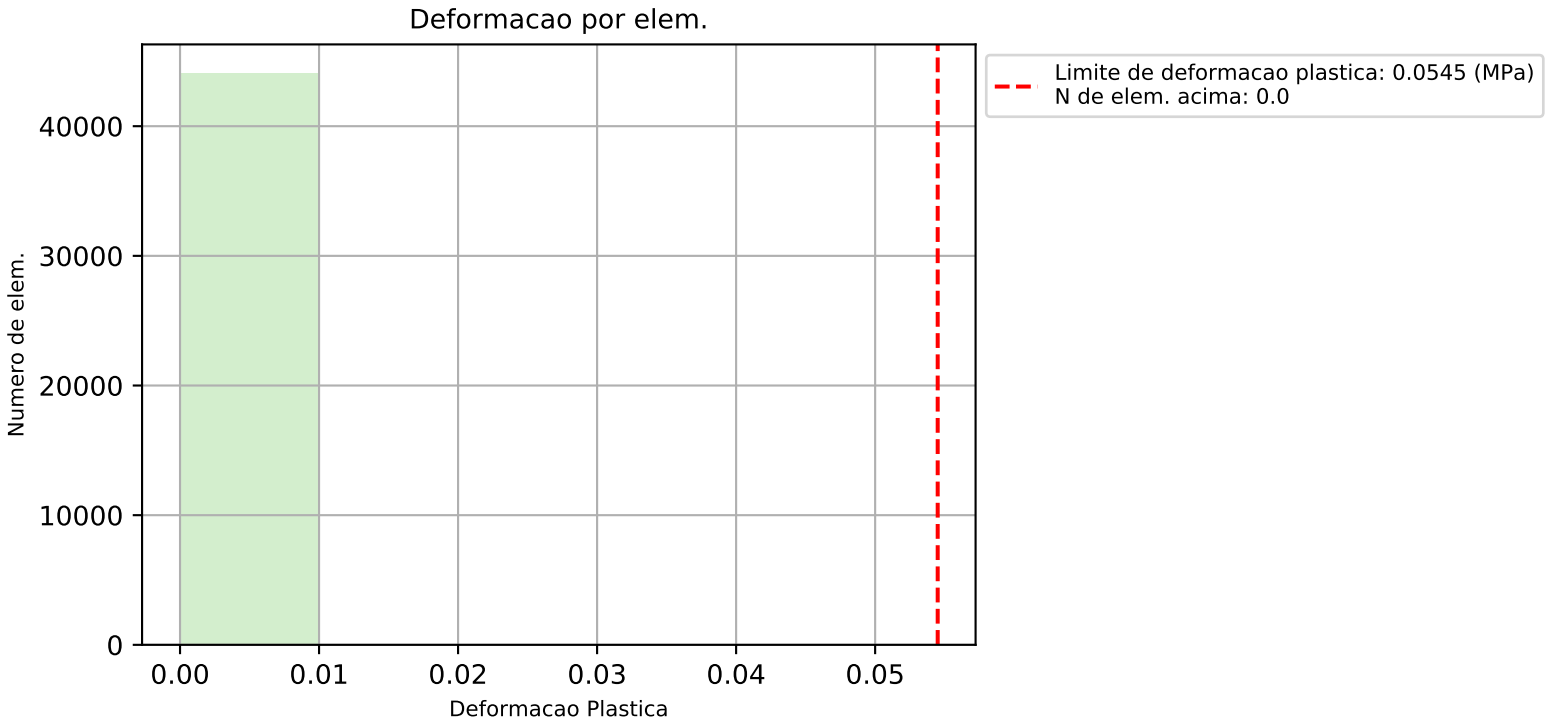

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.558e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.664e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.785e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.743e+00 (mm)
t = 2.000e+00 (s)

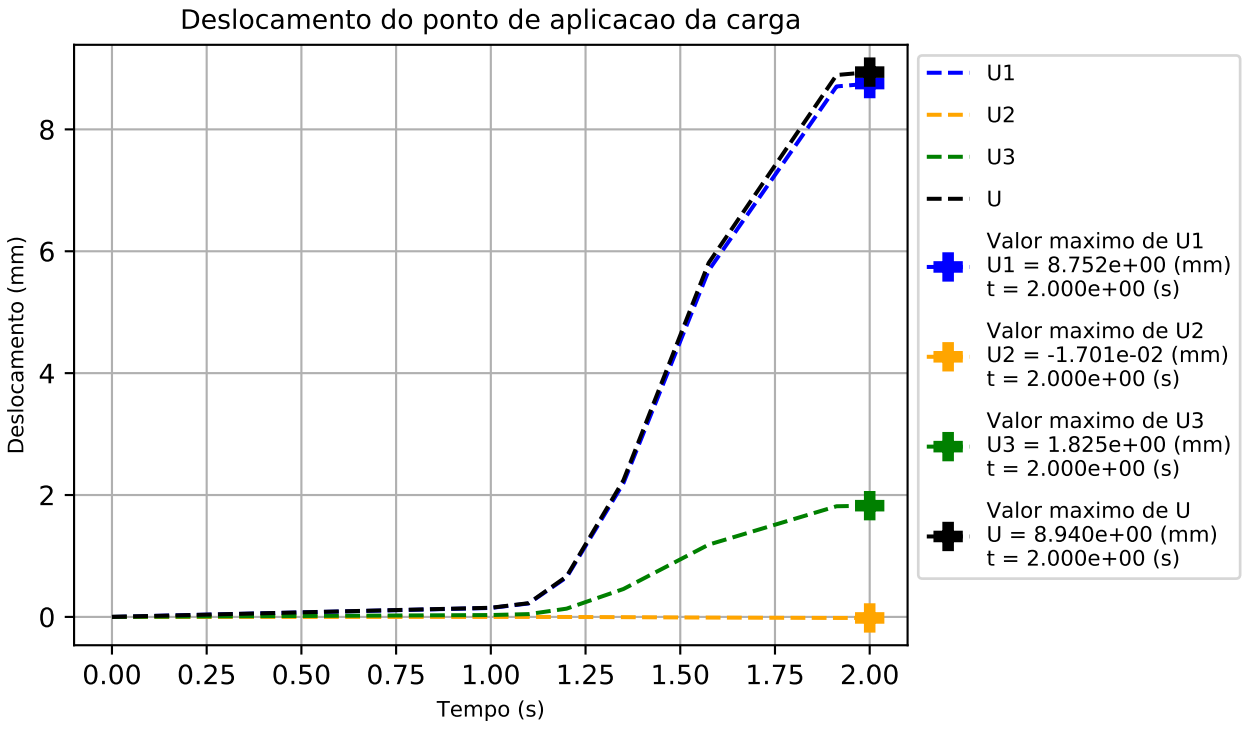

Tensao por elem.
Largura_30.0 Altura_3.0 Espessura_2.5
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

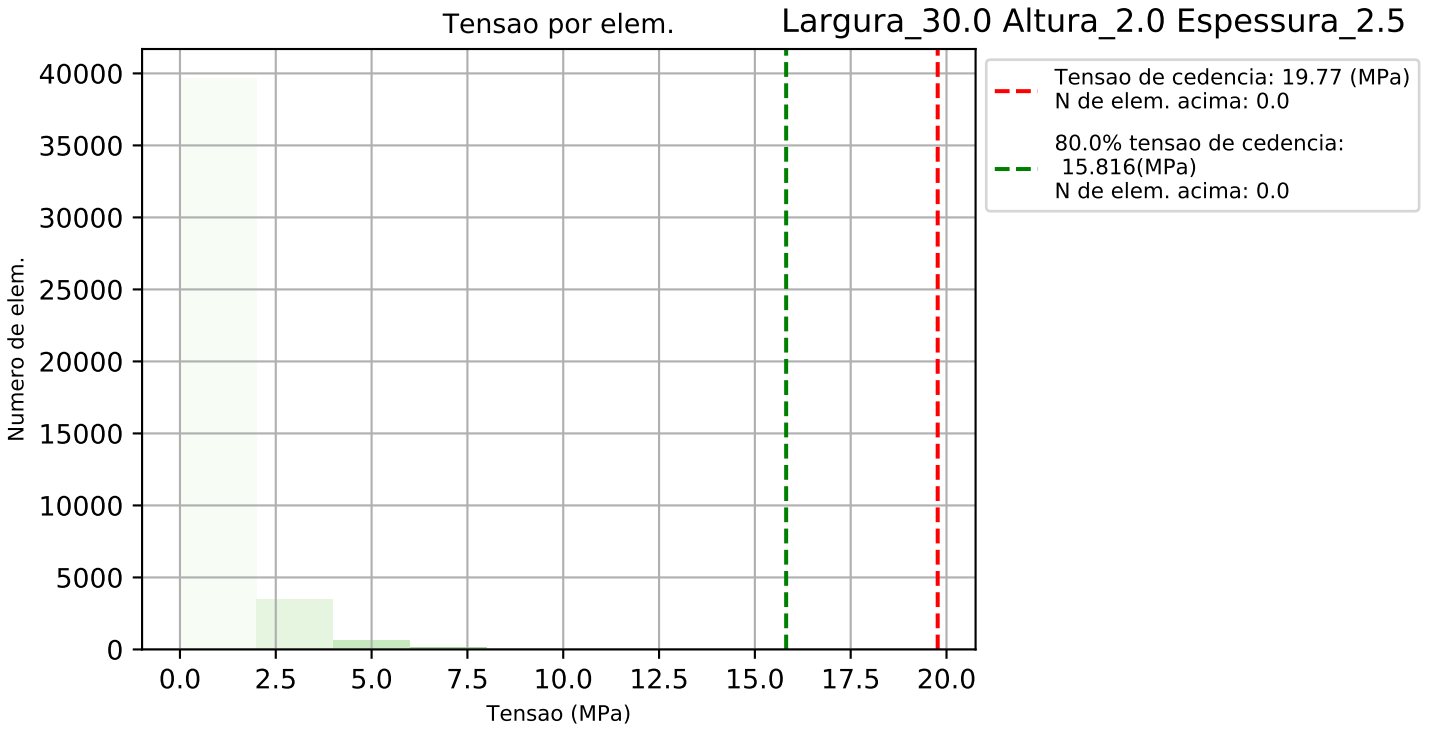

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

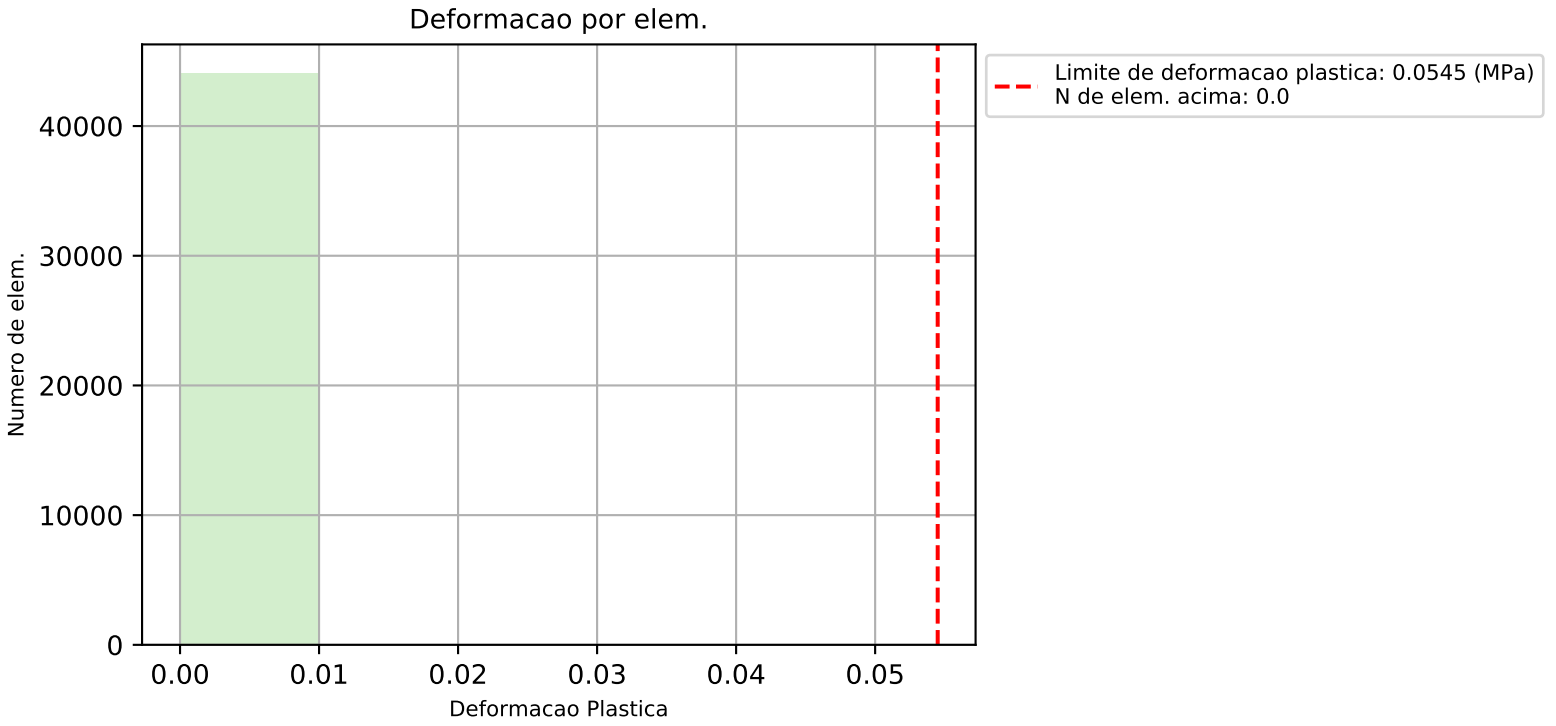

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.698e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.691e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.814e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.885e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

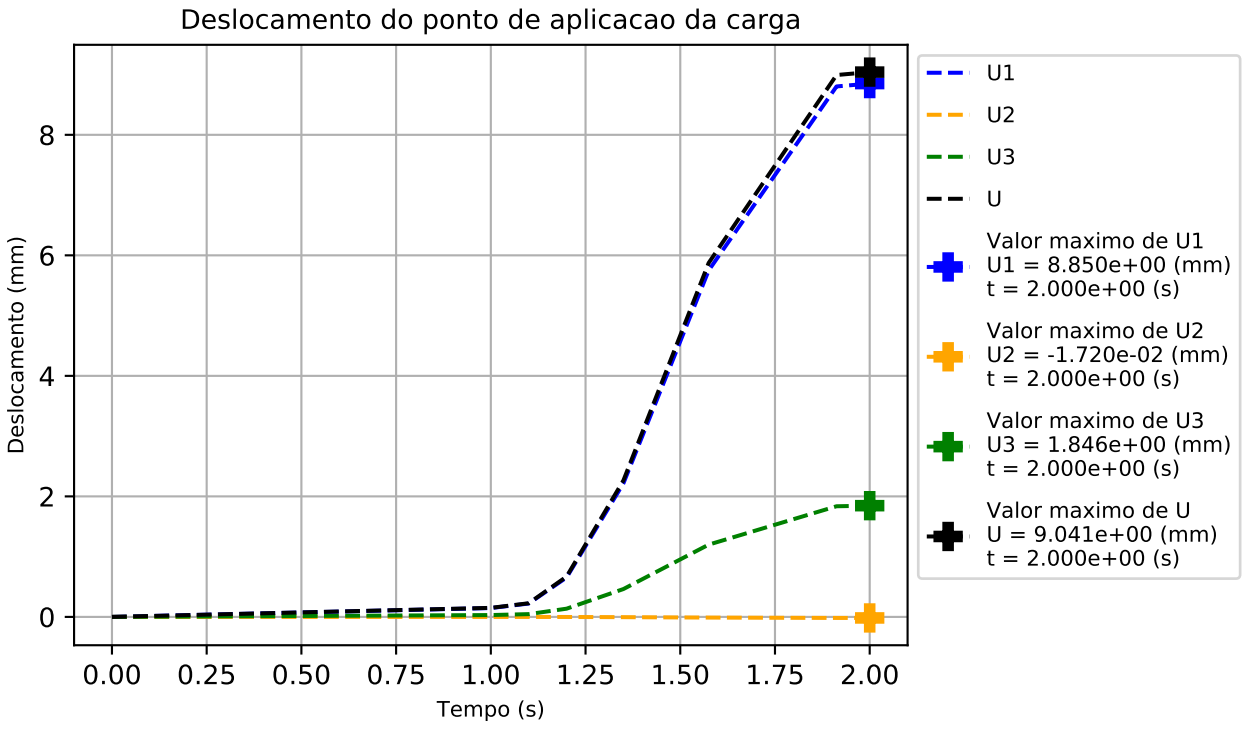

Tensao por elem.
Largura_30.0 Altura_2.0 Espessura_3.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

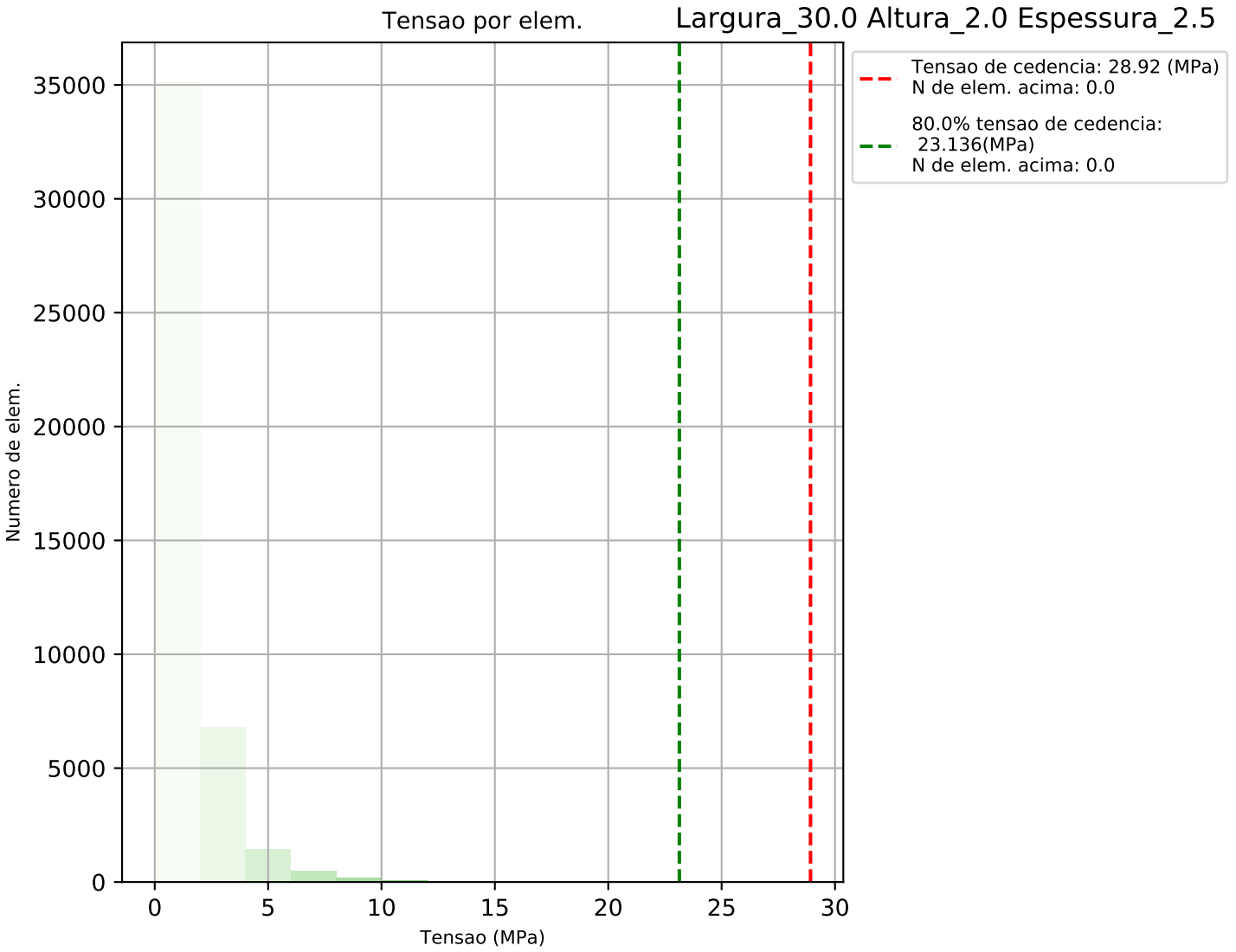

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

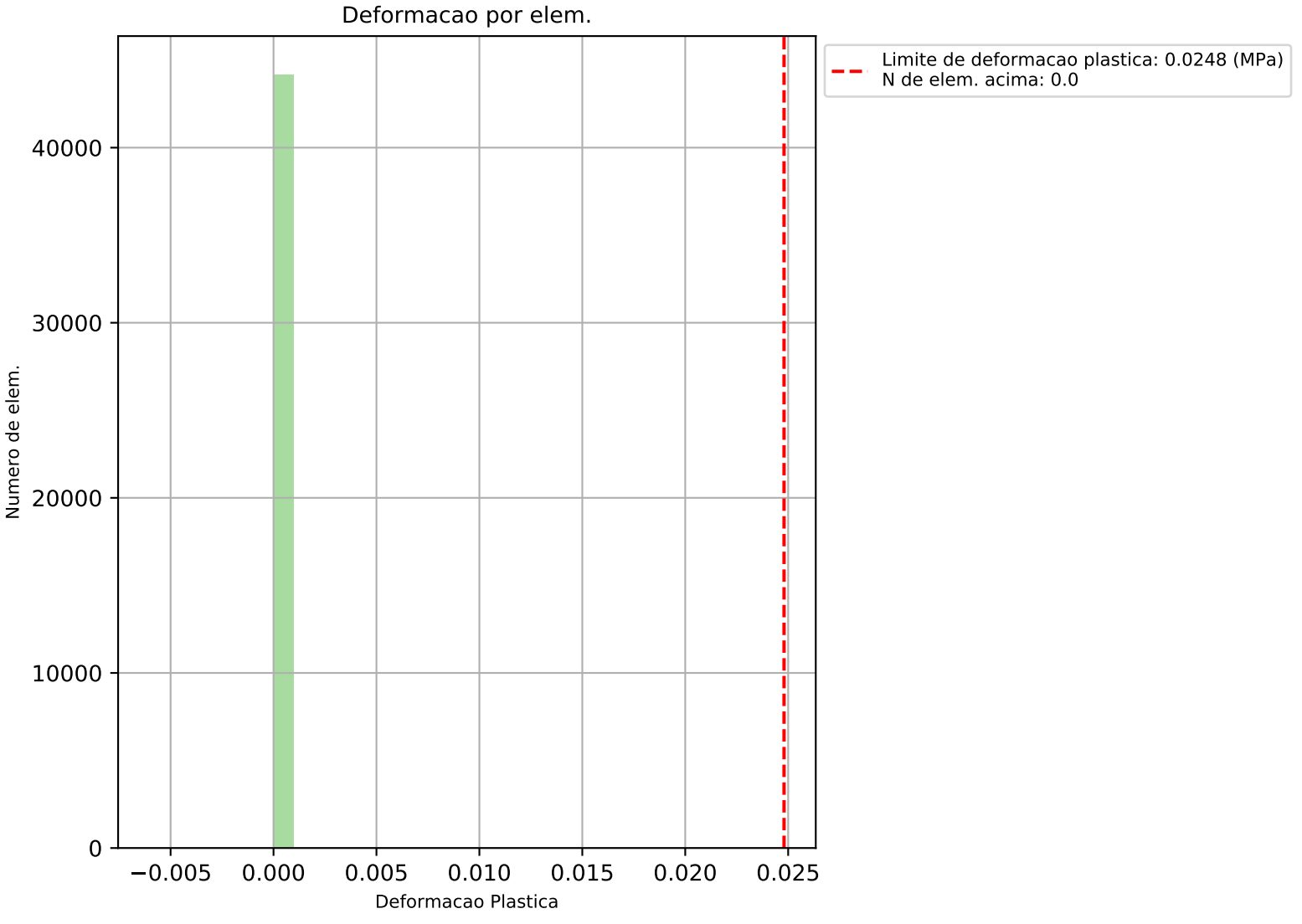

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.752e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.701e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.825e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.940e+00 (mm)
t = 2.000e+00 (s)

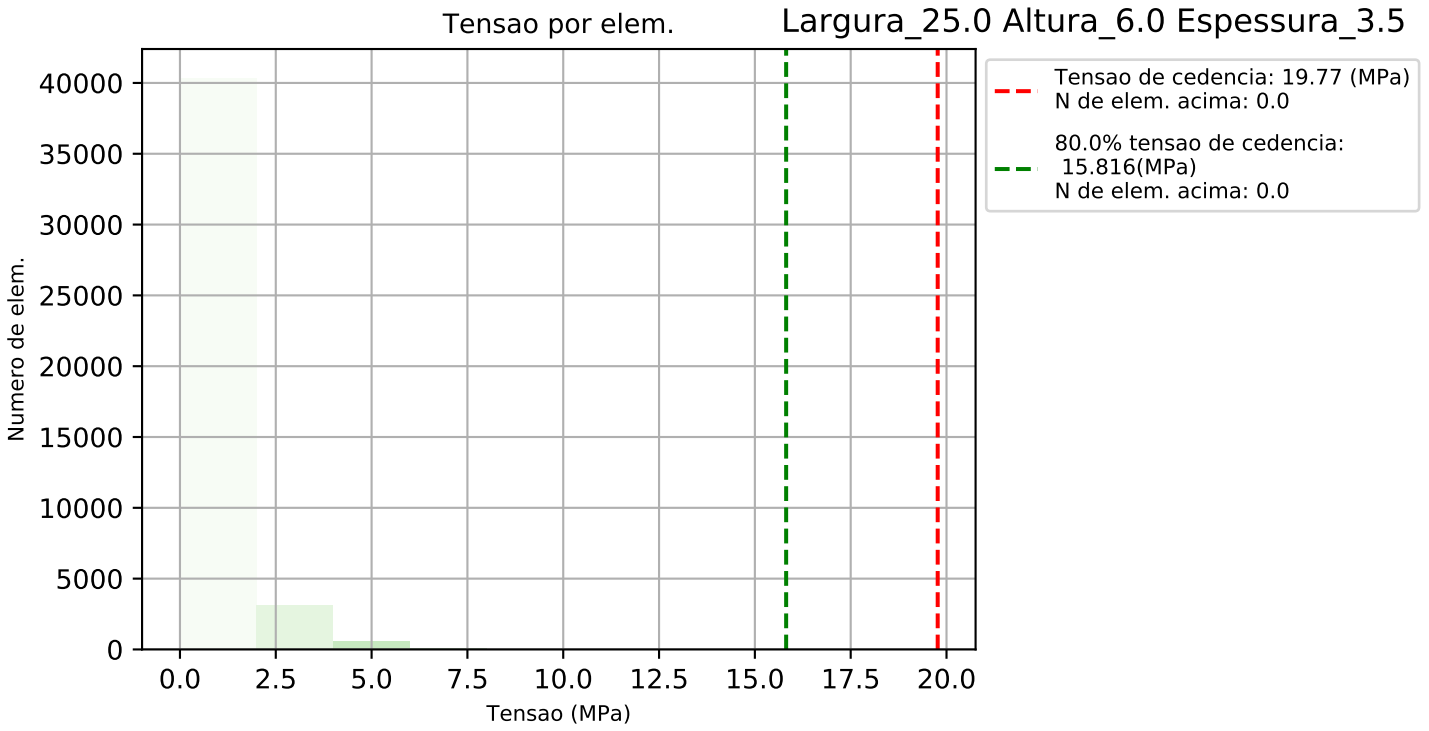

Tensao por elem.
Largura_30.0 Altura_2.0 Espessura_2.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

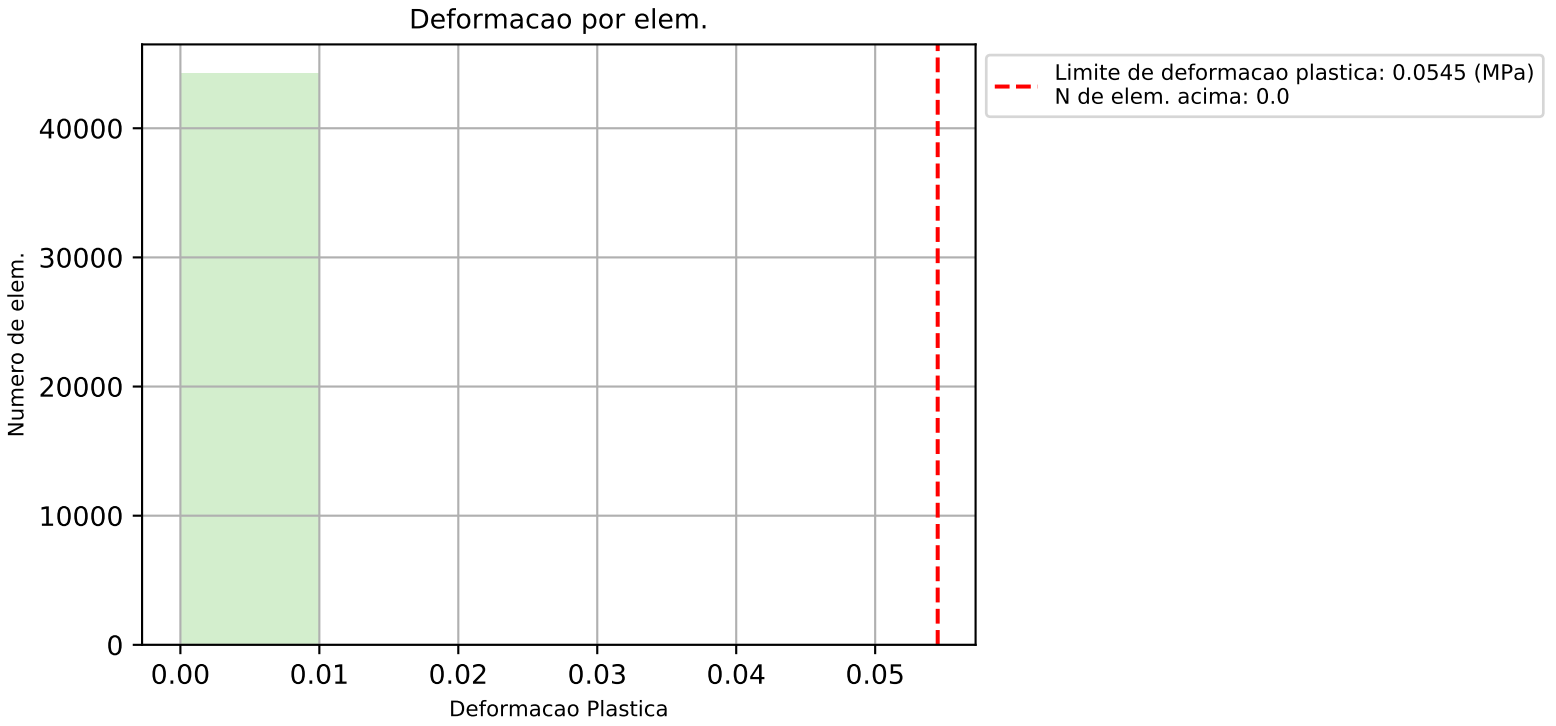

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

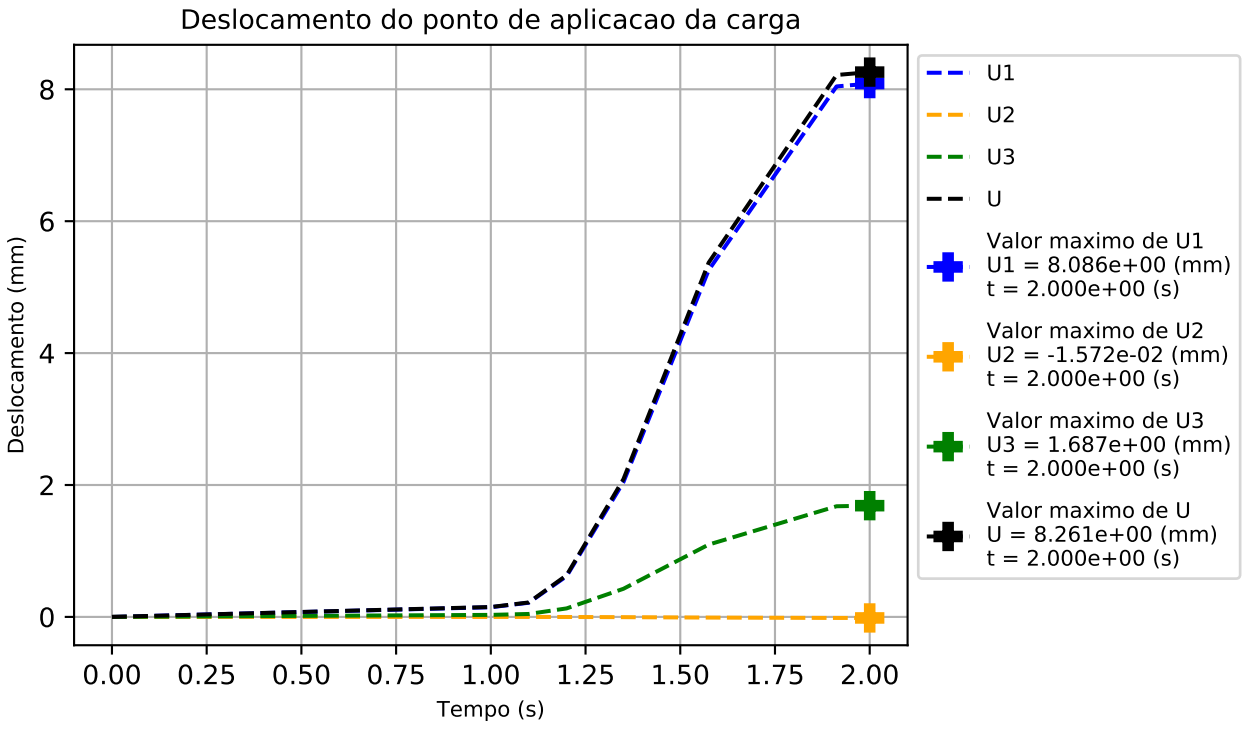

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.850e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.720e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.846e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 9.041e+00 (mm)
t = 2.000e+00 (s)

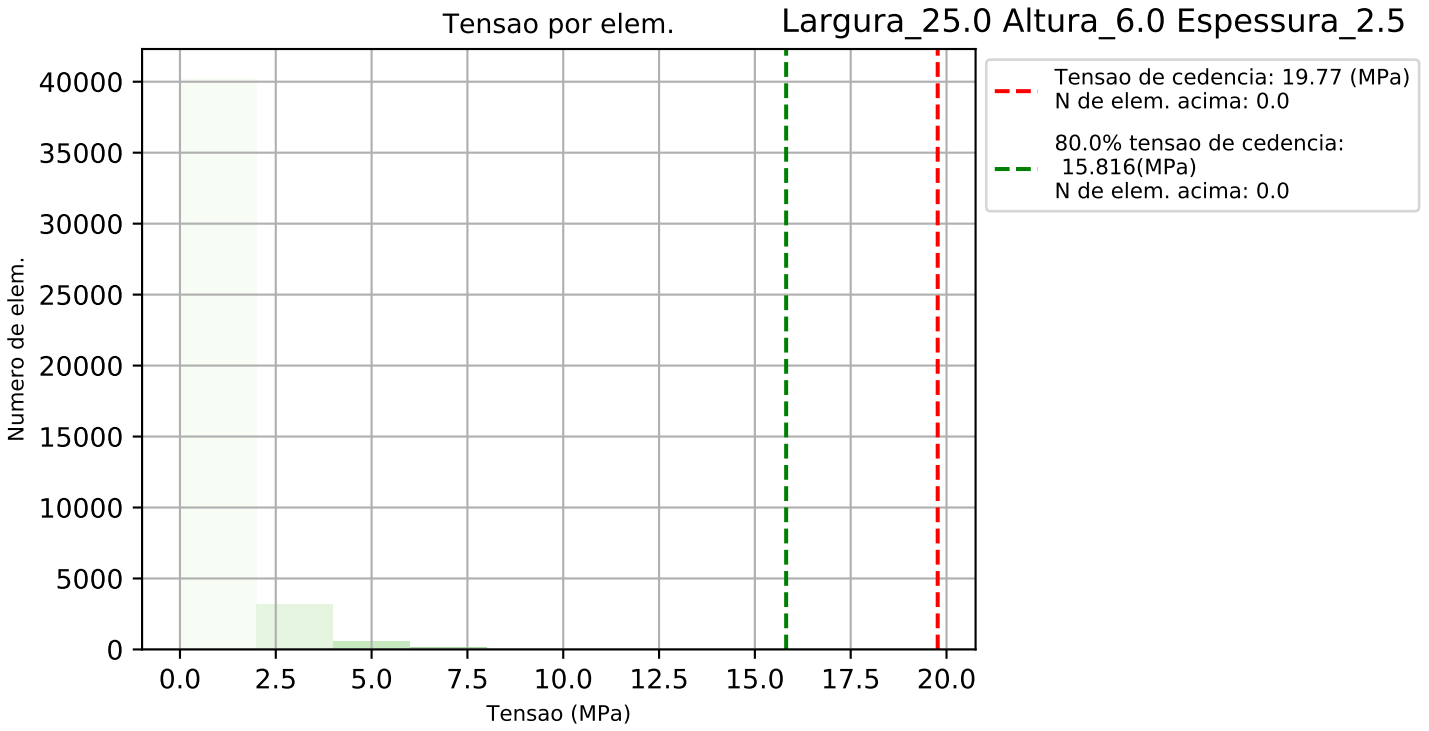

Tensao por elem.
Largura_30.0 Altura_2.0 Espessura_2.5
Tensao de cedencia: 28.92 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
23.136(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

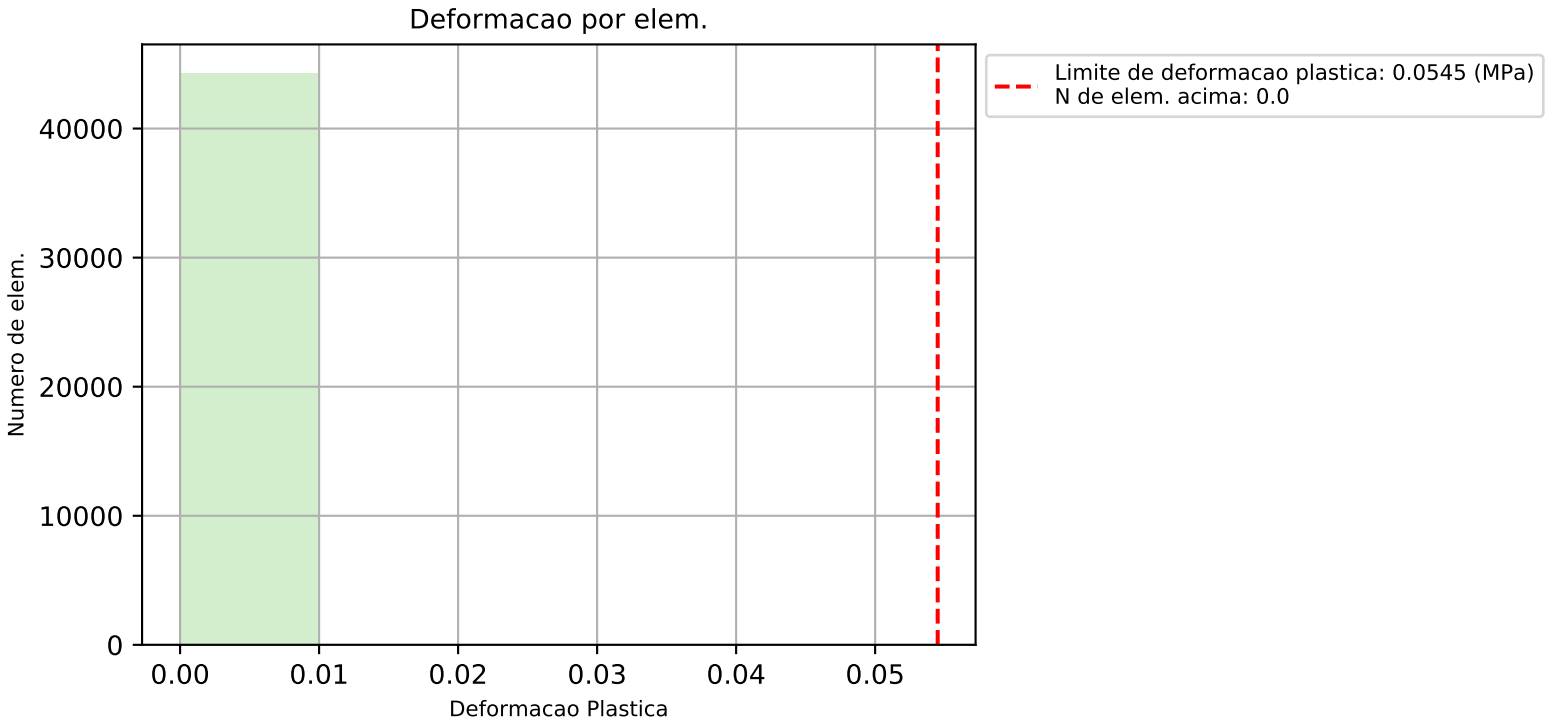

Deformacao por elem.
Limite de deformacao plastica: 0.0248 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

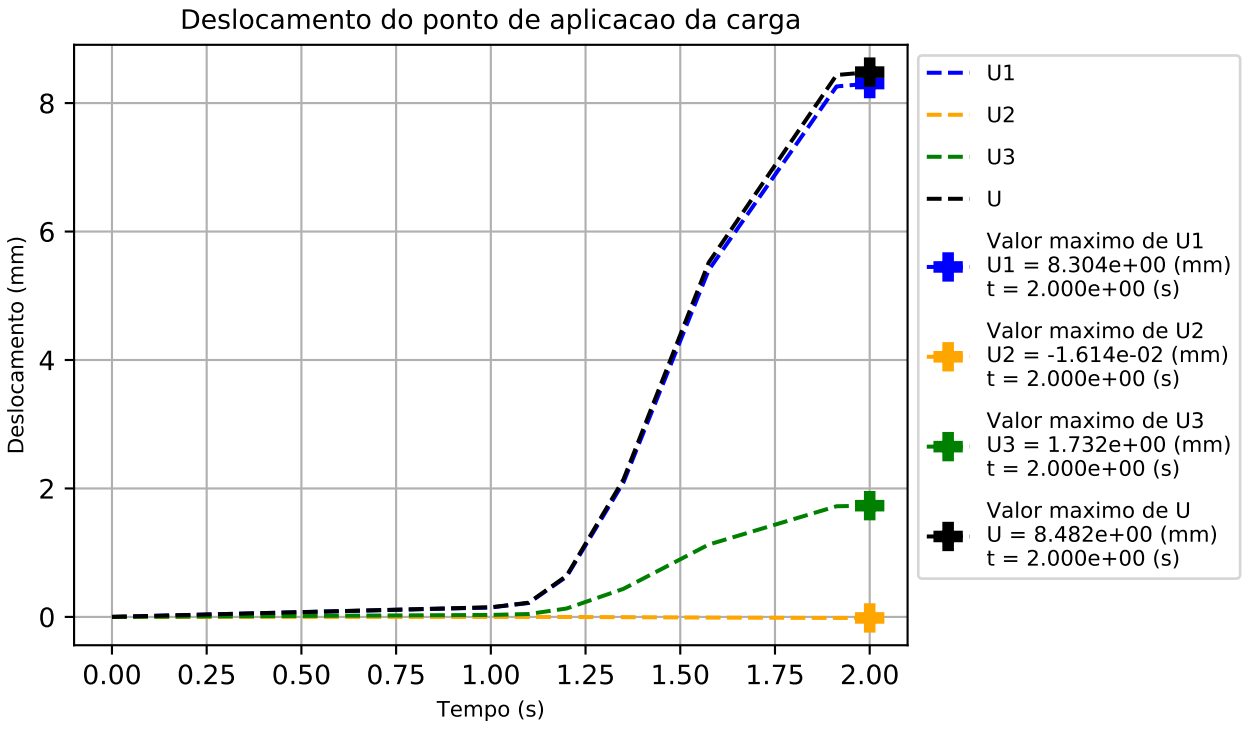

Tensao por elem.
Largura_25.0 Altura_6.0 Espessura_3.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

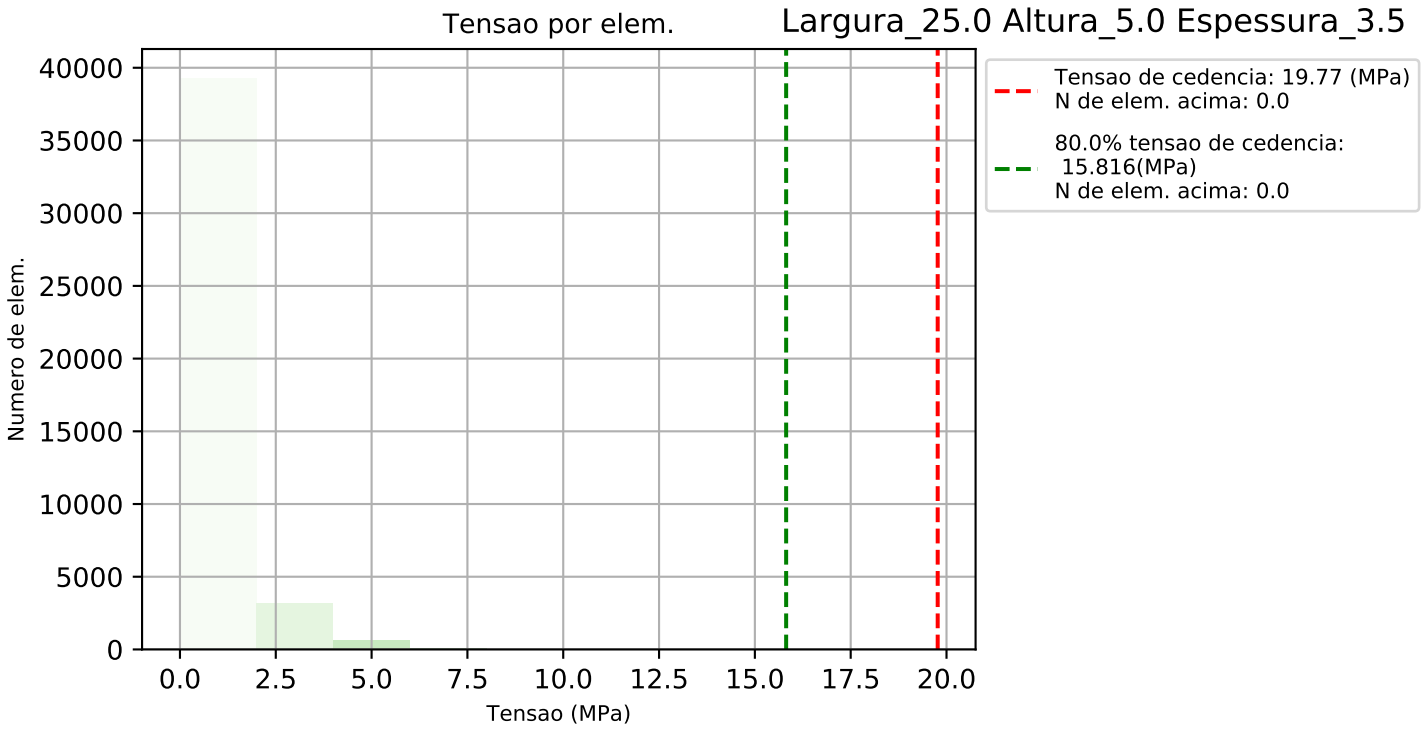

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

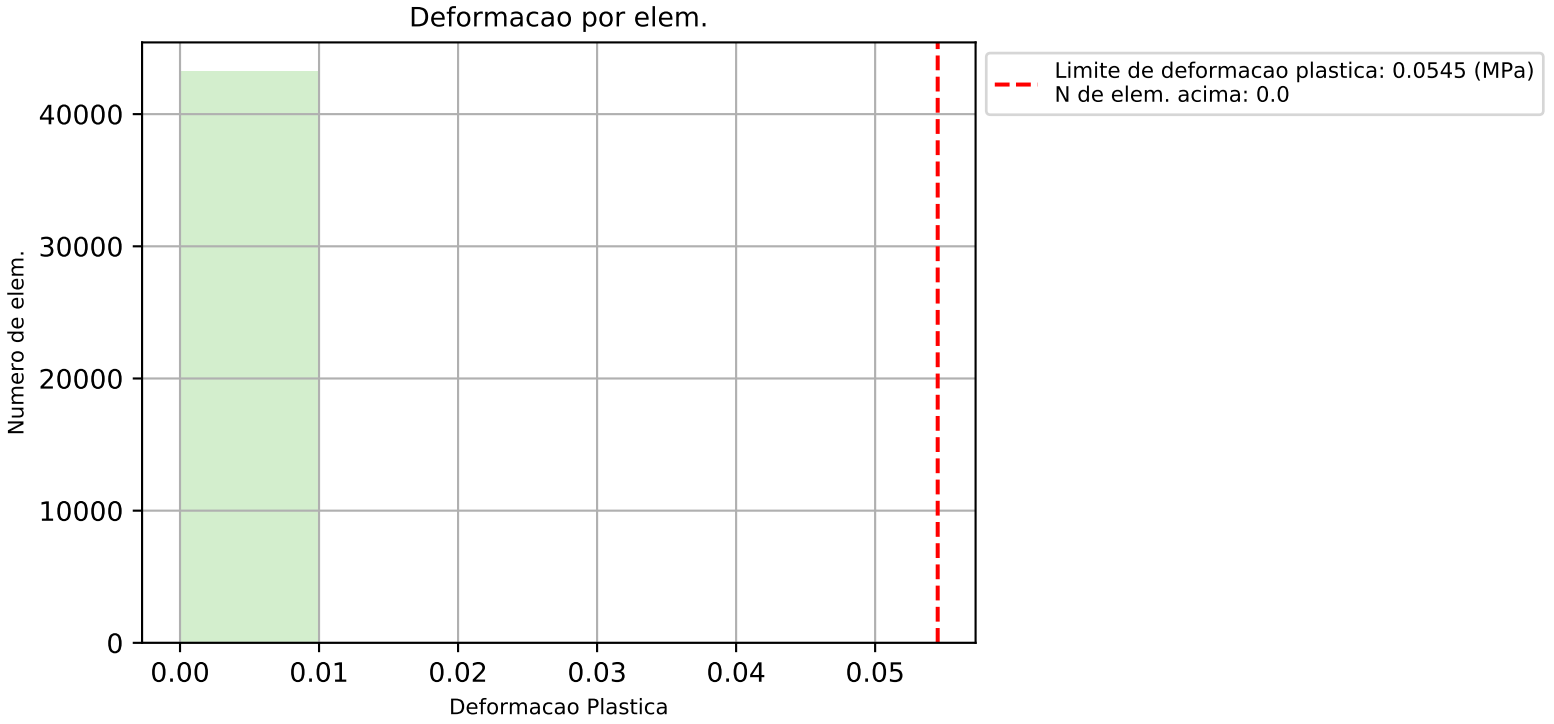

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.086e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.572e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.687e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.261e+00 (mm)
t = 2.000e+00 (s)

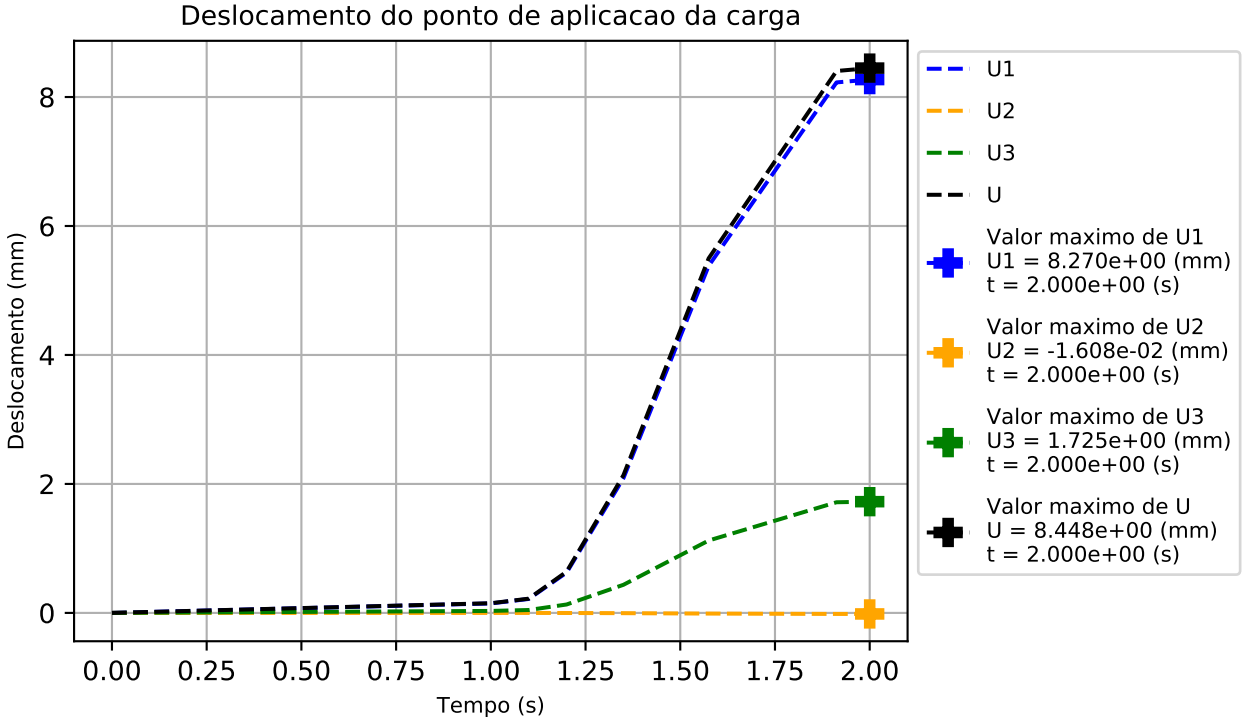

Tensao por elem.
Largura_25.0 Altura_6.0 Espessura_2.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

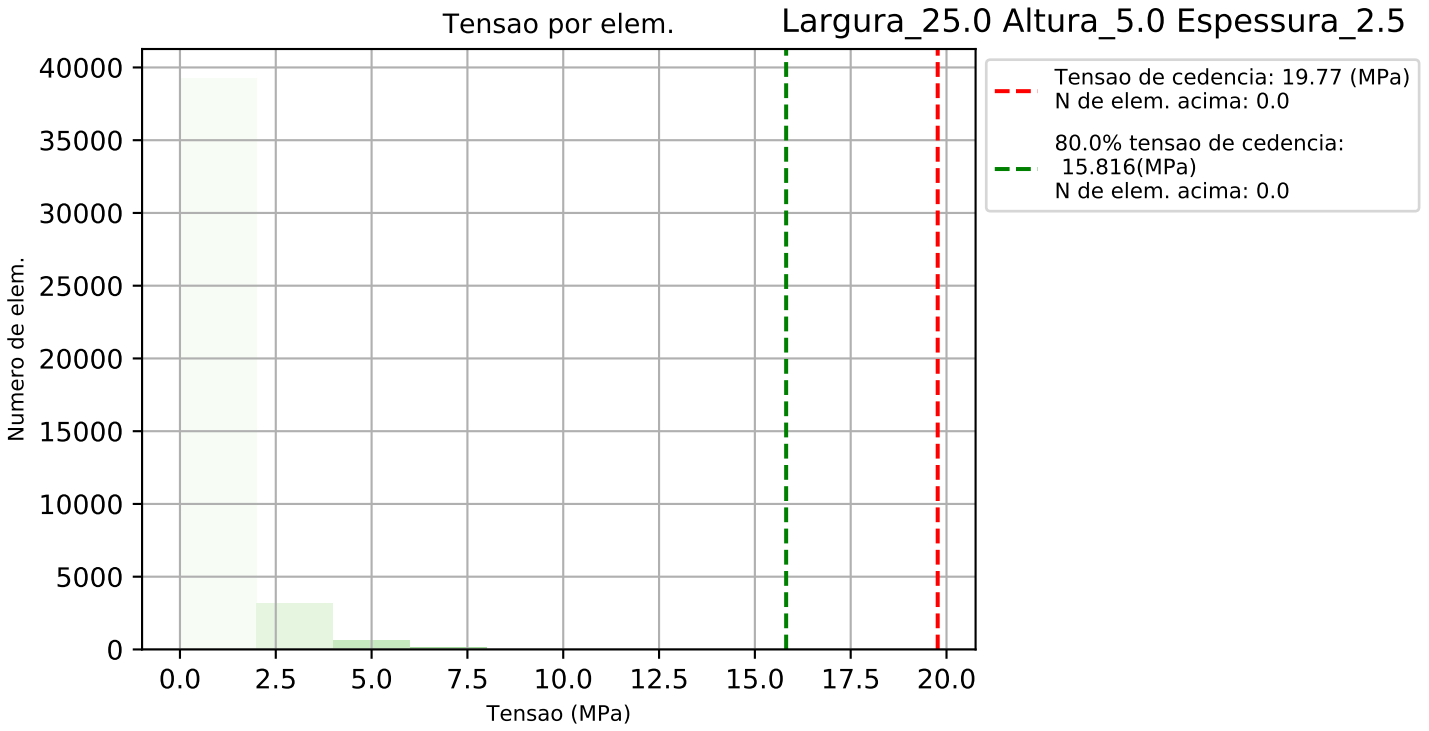

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

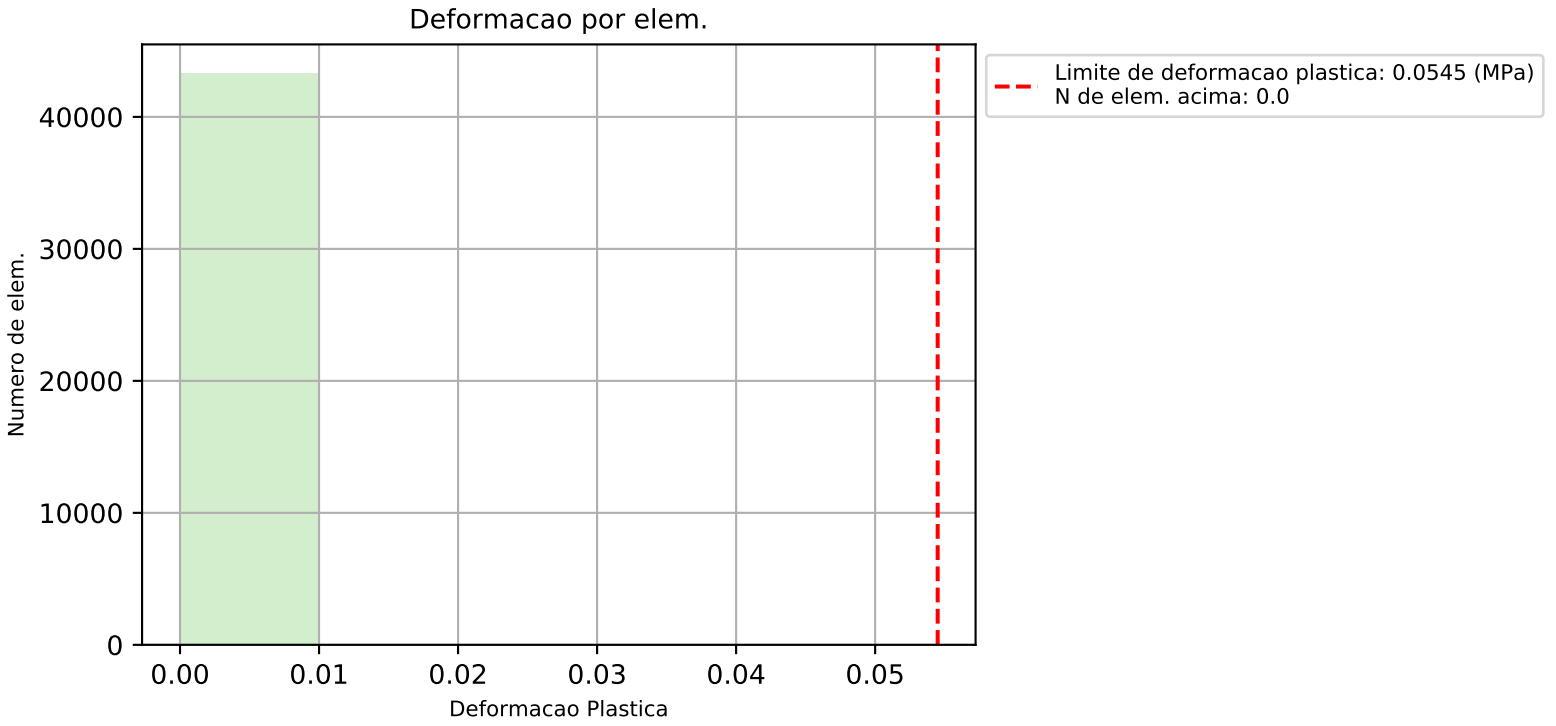

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.304e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.614e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.732e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.482e+00 (mm)
t = 2.000e+00 (s)

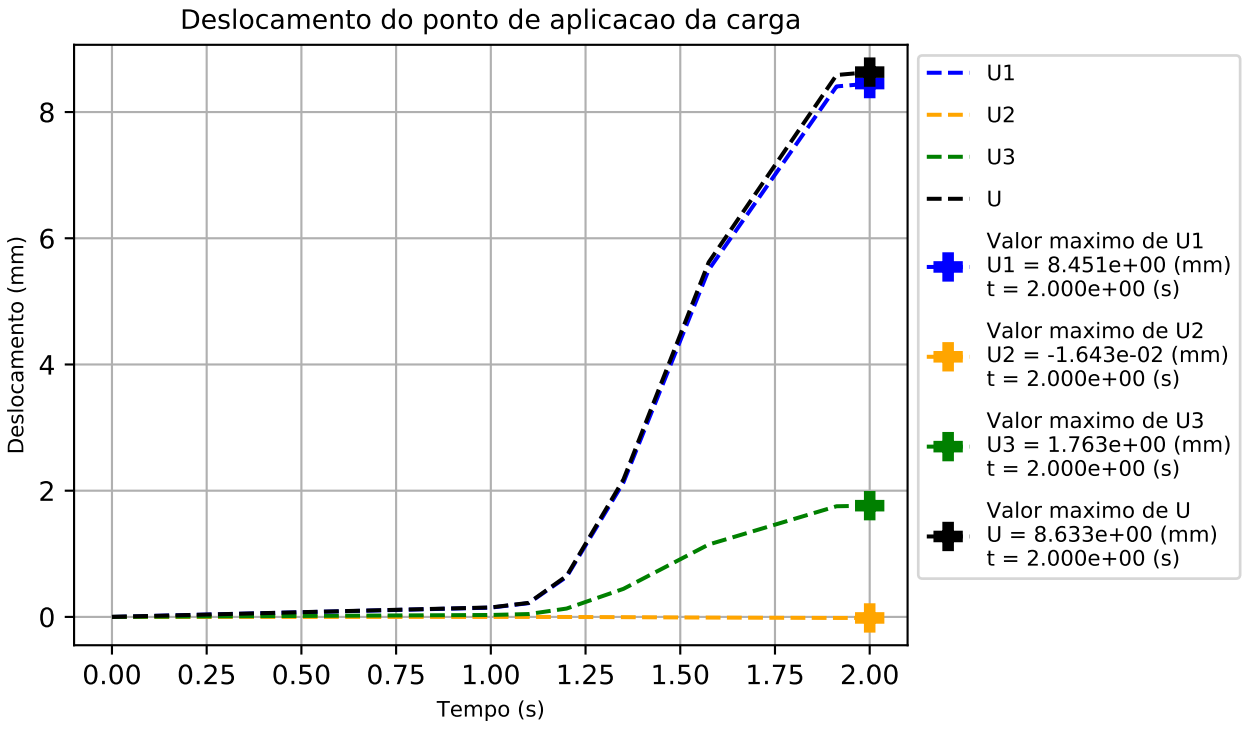

Tensao por elem.
Largura_25.0 Altura_5.0 Espessura_3.5
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

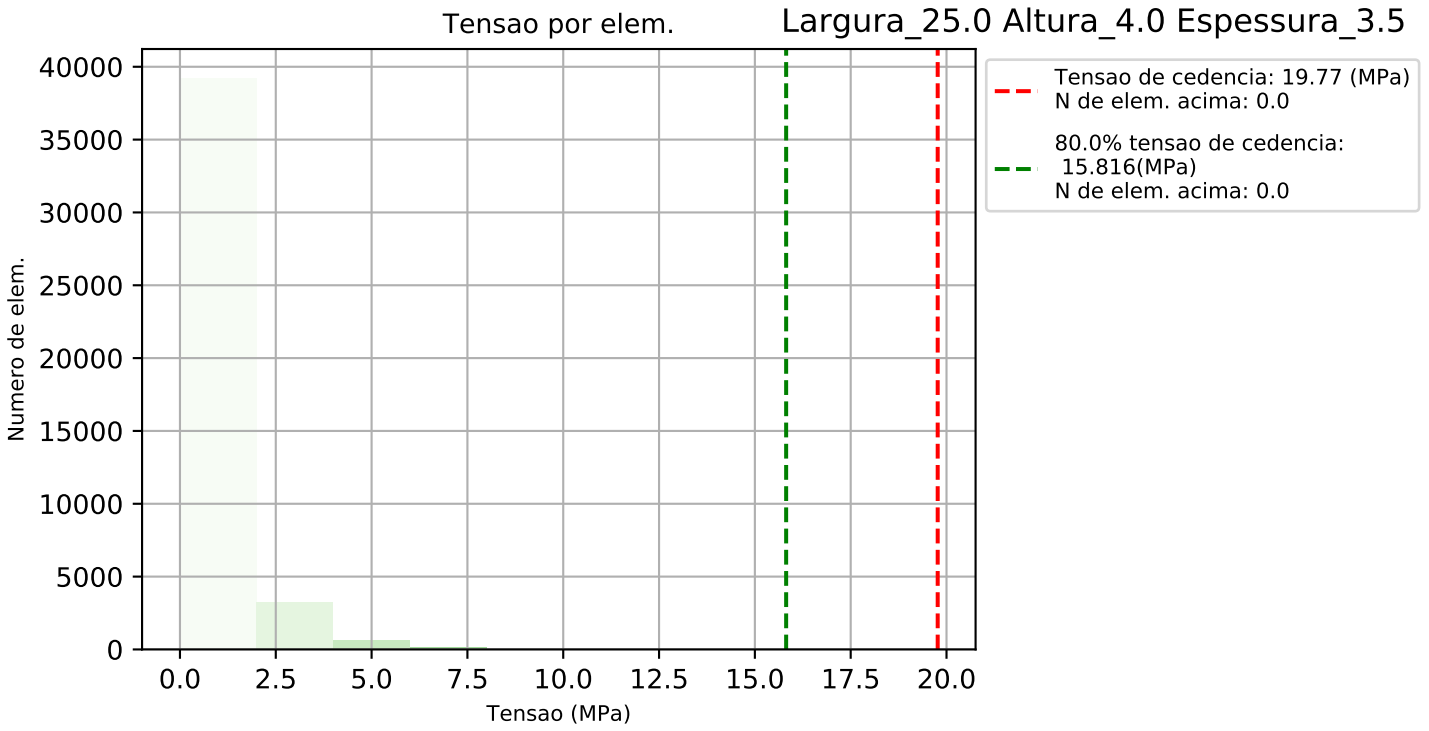

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

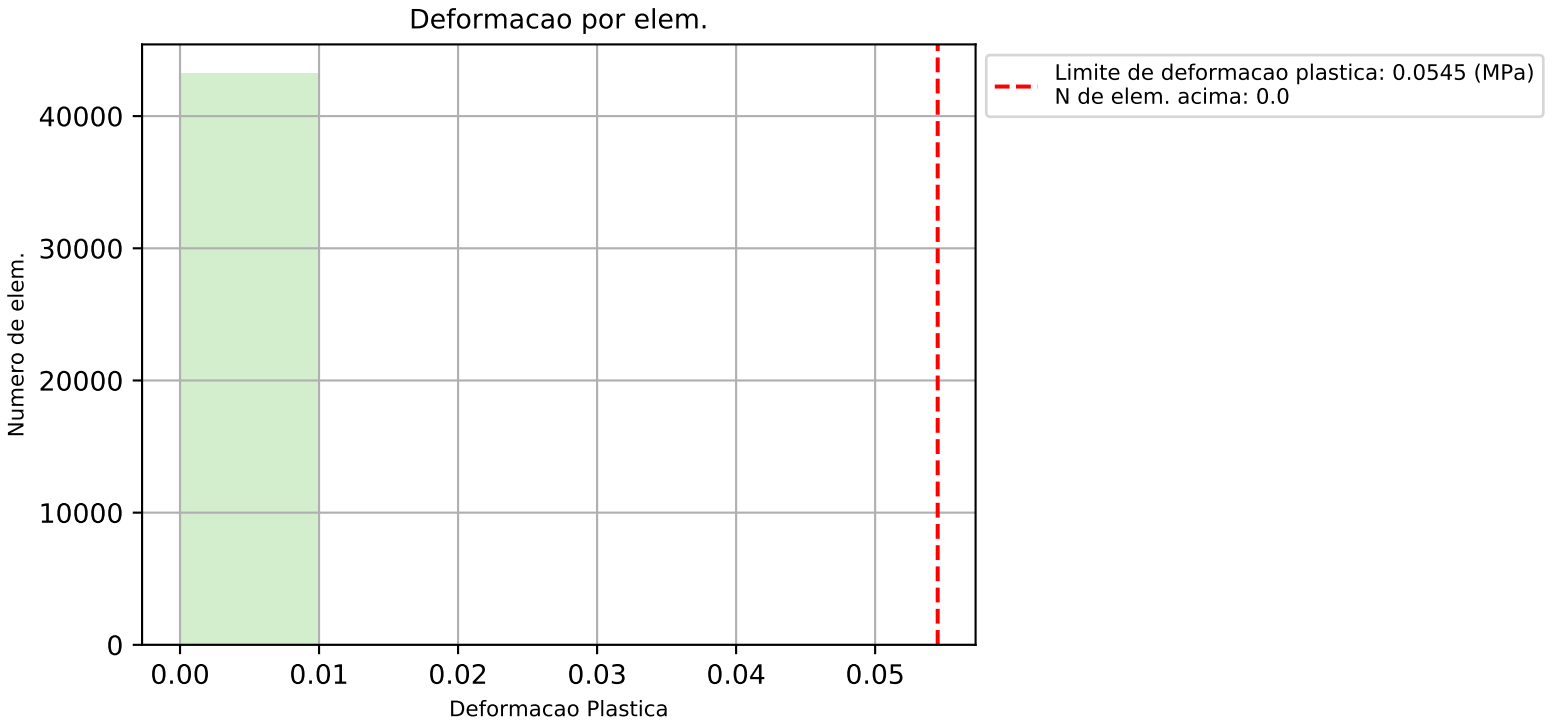

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.270e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.608e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.725e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.448e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

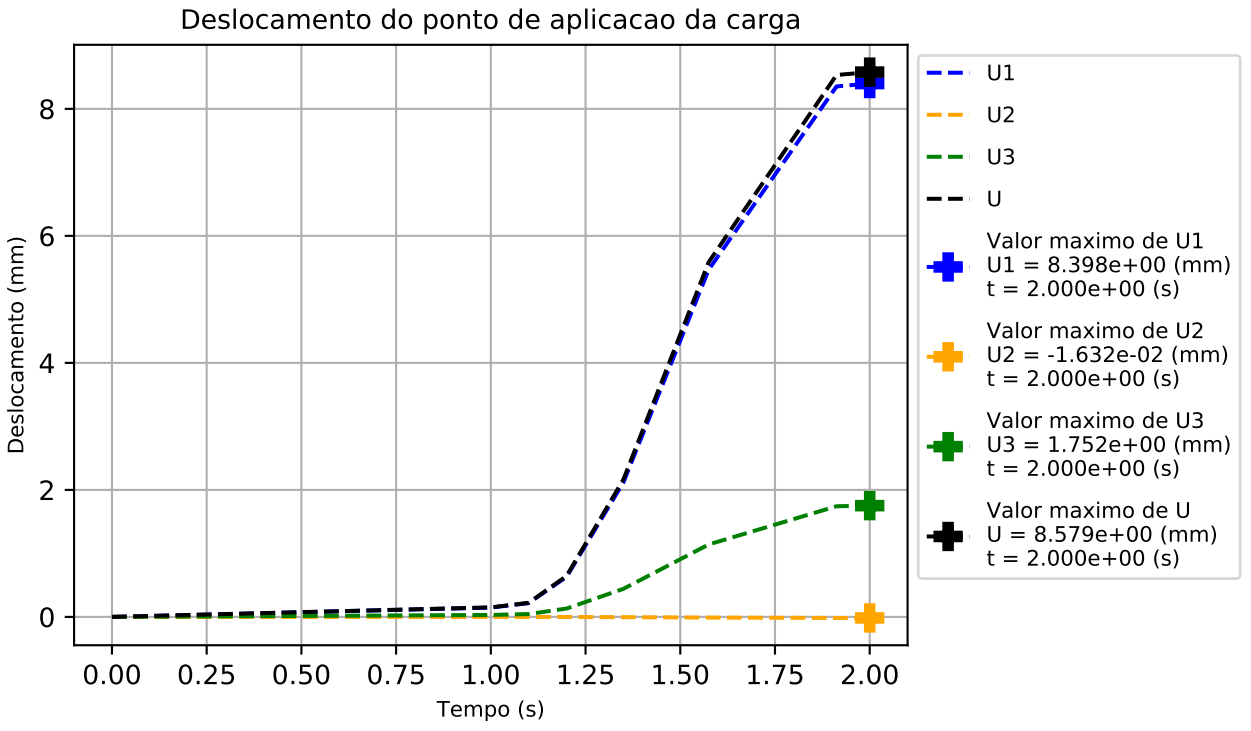

Tensao por elem.
Largura_25.0 Altura_5.0 Espessura_2.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

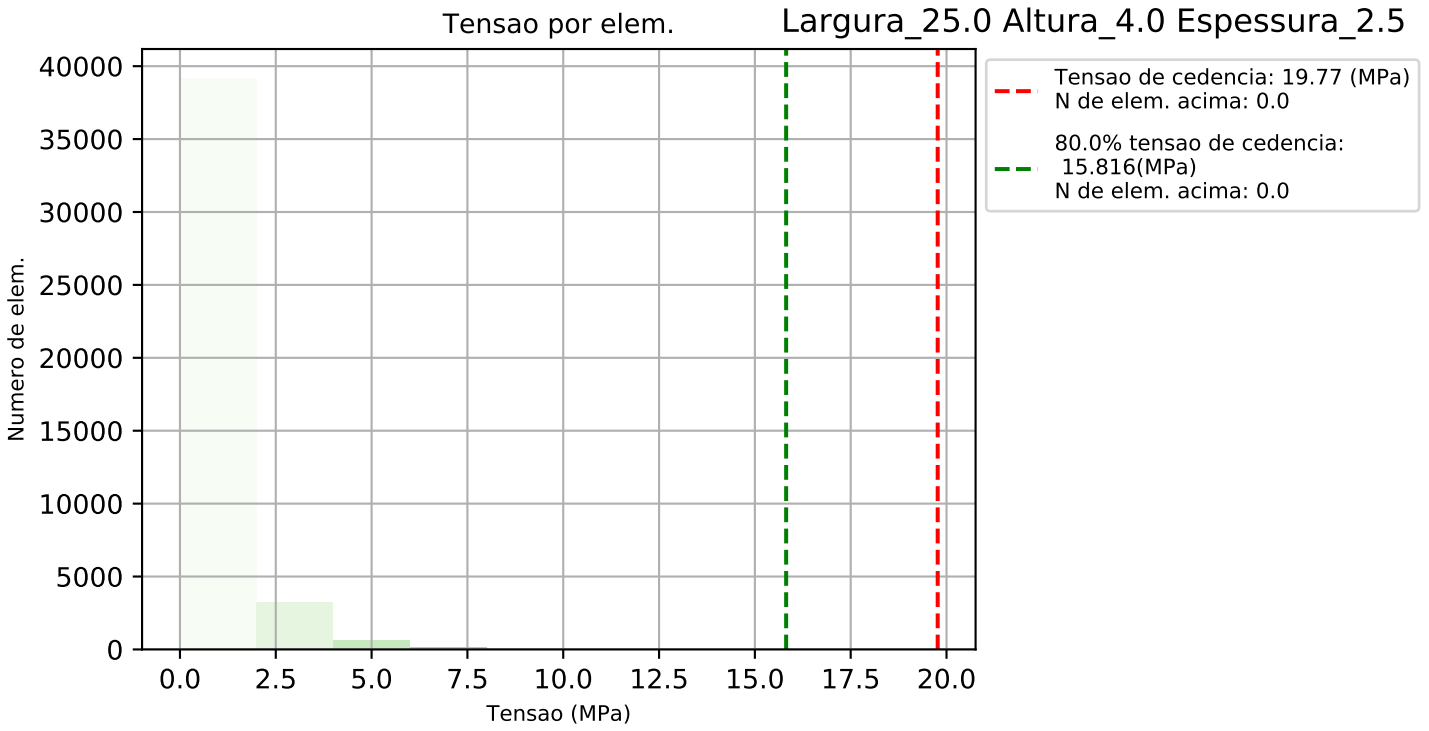

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

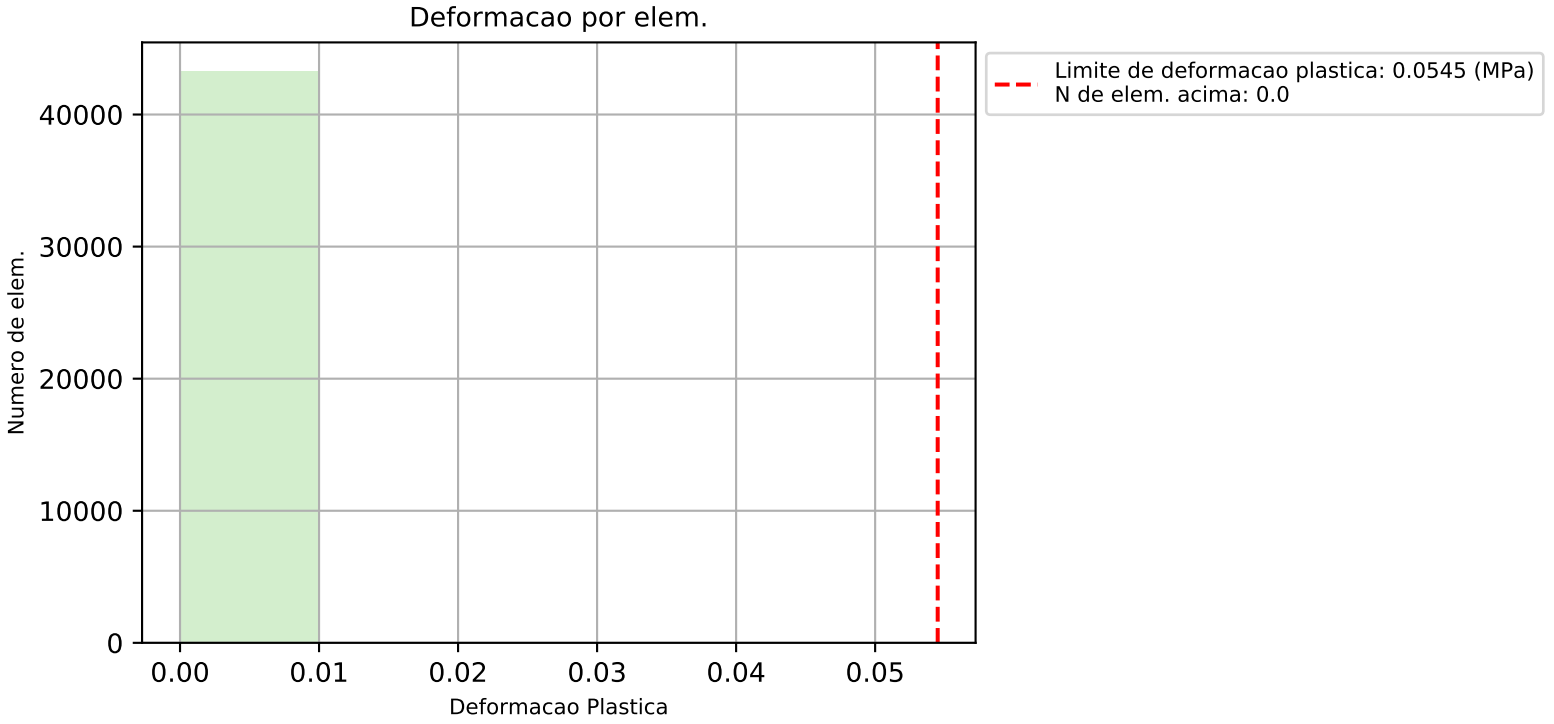

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.451e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.643e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.763e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.633e+00 (mm)
t = 2.000e+00 (s)

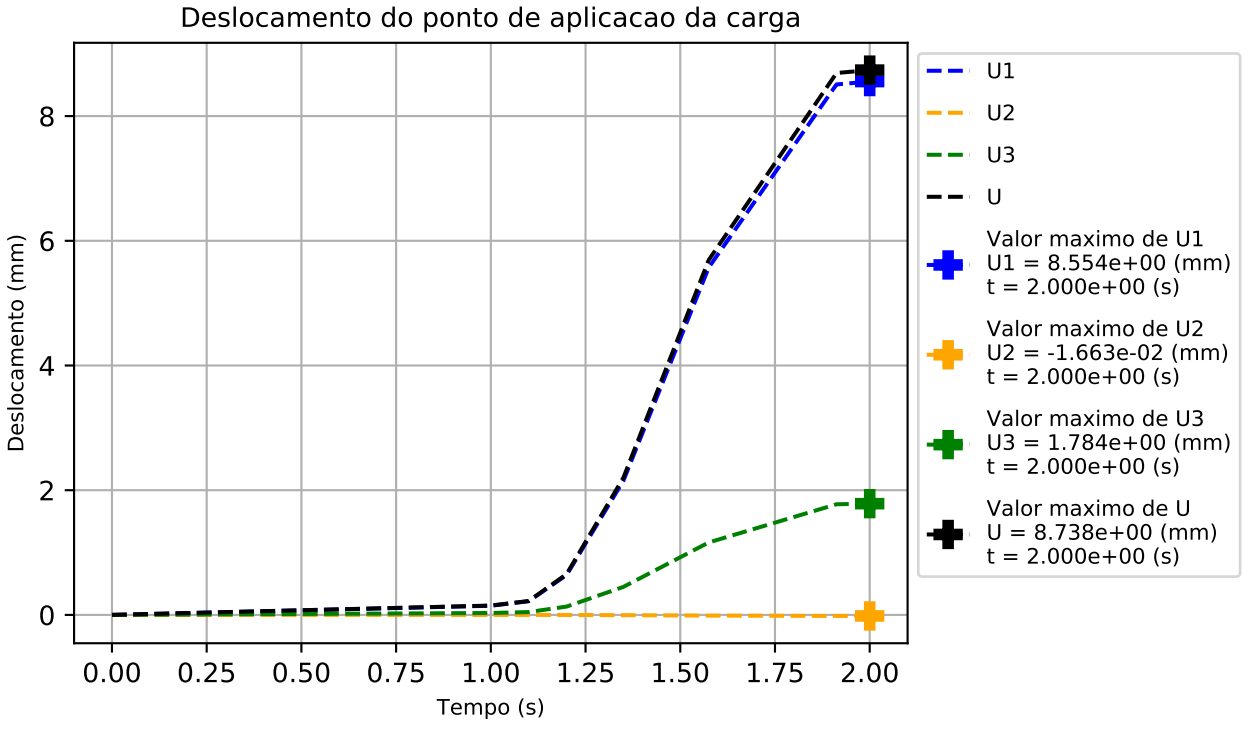

Tensao por elem.
Largura_25.0 Altura_4.0 Espessura_3.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

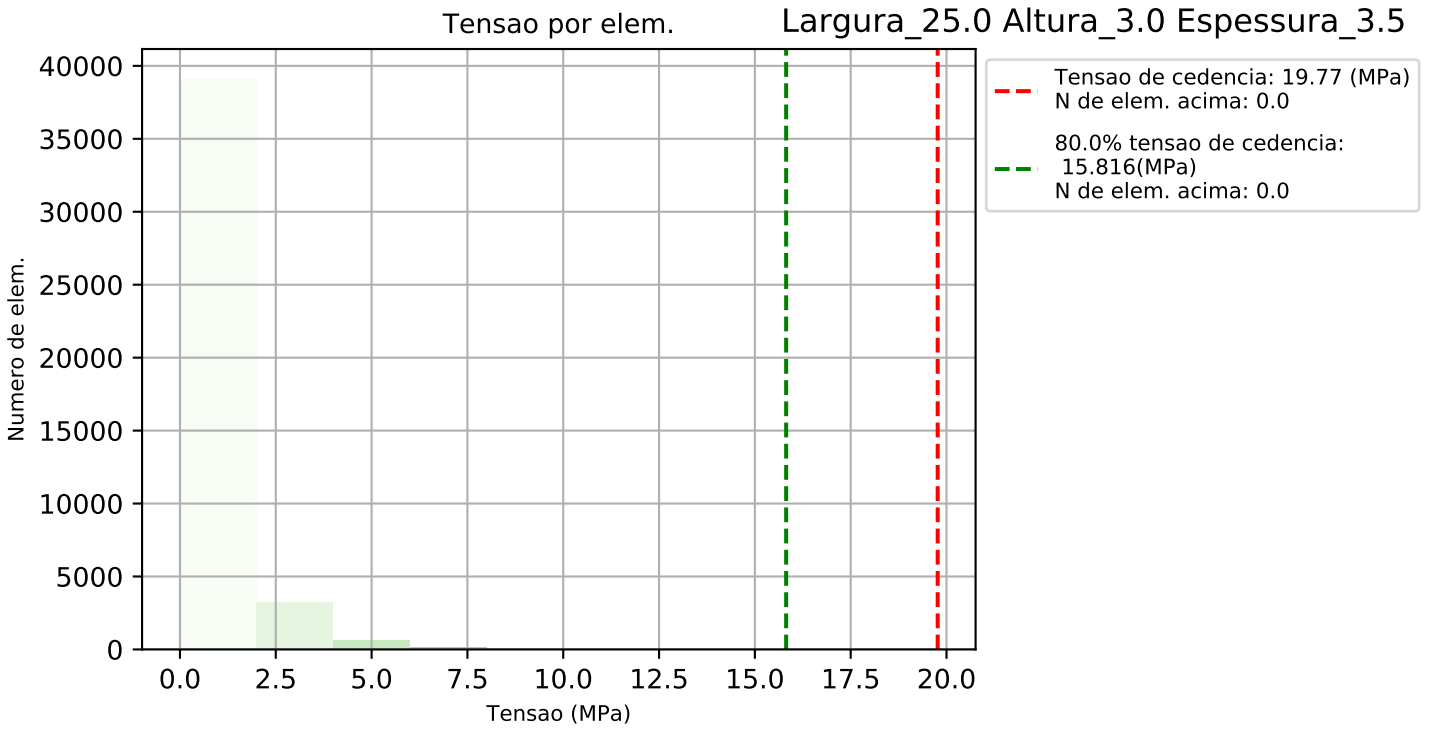

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

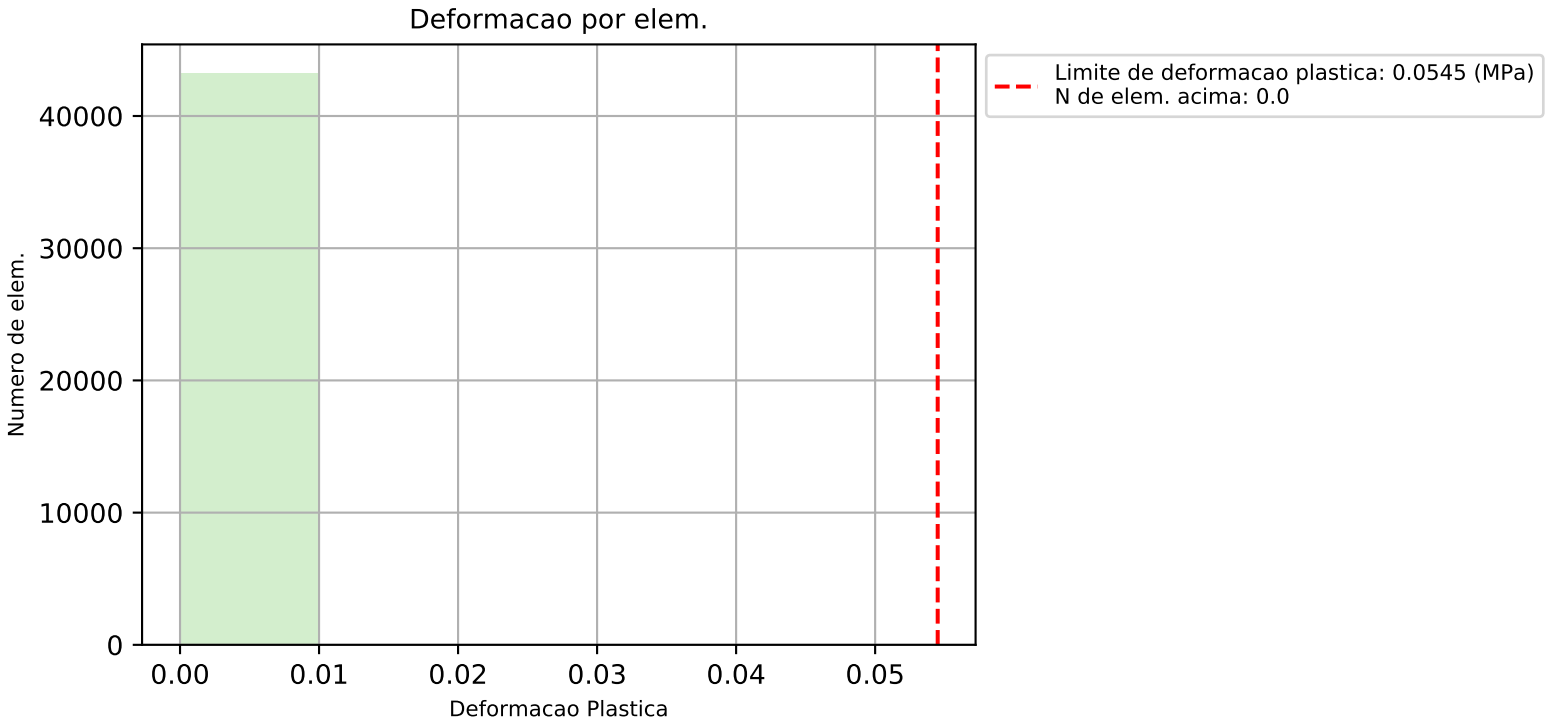

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.398e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.632e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.752e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.579e+00 (mm)
t = 2.000e+00 (s)

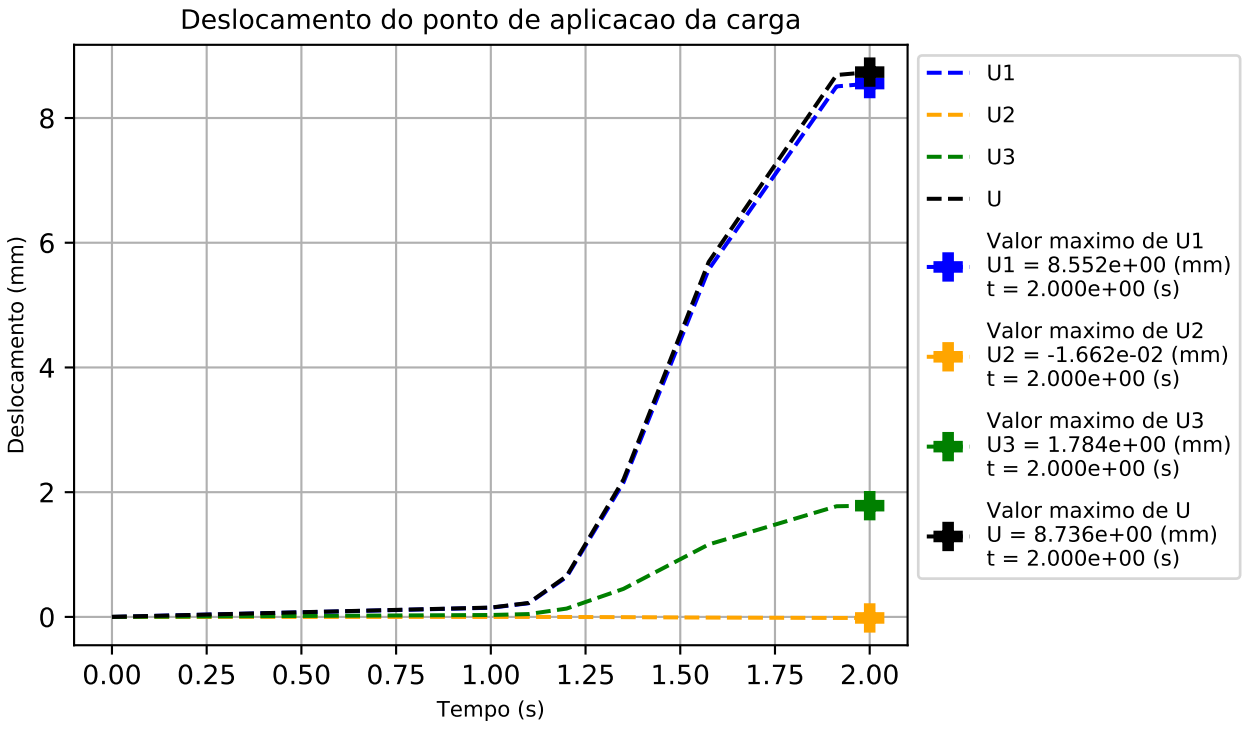

Tensao por elem.
Largura_25.0 Altura_4.0 Espessura_2.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

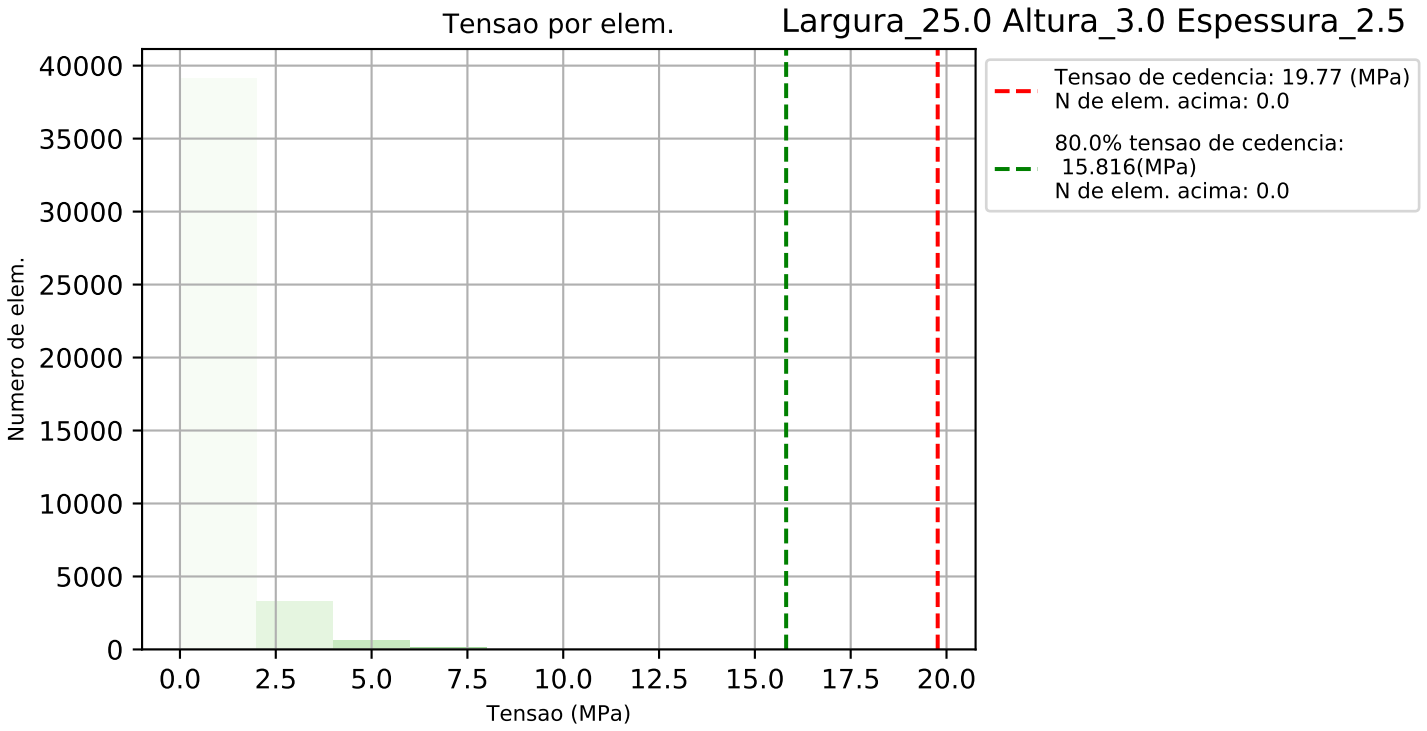

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

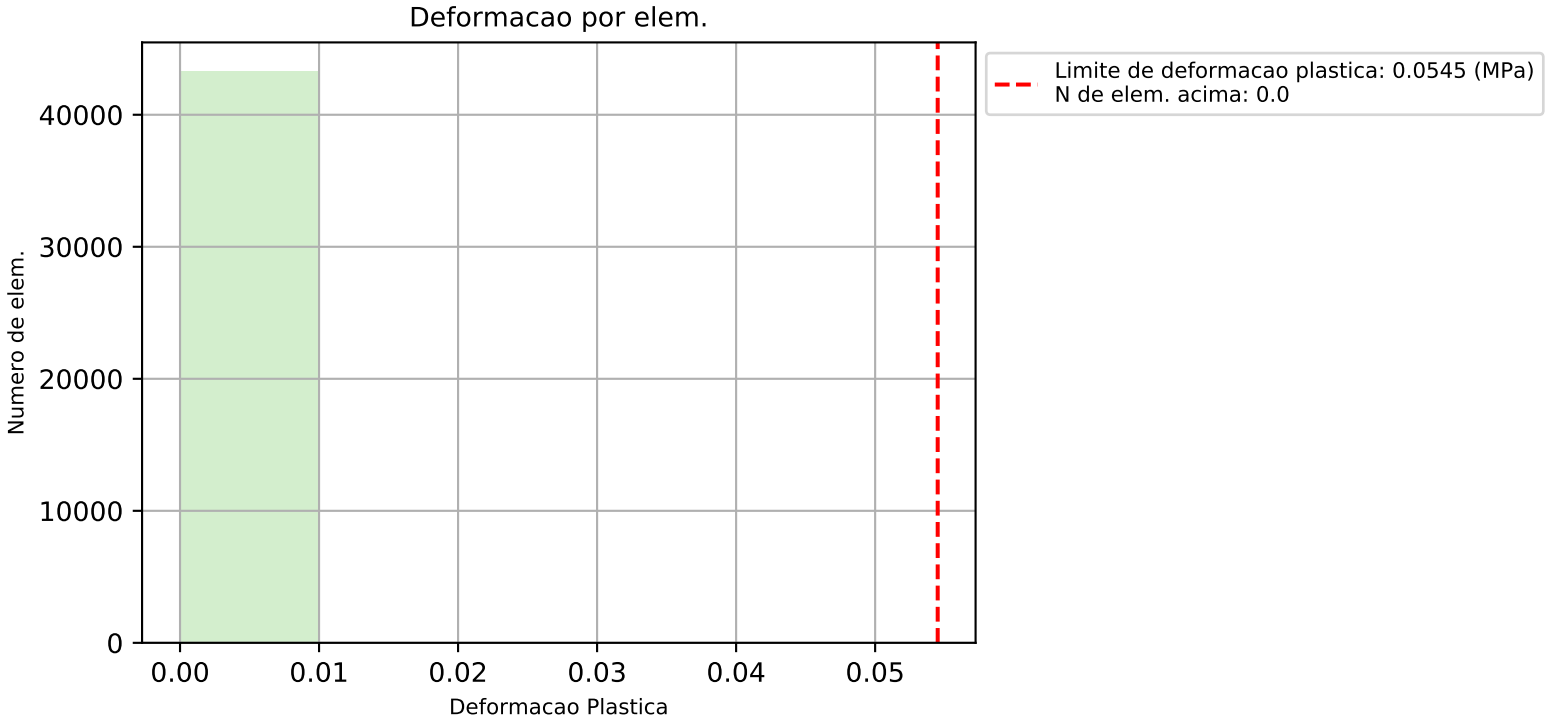

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.554e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.663e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.784e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.738e+00 (mm)
t = 2.000e+00 (s)

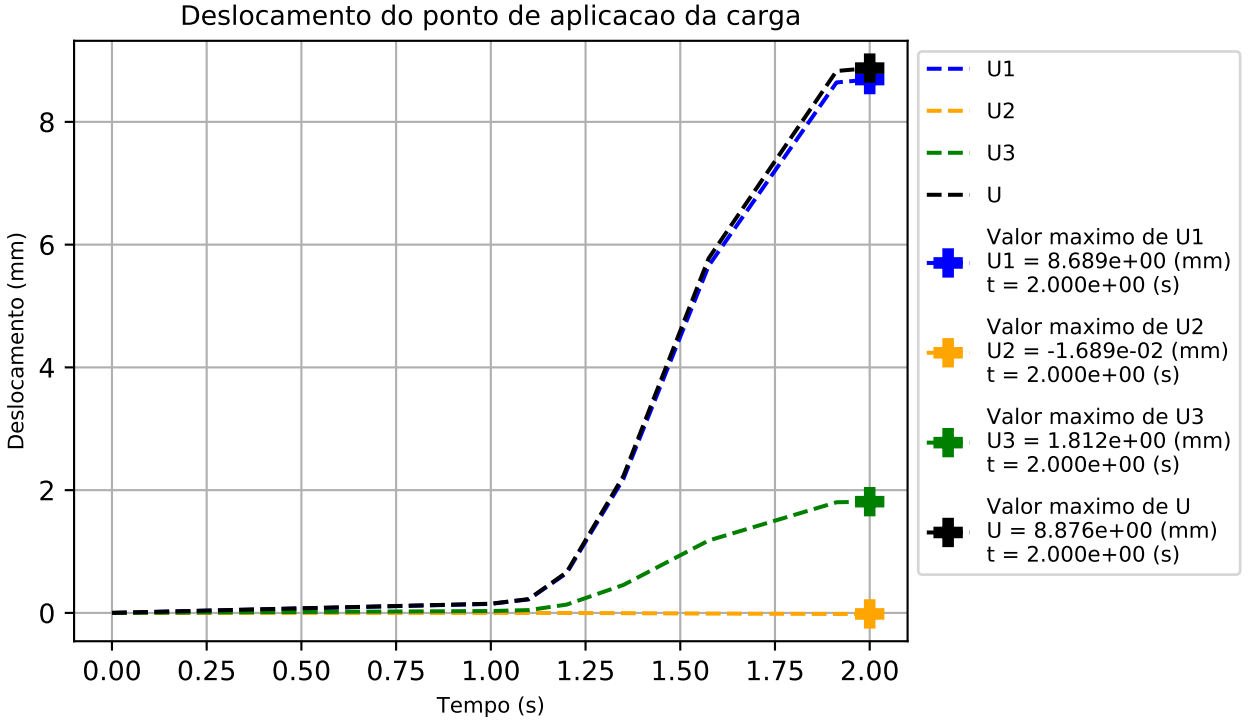

Tensao por elem.
Largura_25.0 Altura_3.0 Espessura_3.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

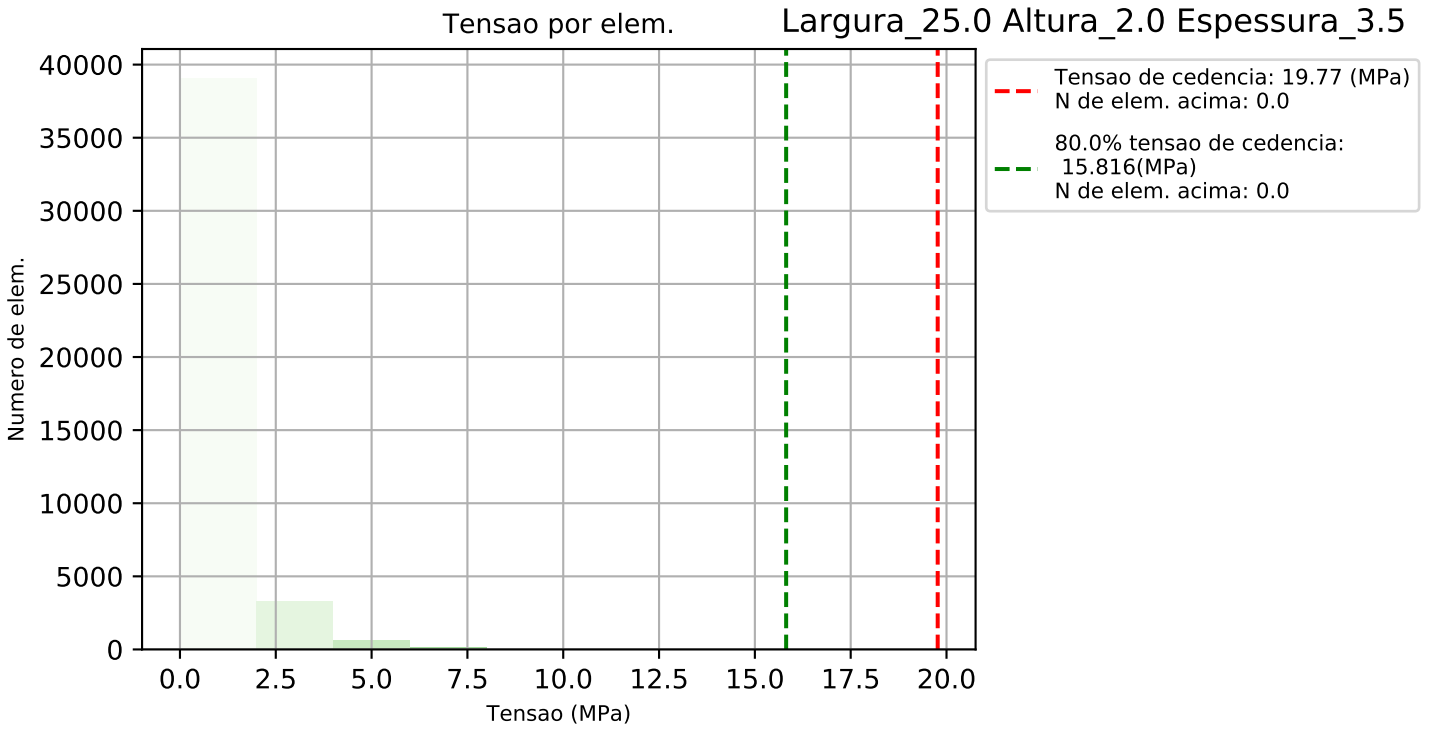

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

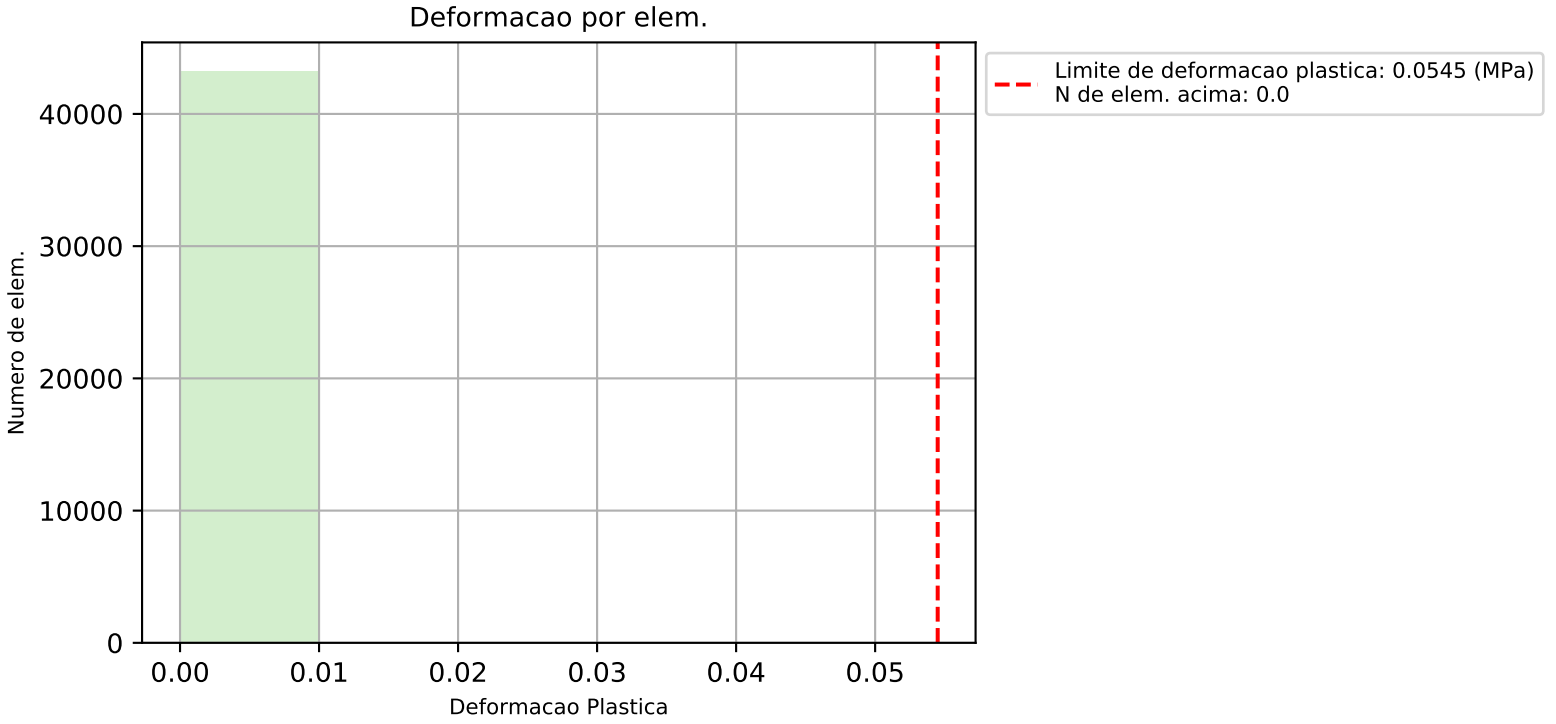

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.552e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.662e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.784e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.736e+00 (mm)
t = 2.000e+00 (s)

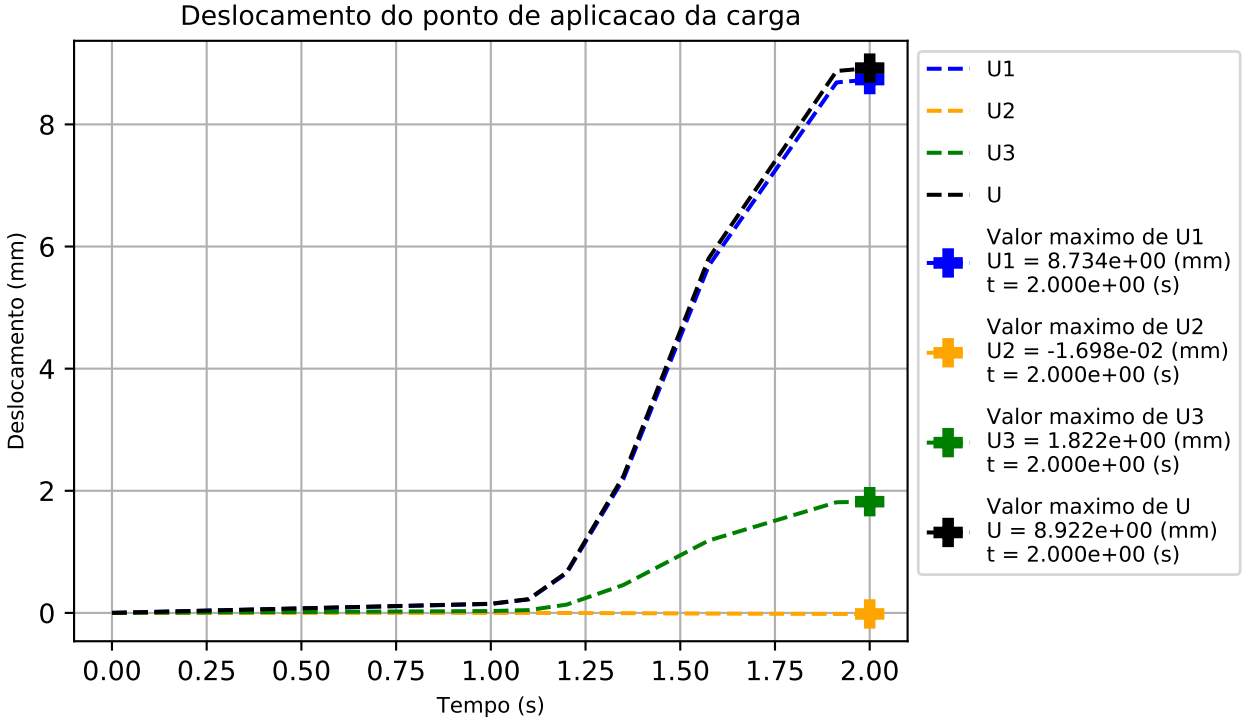

Tensao por elem.
Largura_25.0 Altura_3.0 Espessura_2.5
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

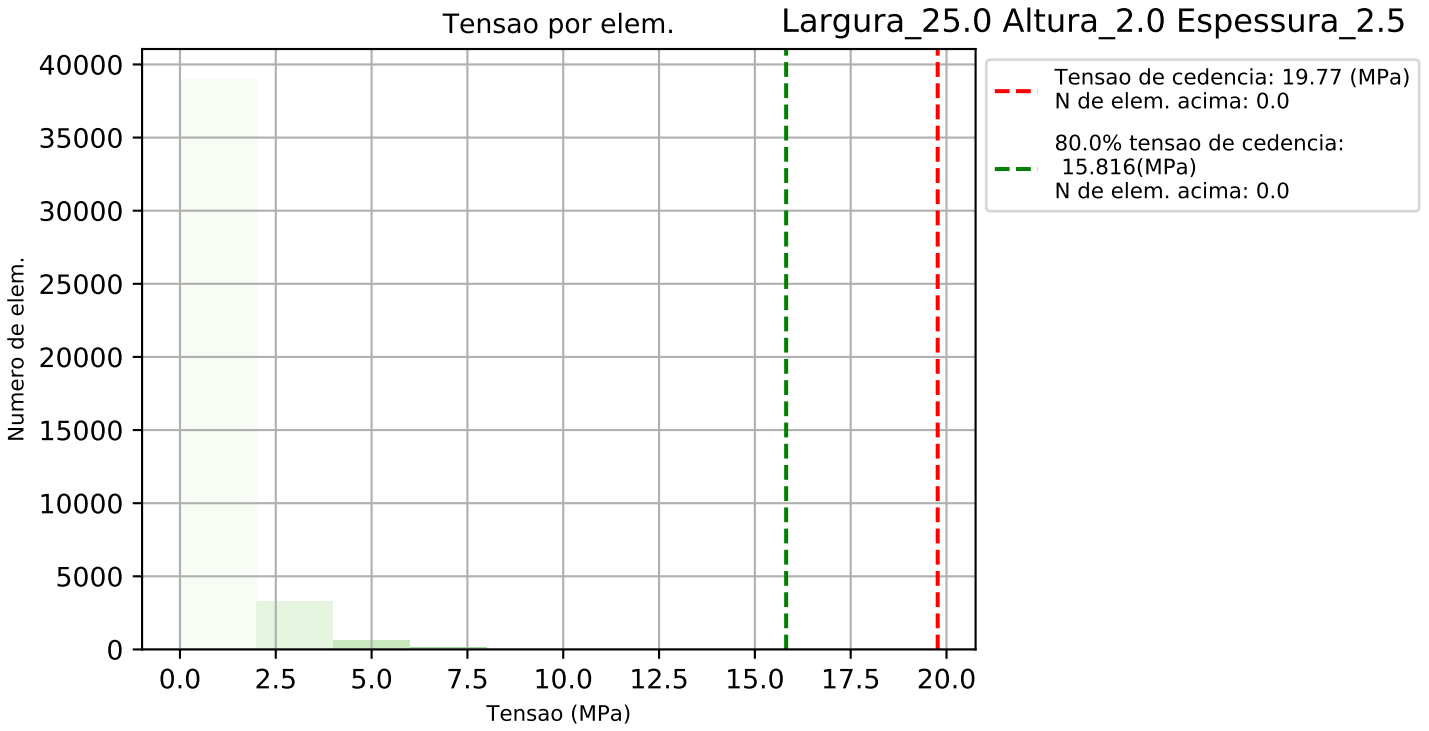

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

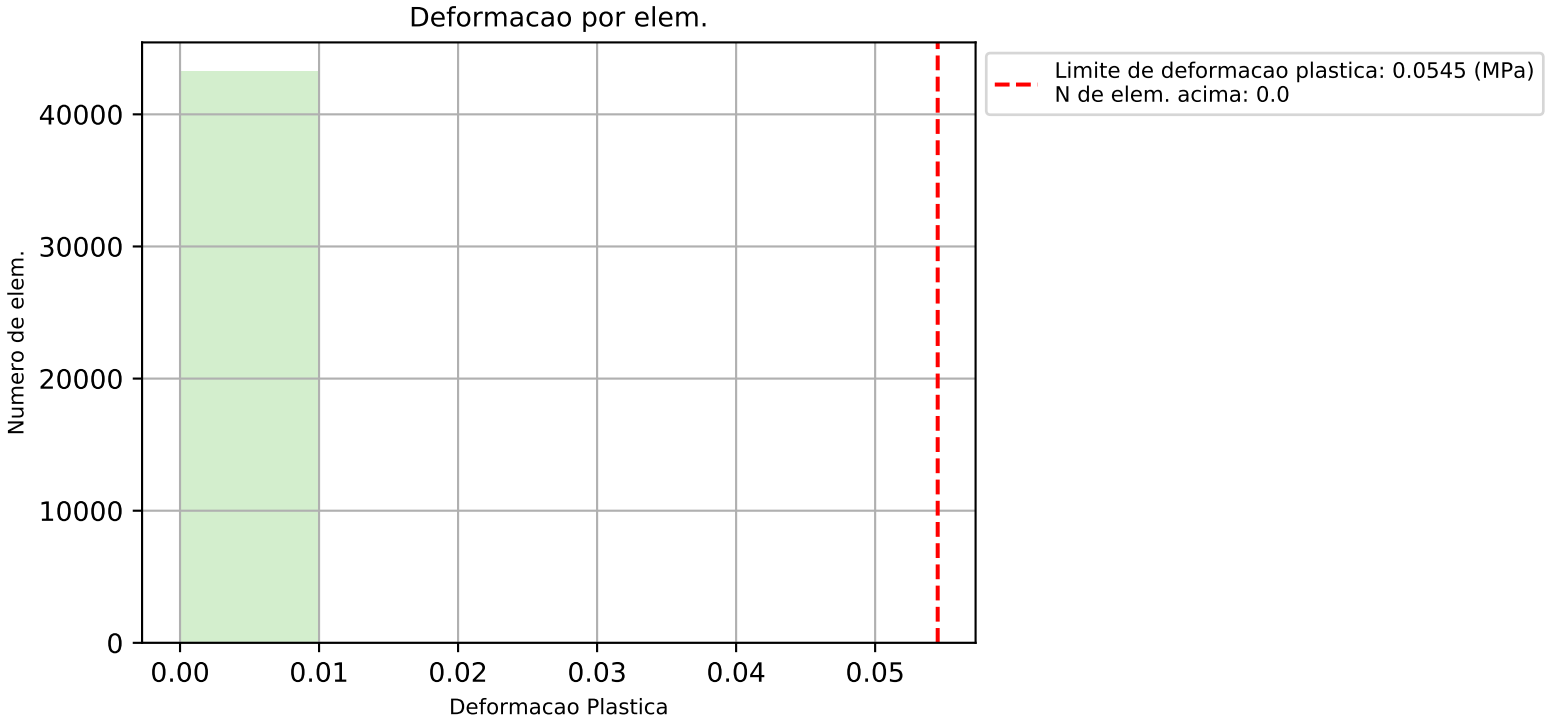

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.689e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.689e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.812e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.876e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

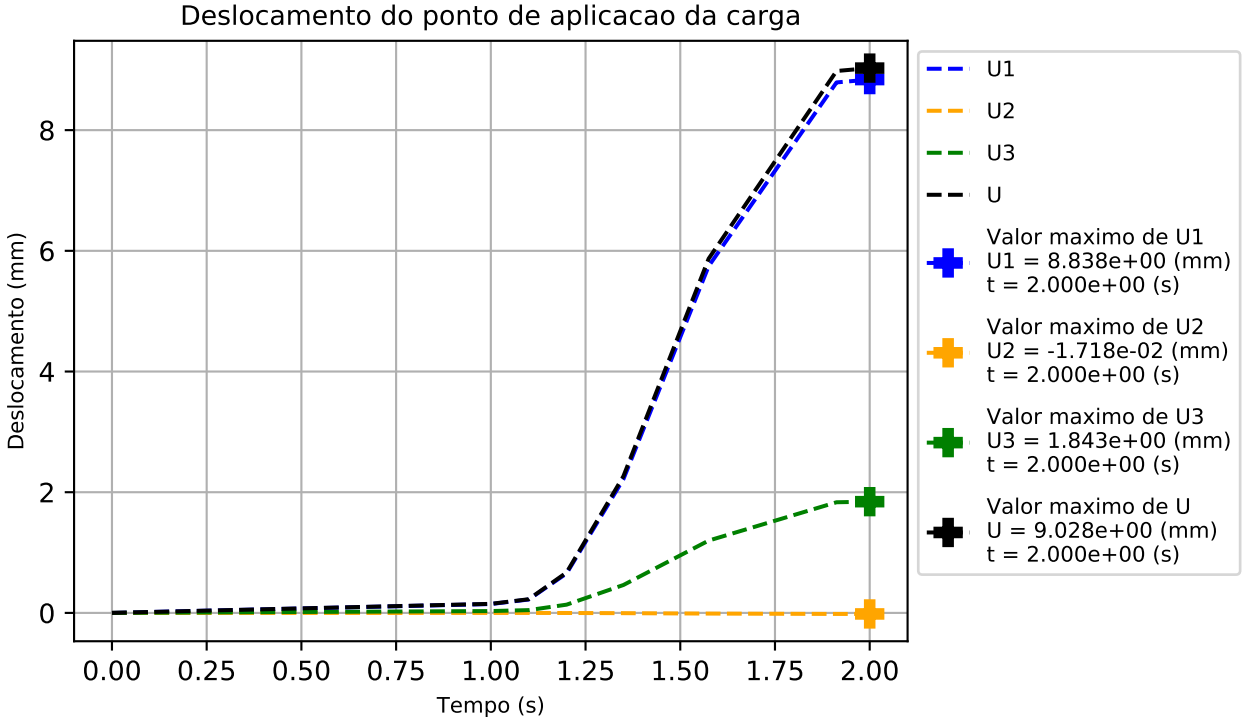

Tensao por elem.
Largura_25.0 Altura_2.0 Espessura_3.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

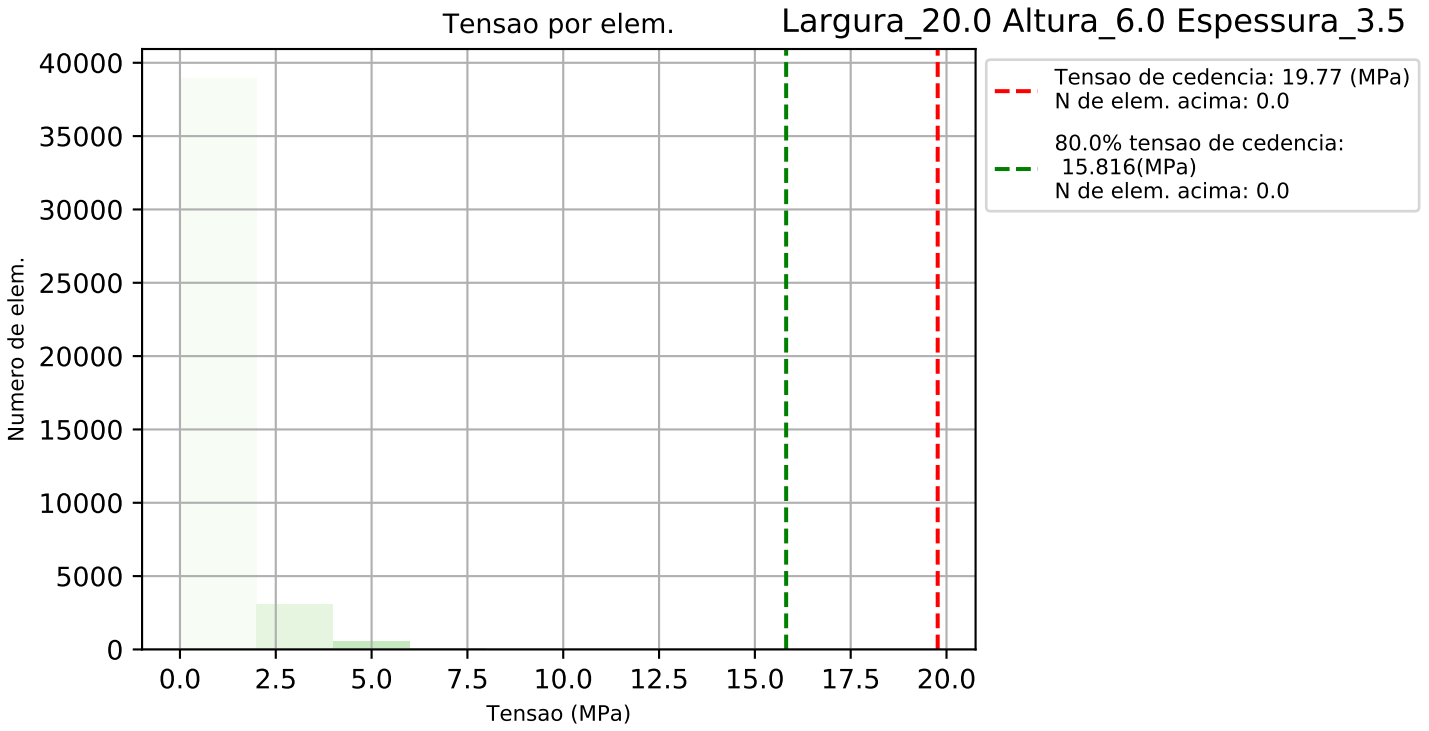

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

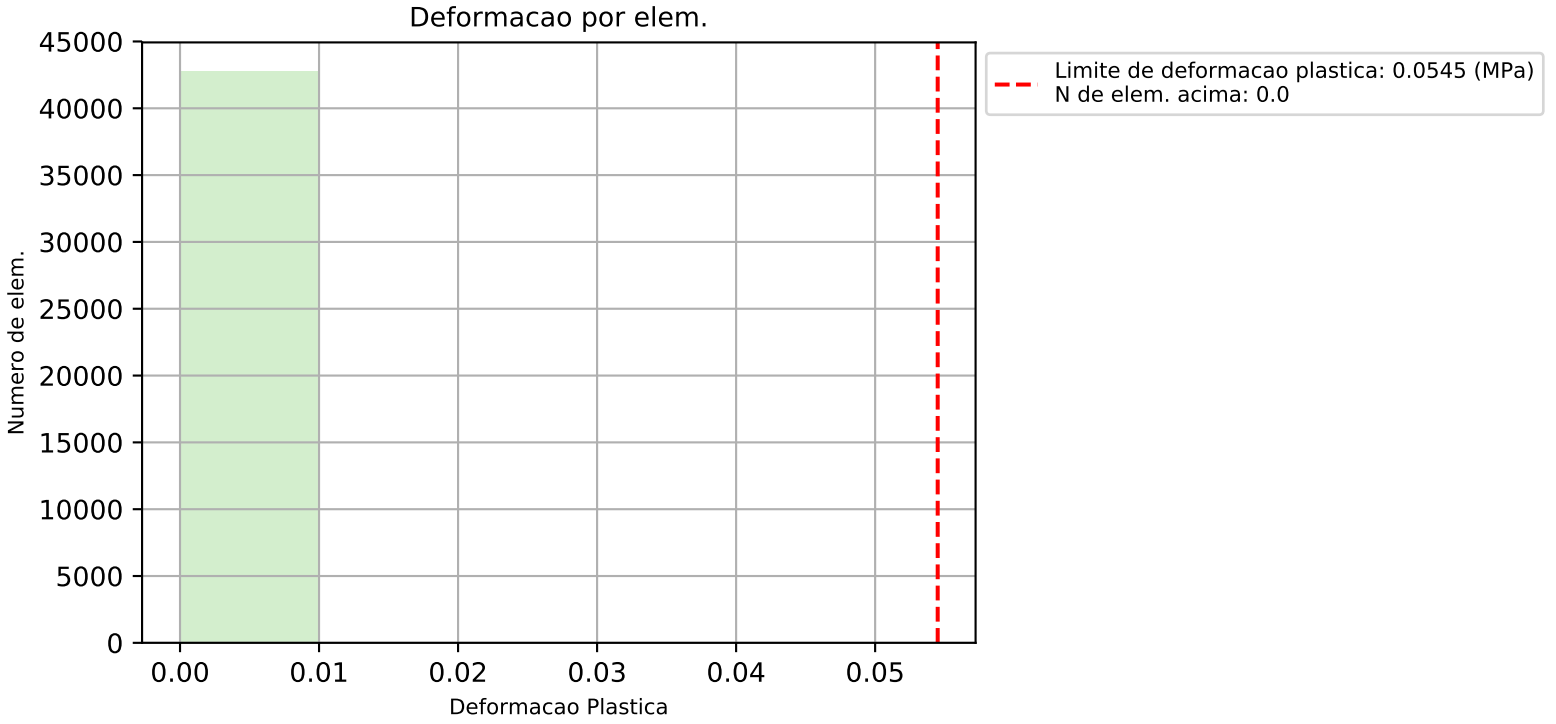

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.734e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.698e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.822e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.922e+00 (mm)
t = 2.000e+00 (s)

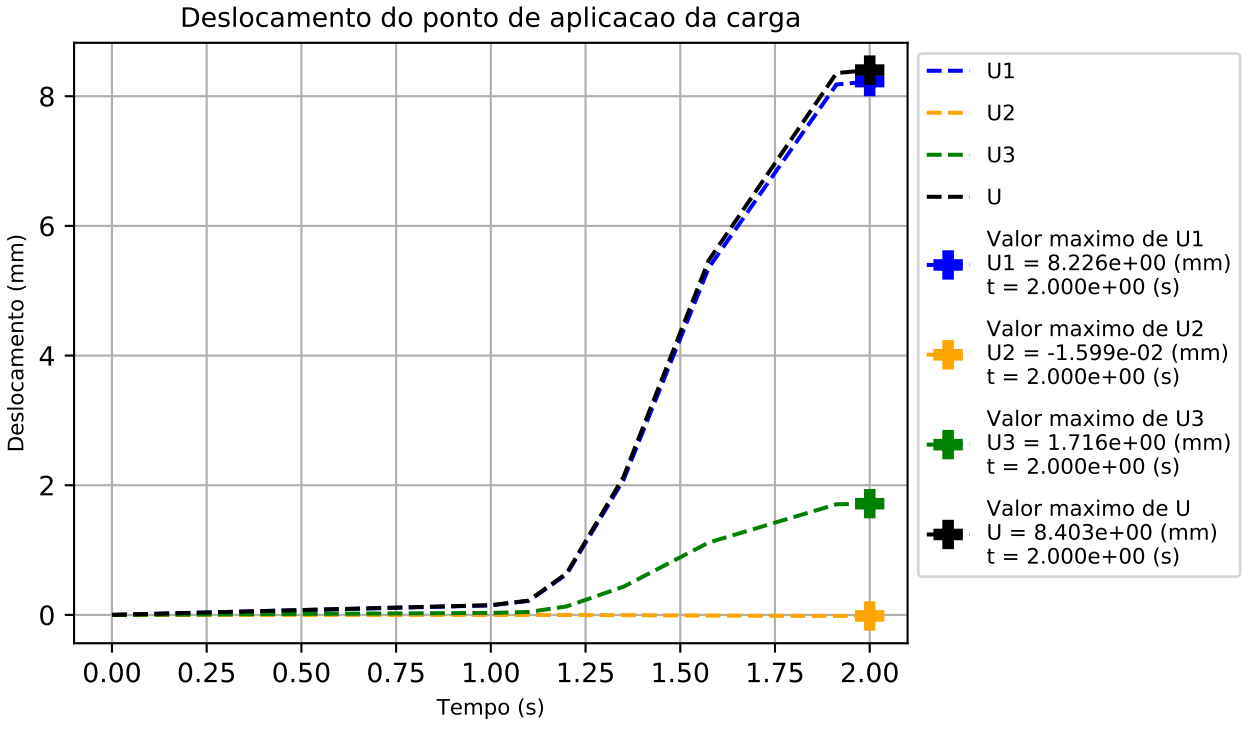

Tensao por elem.
Largura_25.0 Altura_2.0 Espessura_2.5
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

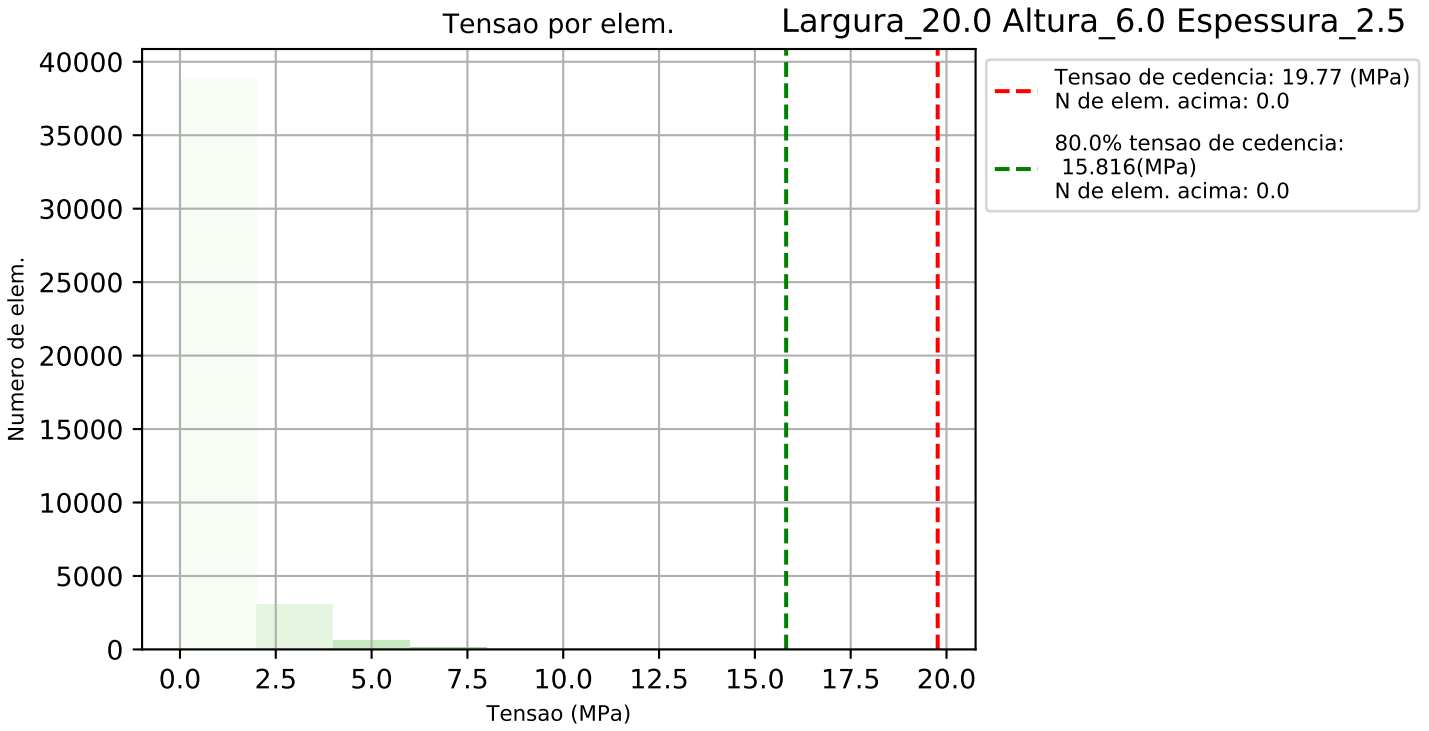

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

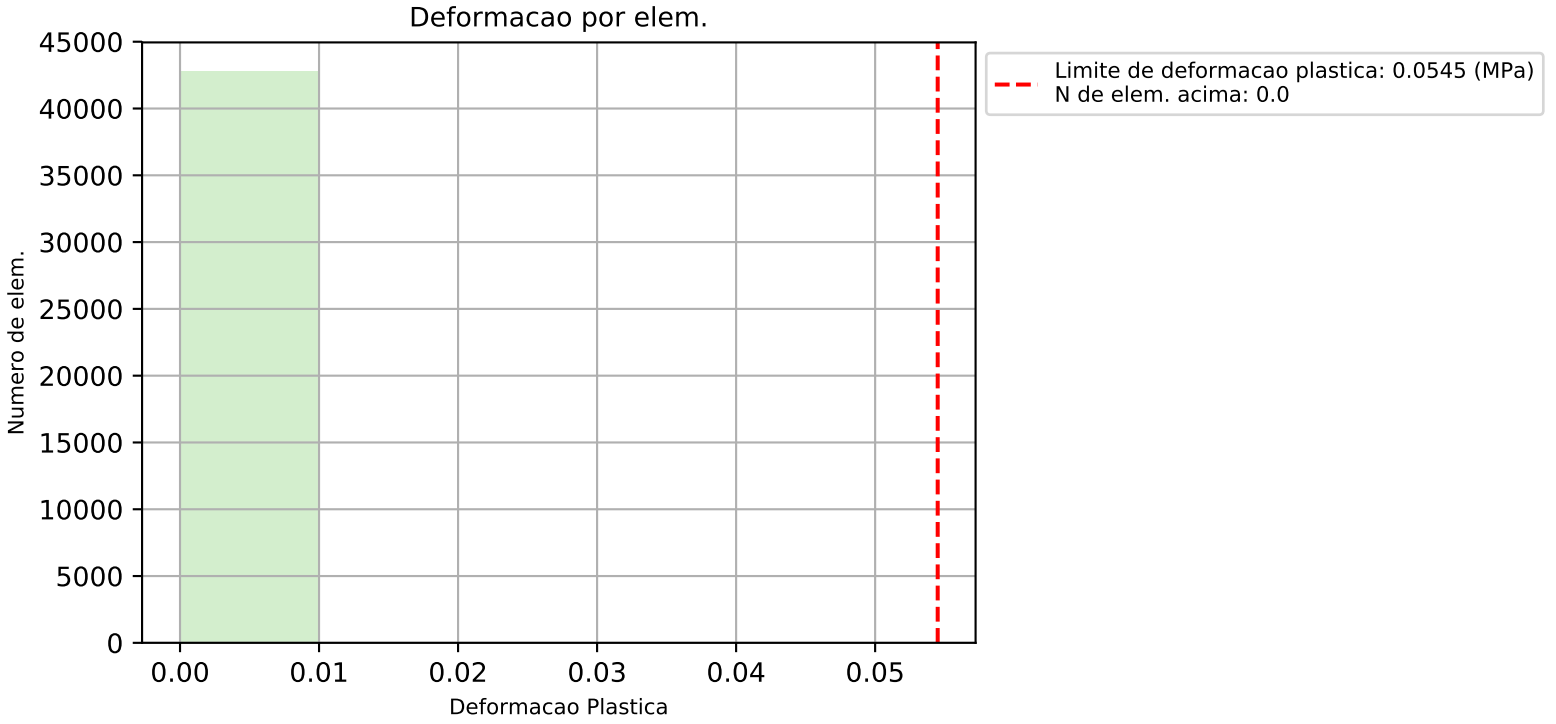

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.838e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.718e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.843e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 9.028e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

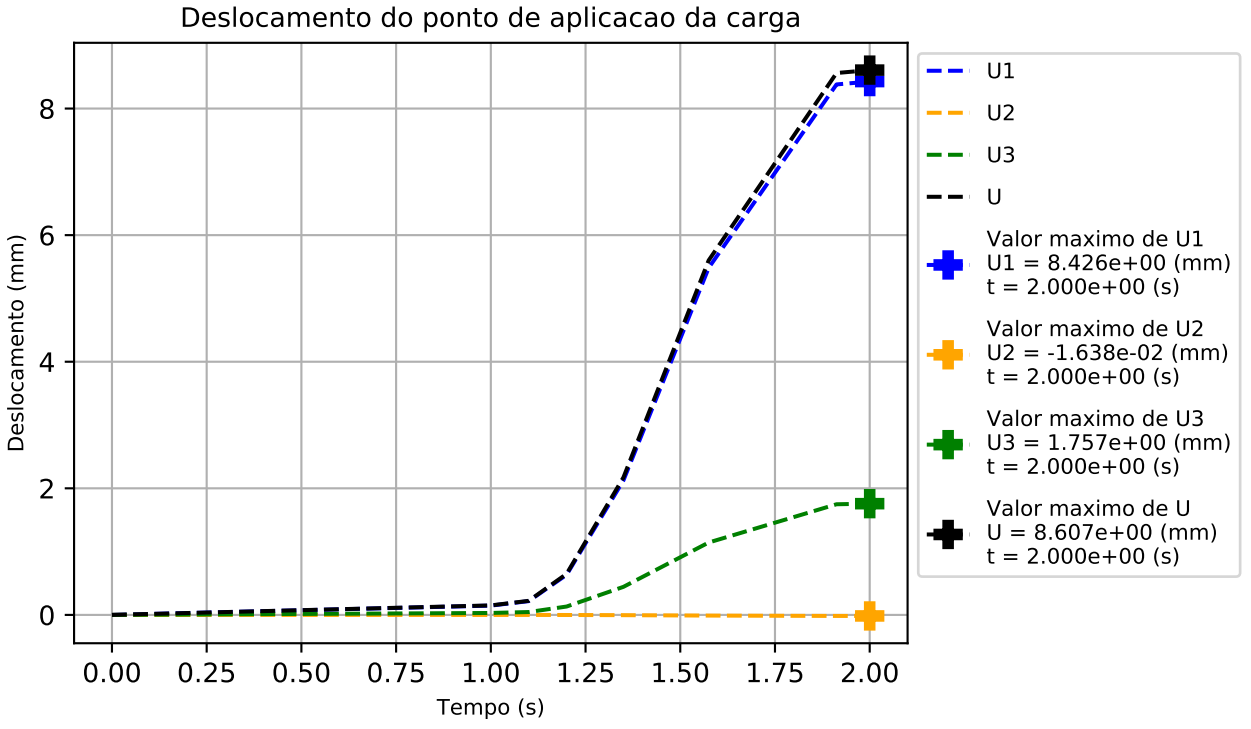

Tensao por elem.
Largura_20.0 Altura_6.0 Espessura_3.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

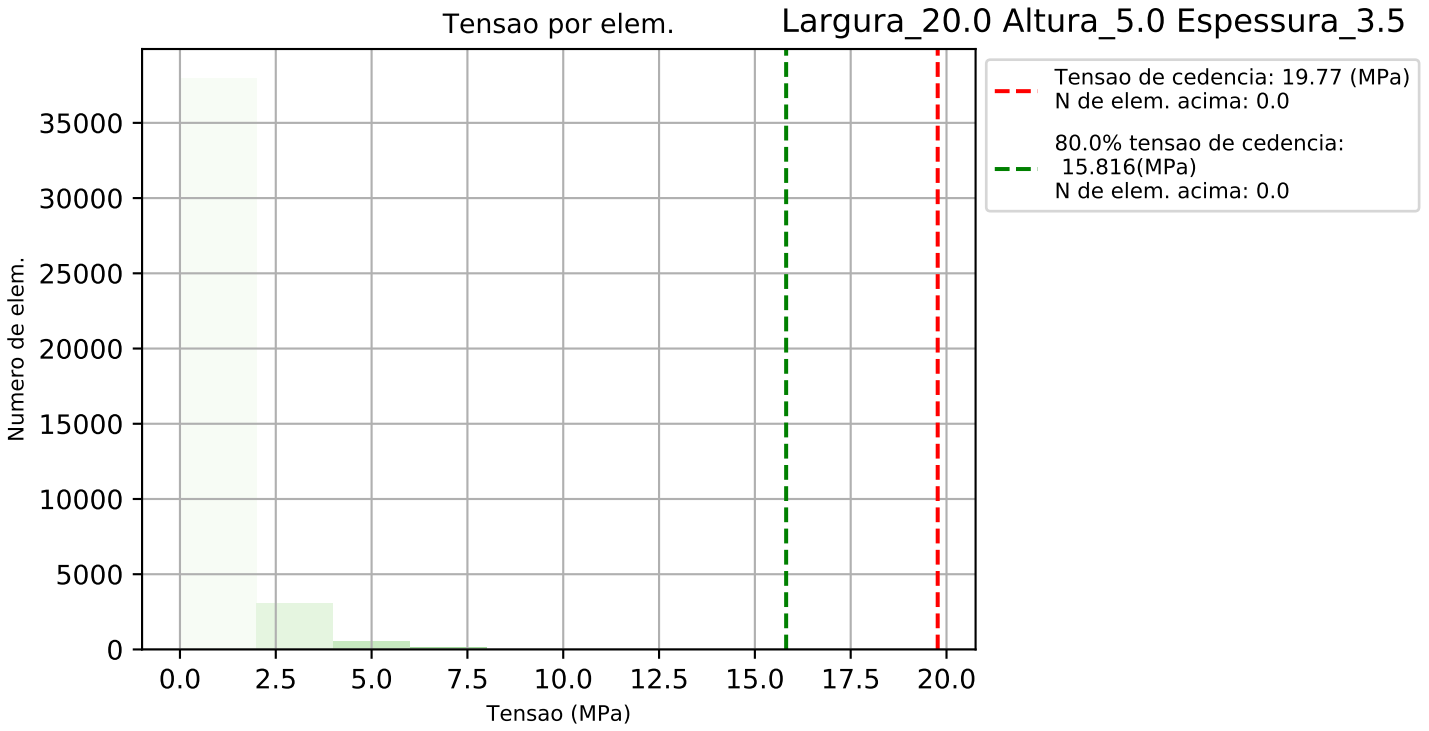

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

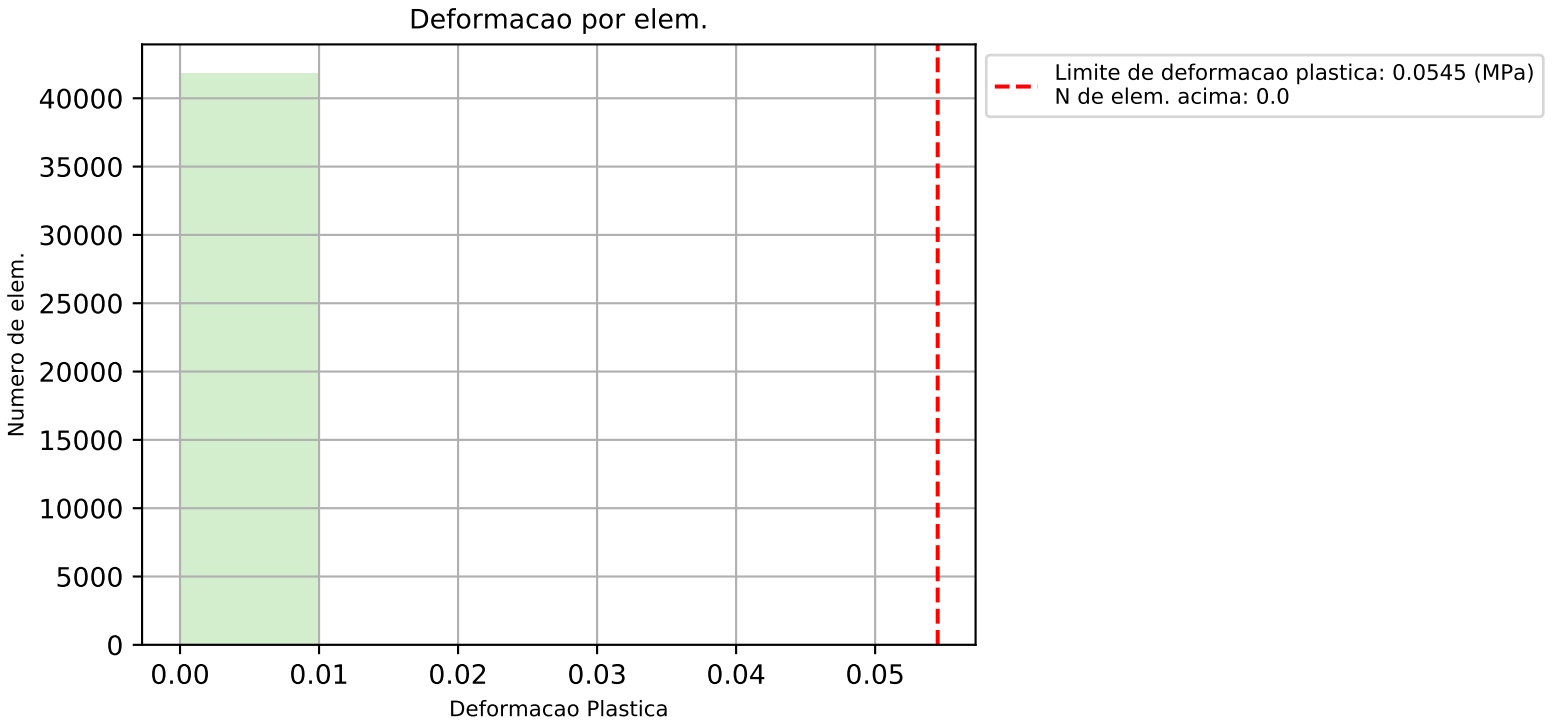

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.226e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.599e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.716e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.403e+00 (mm)
t = 2.000e+00 (s)

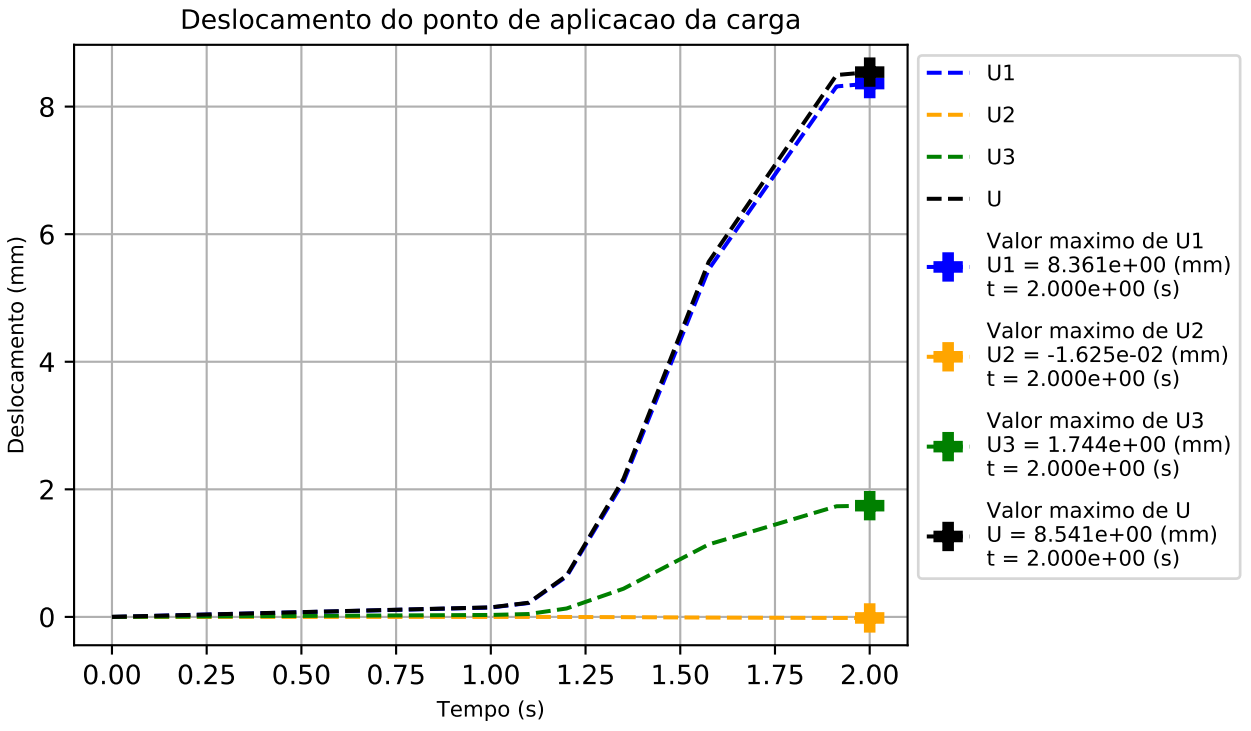

Tensao por elem.
Largura_20.0 Altura_6.0 Espessura_2.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

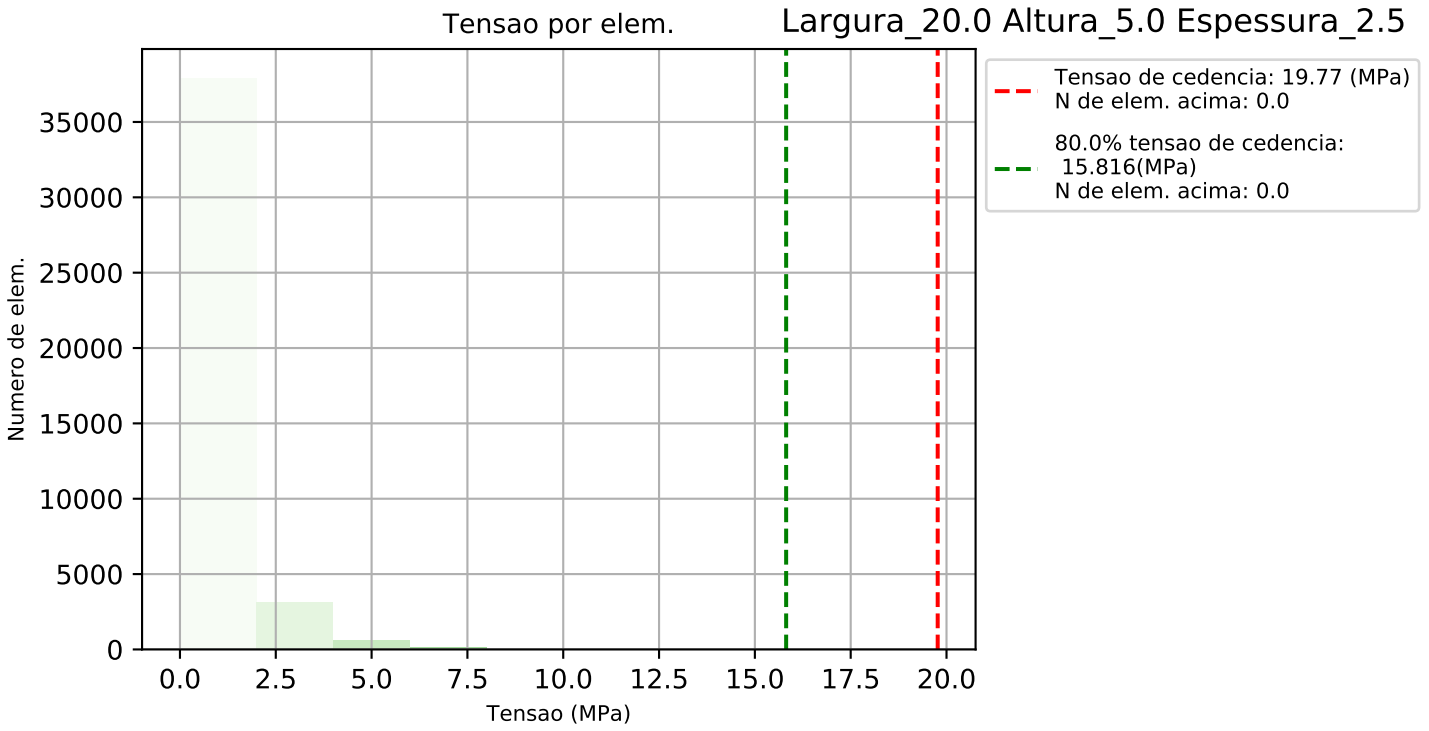

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

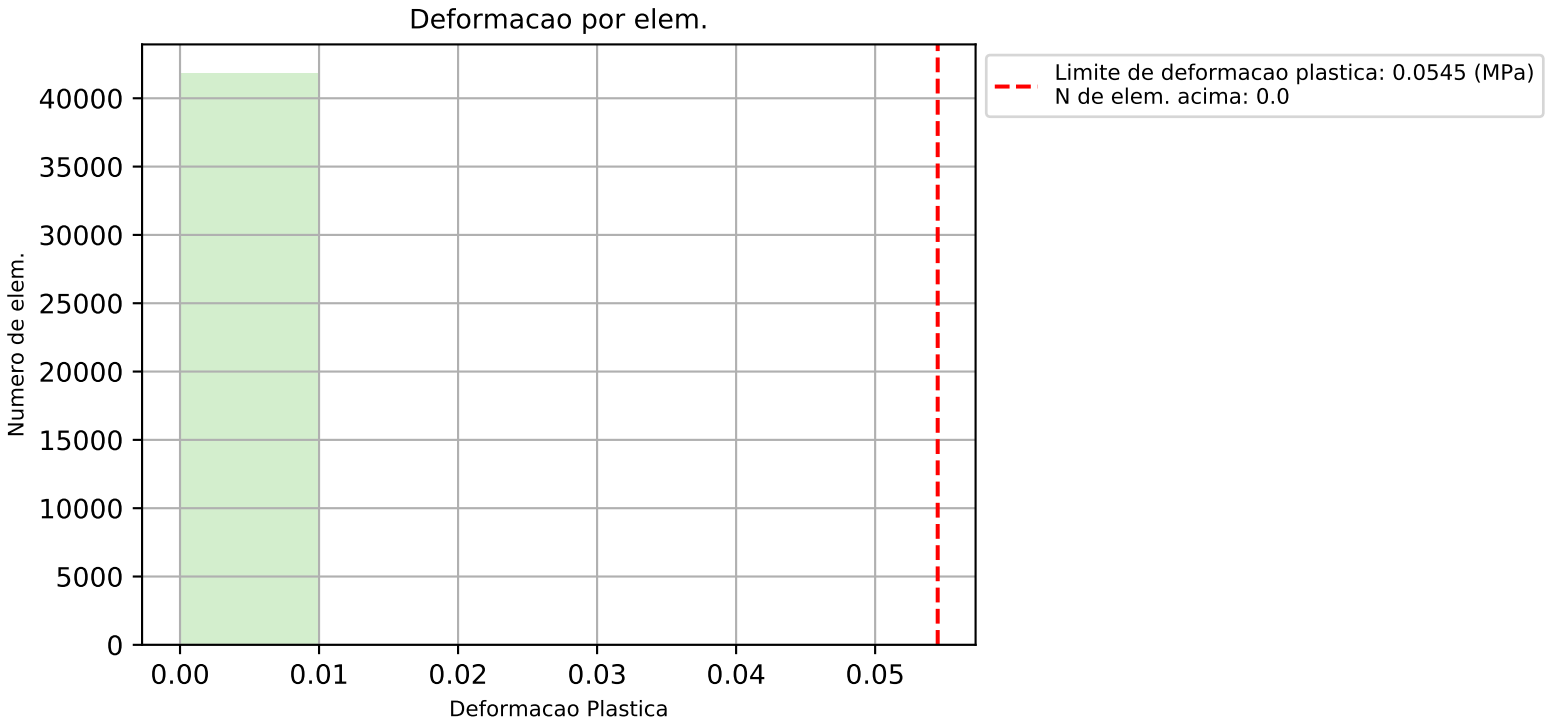

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.426e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.638e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.757e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.607e+00 (mm)
t = 2.000e+00 (s)

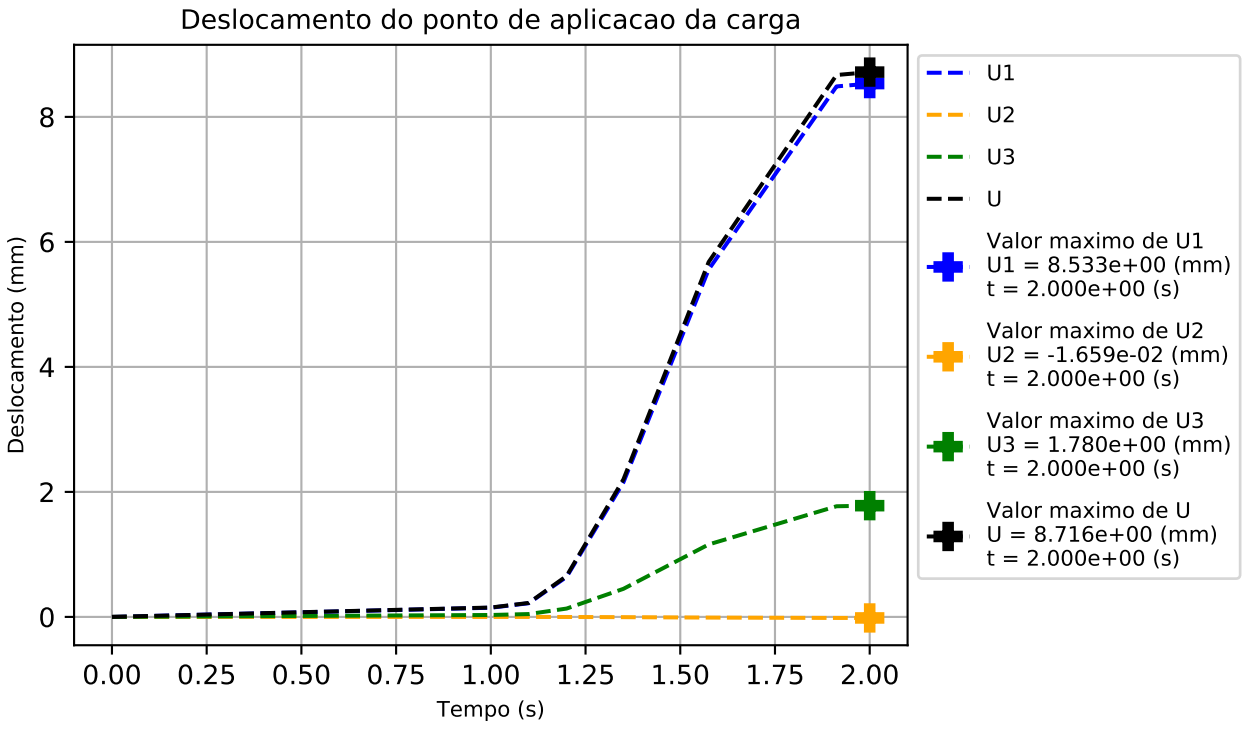

Tensao por elem.
Largura_20.0 Altura_5.0 Espessura_3.5
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

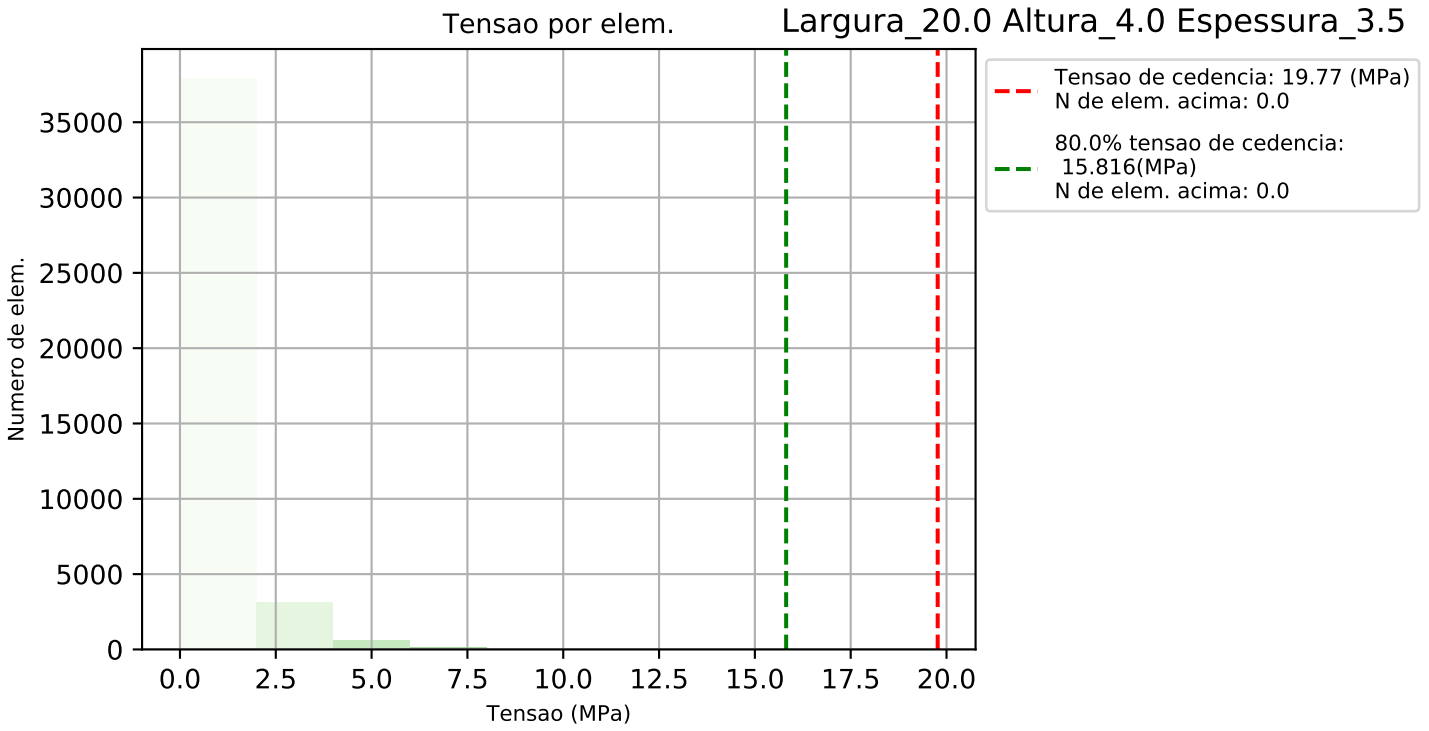

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

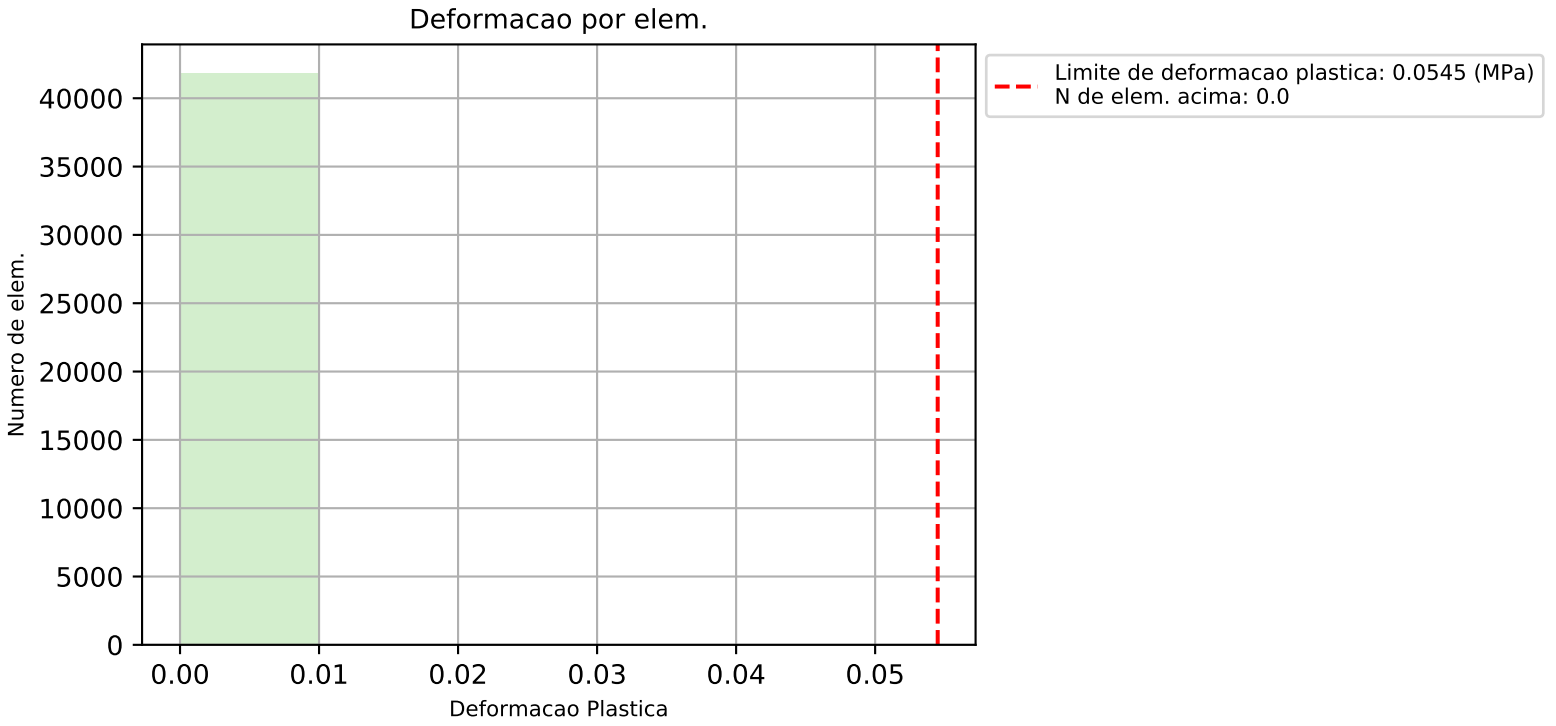

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.361e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.625e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.744e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.541e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

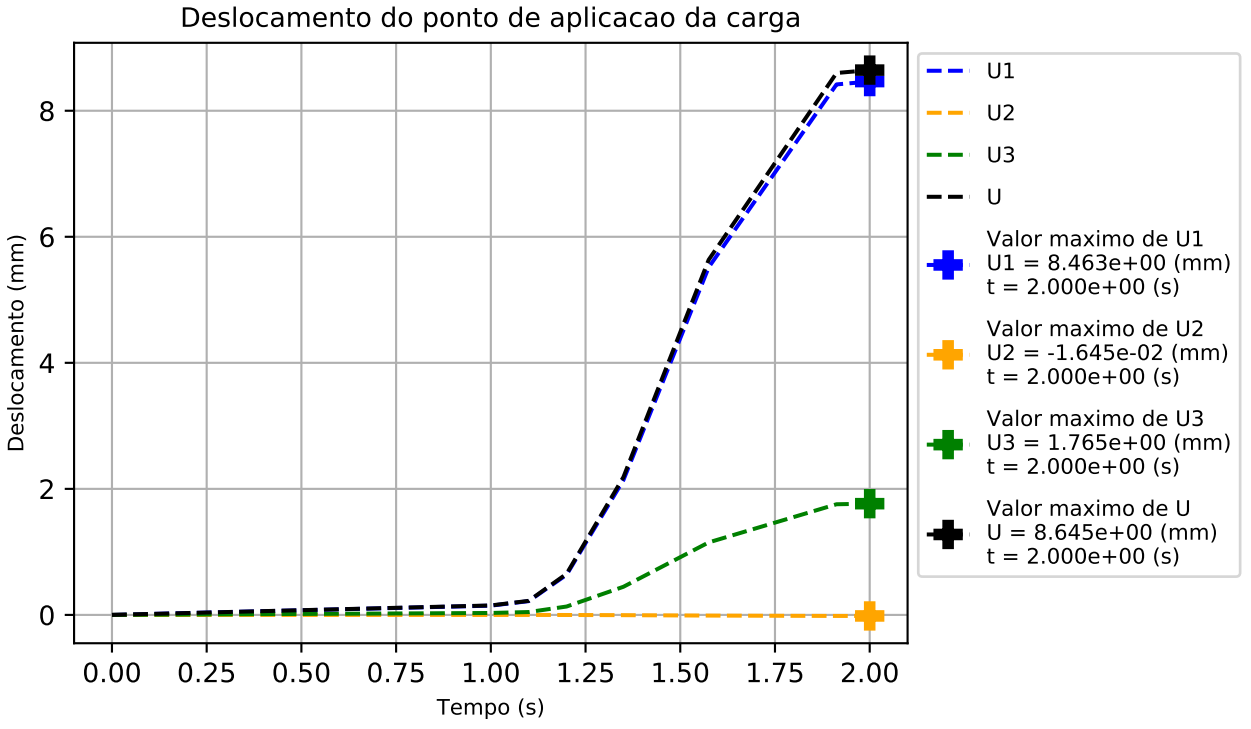

Tensao por elem.
Largura_20.0 Altura_5.0 Espessura_2.5
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

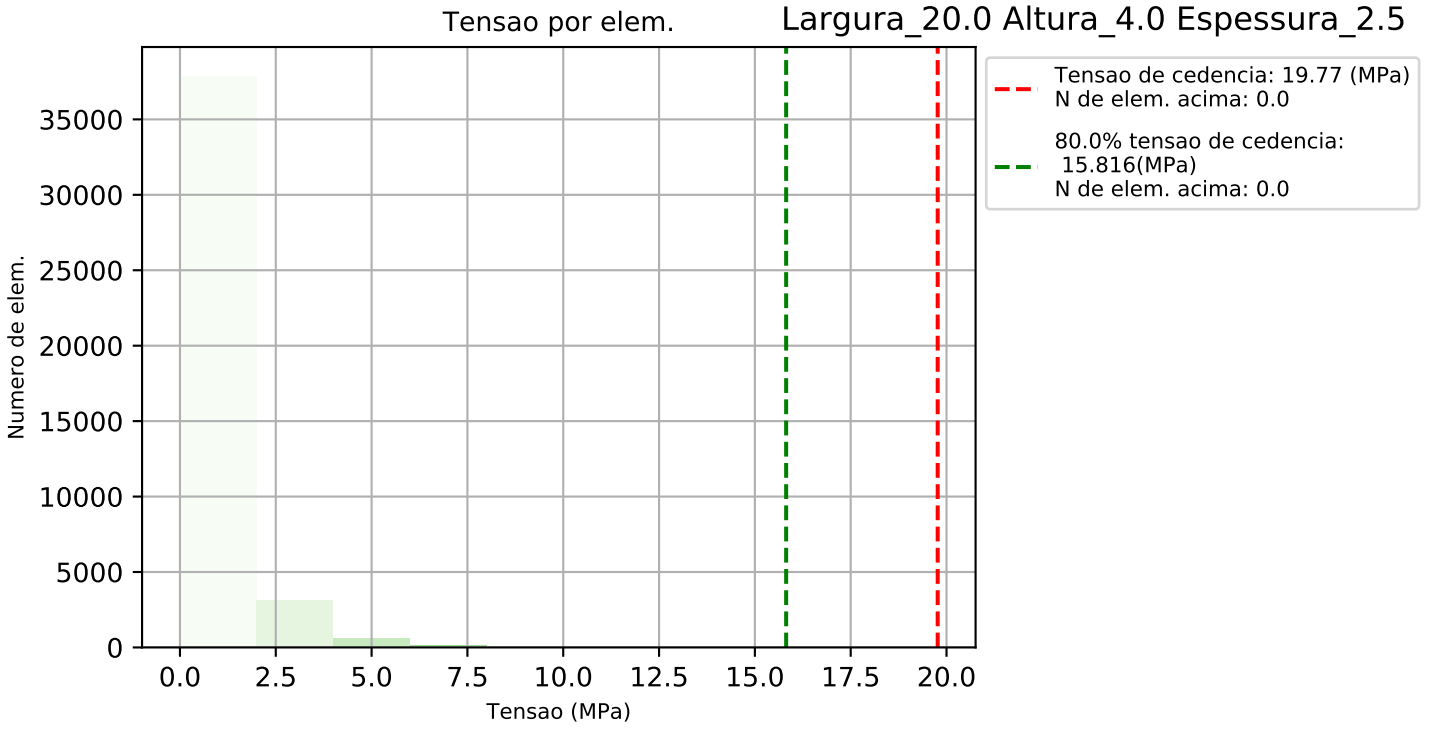

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

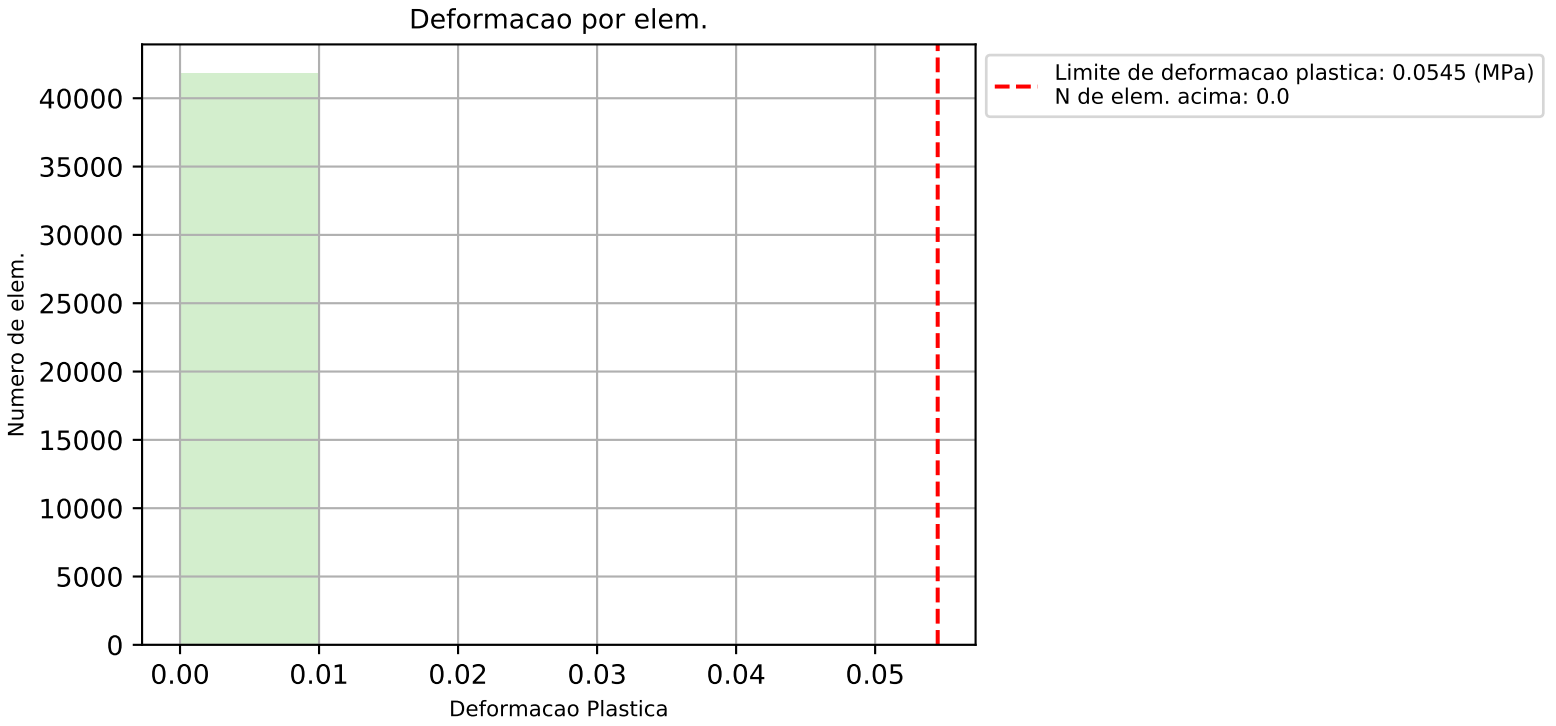

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.533e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.659e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.780e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.716e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

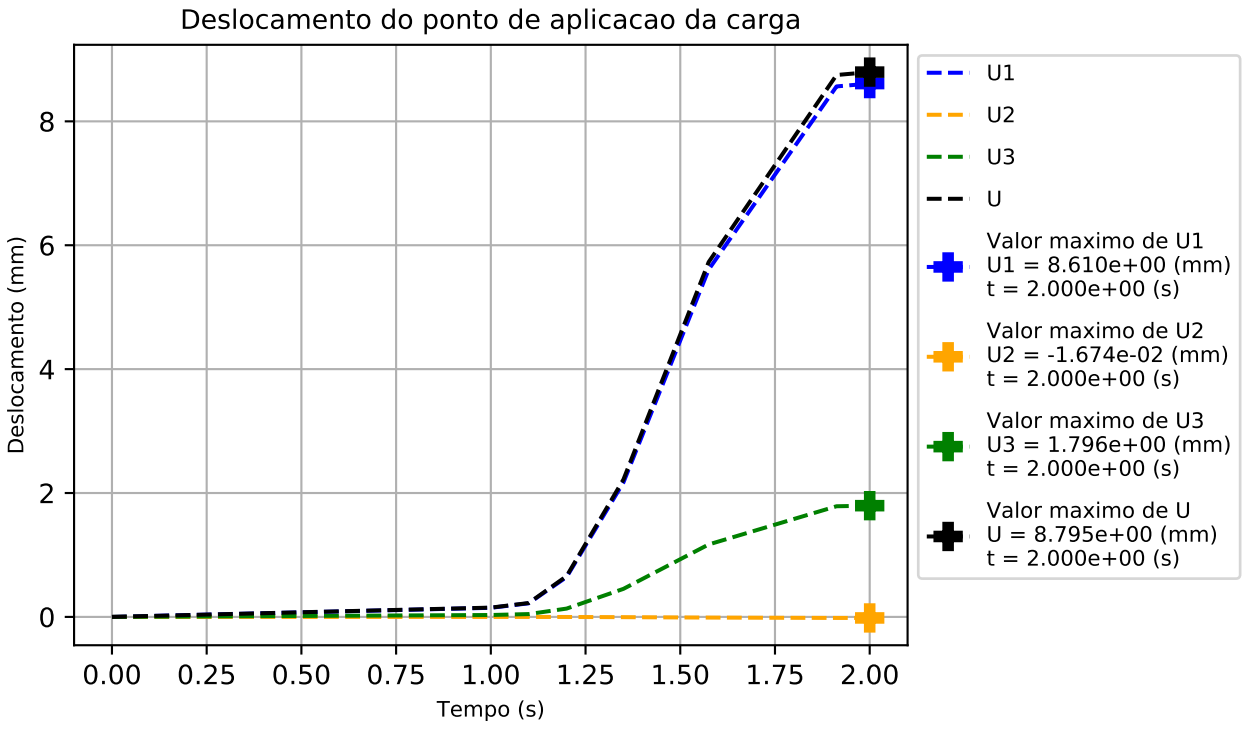

Tensao por elem.
Largura_20.0 Altura_4.0 Espessura_3.5
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

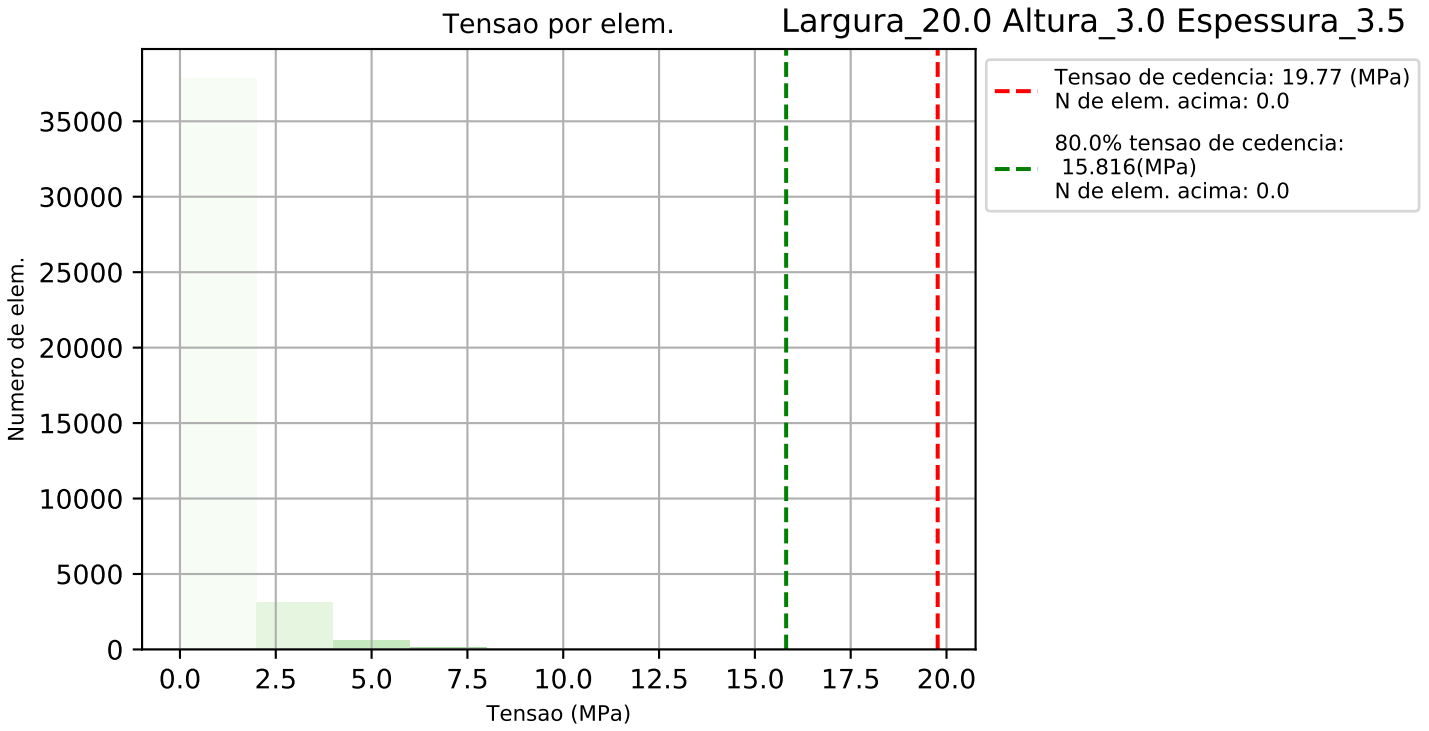

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

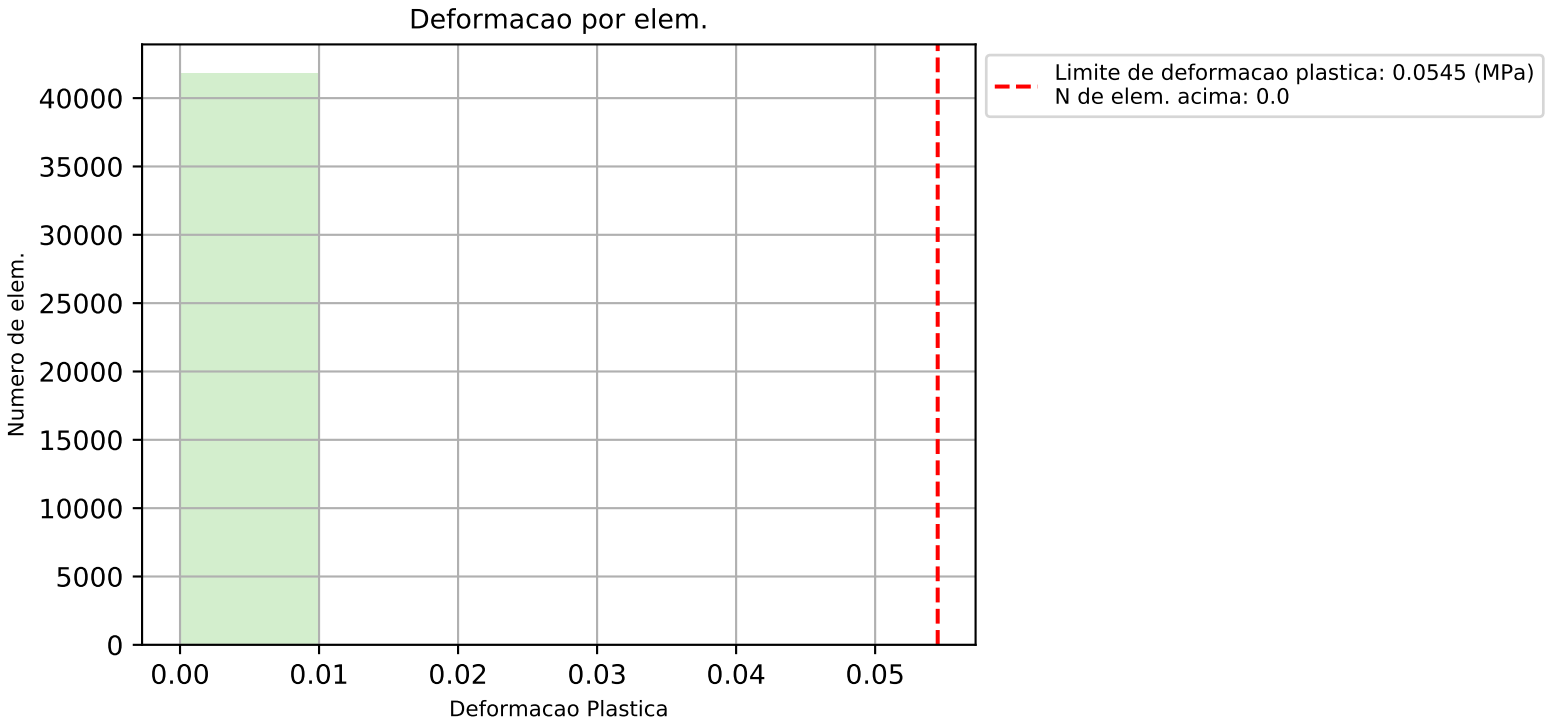

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.463e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.645e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.765e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.645e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

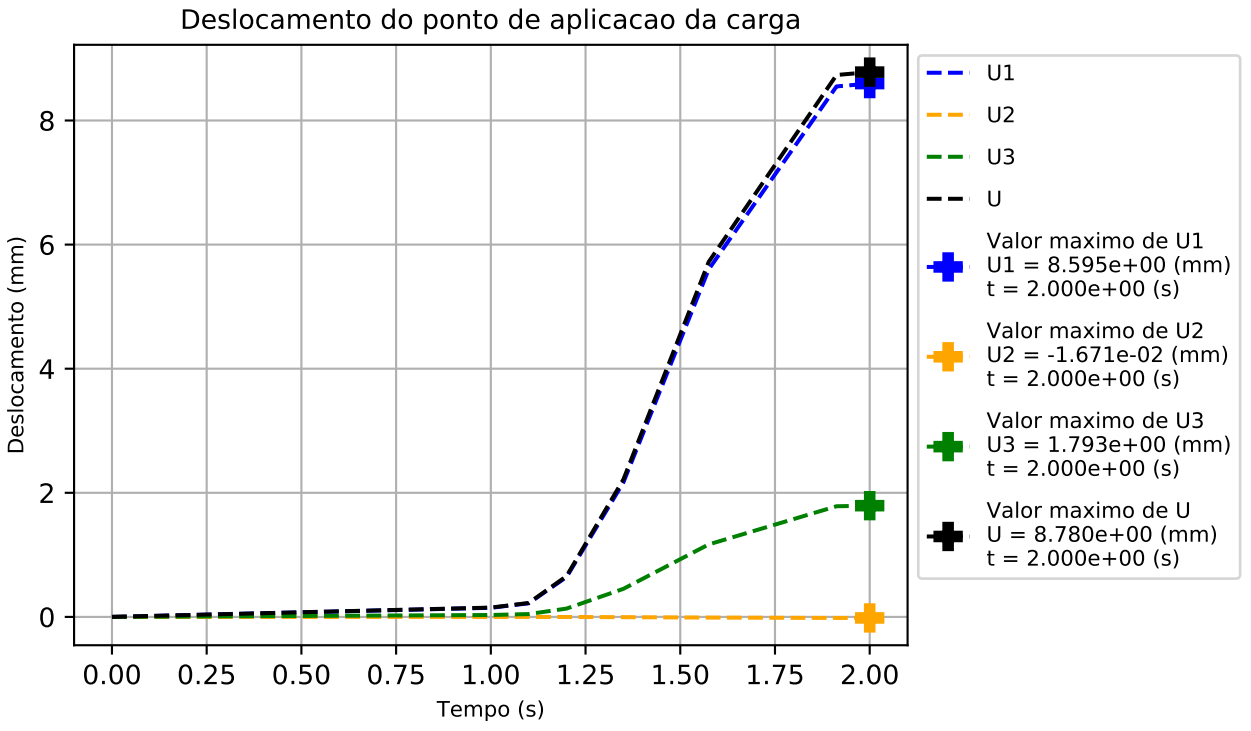

Tensao por elem.
Largura_20.0 Altura_4.0 Espessura_2.5
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

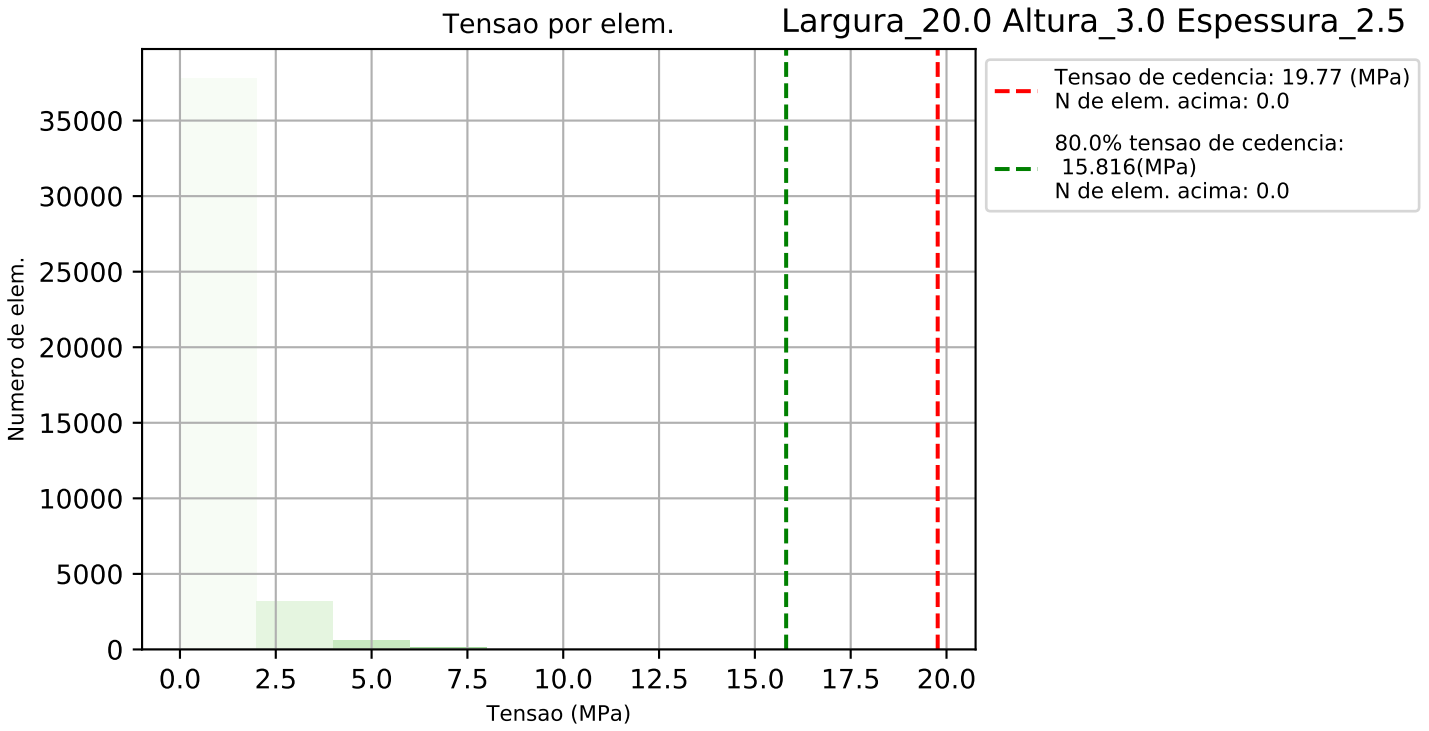

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

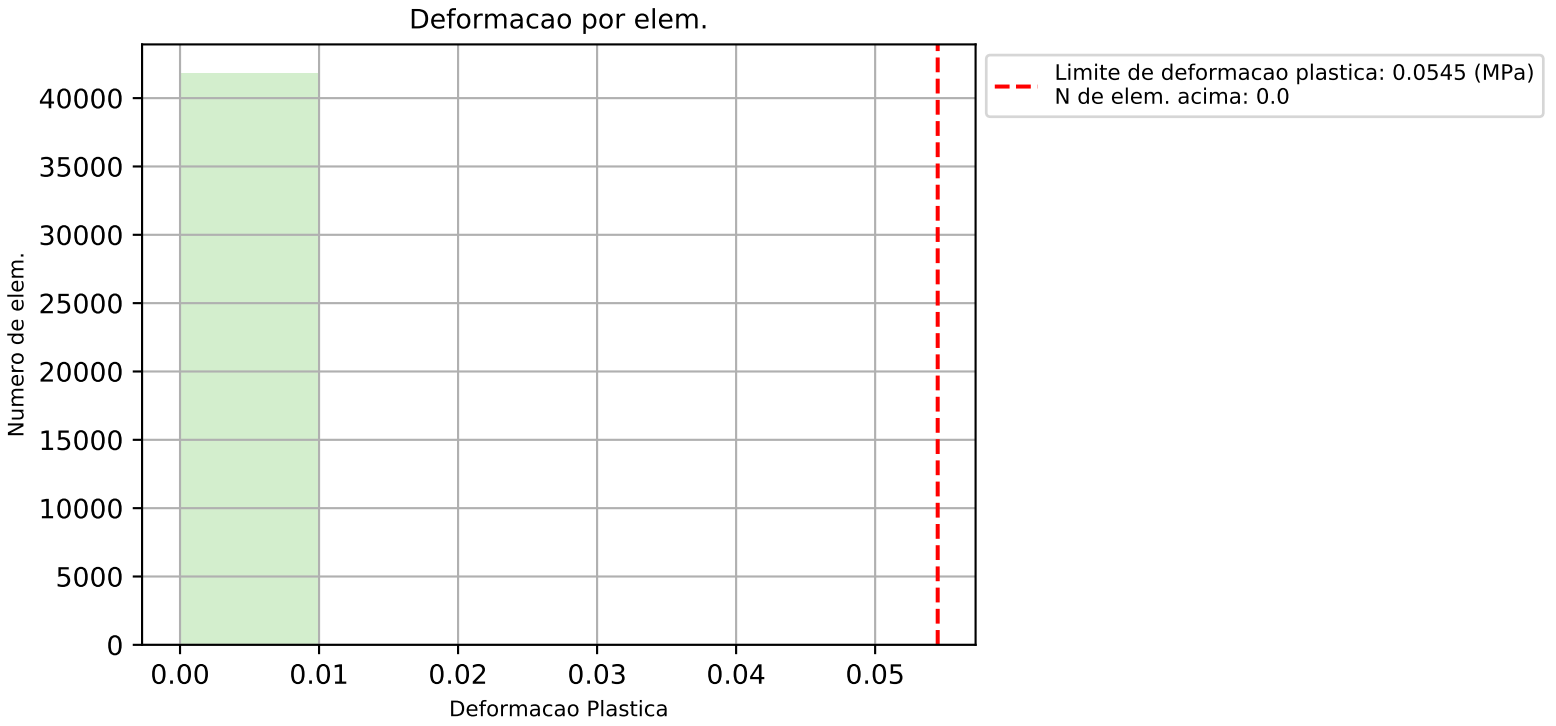

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.610e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.674e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.796e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.795e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

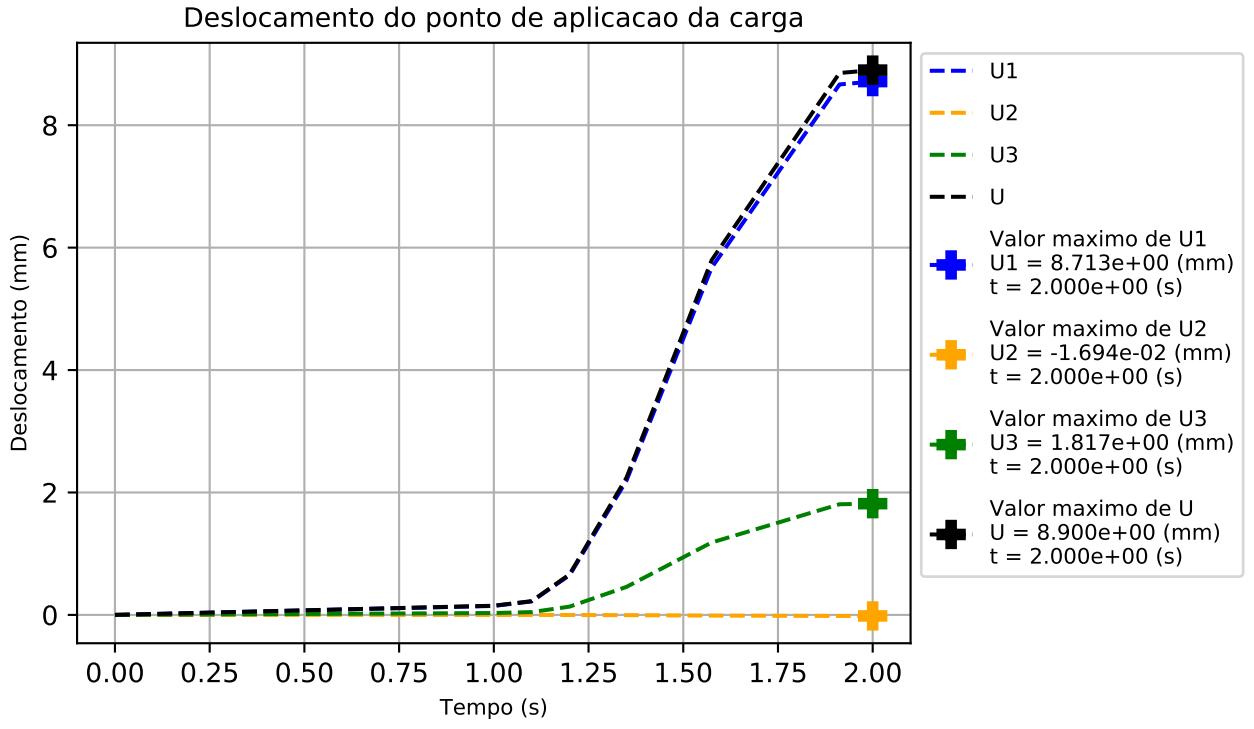

Tensao por elem.
Largura_20.0 Altura_3.0 Espessura_3.5
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

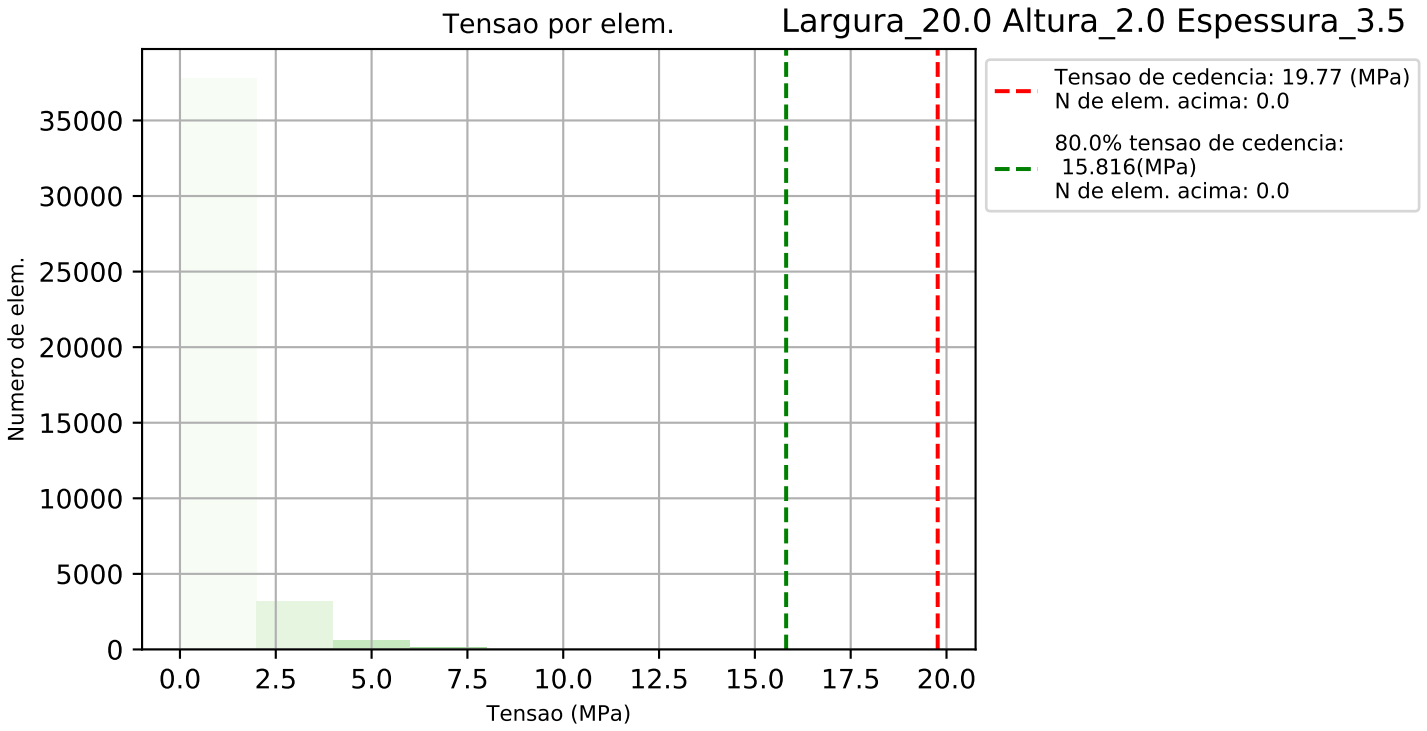

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

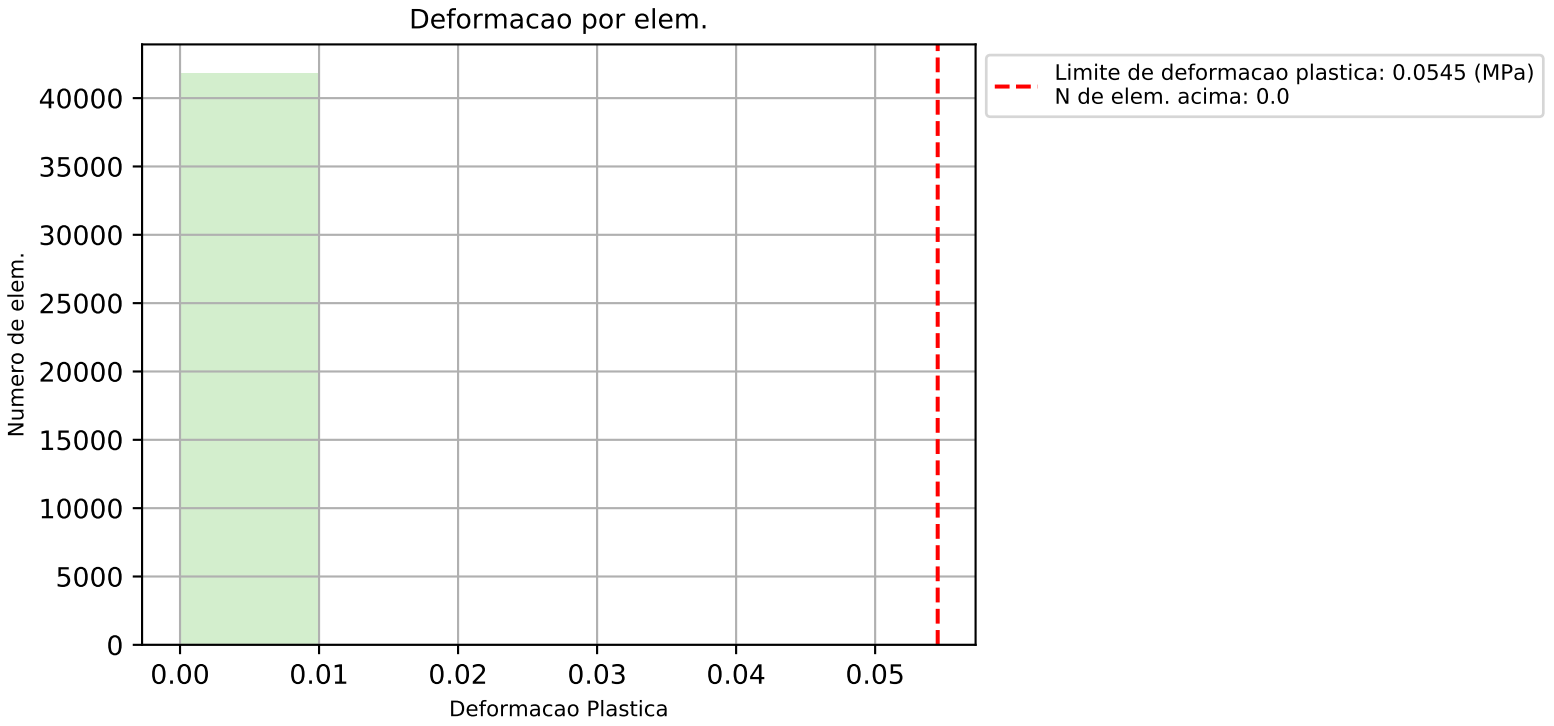

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.595e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.671e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.793e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.780e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

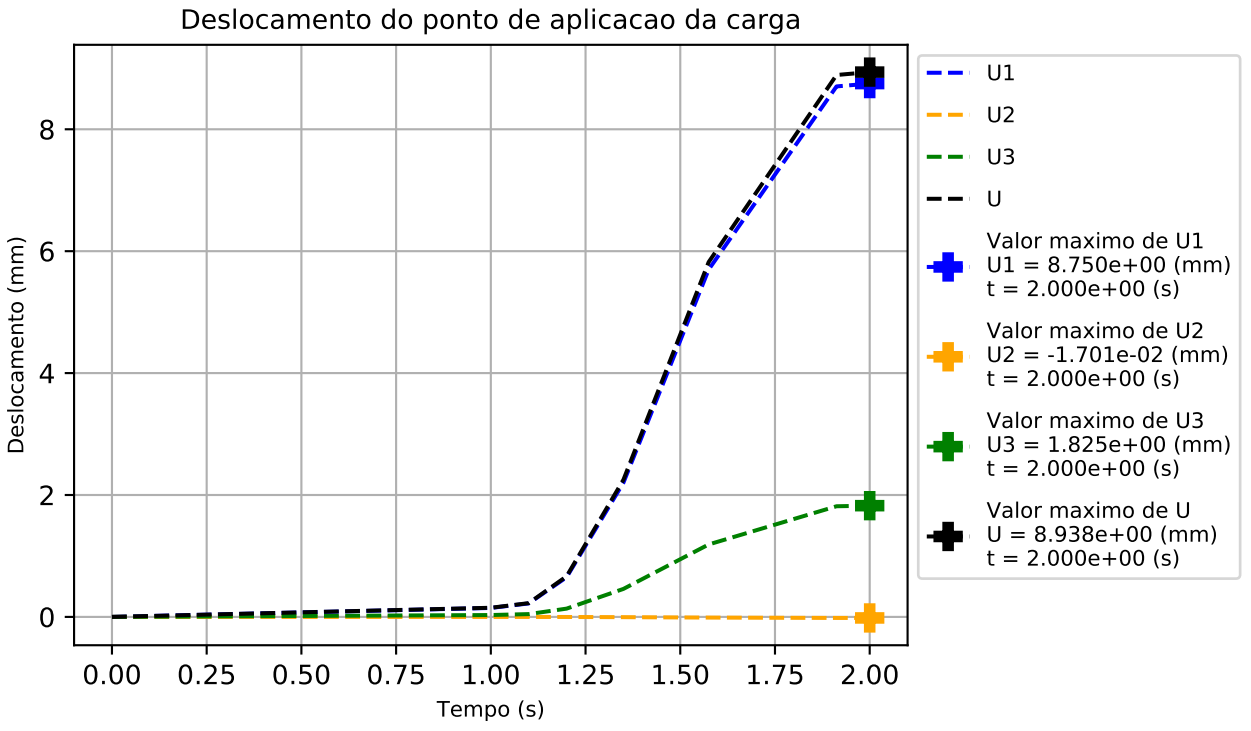

Tensao por elem.
Largura_20.0 Altura_3.0 Espessura_2.5
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

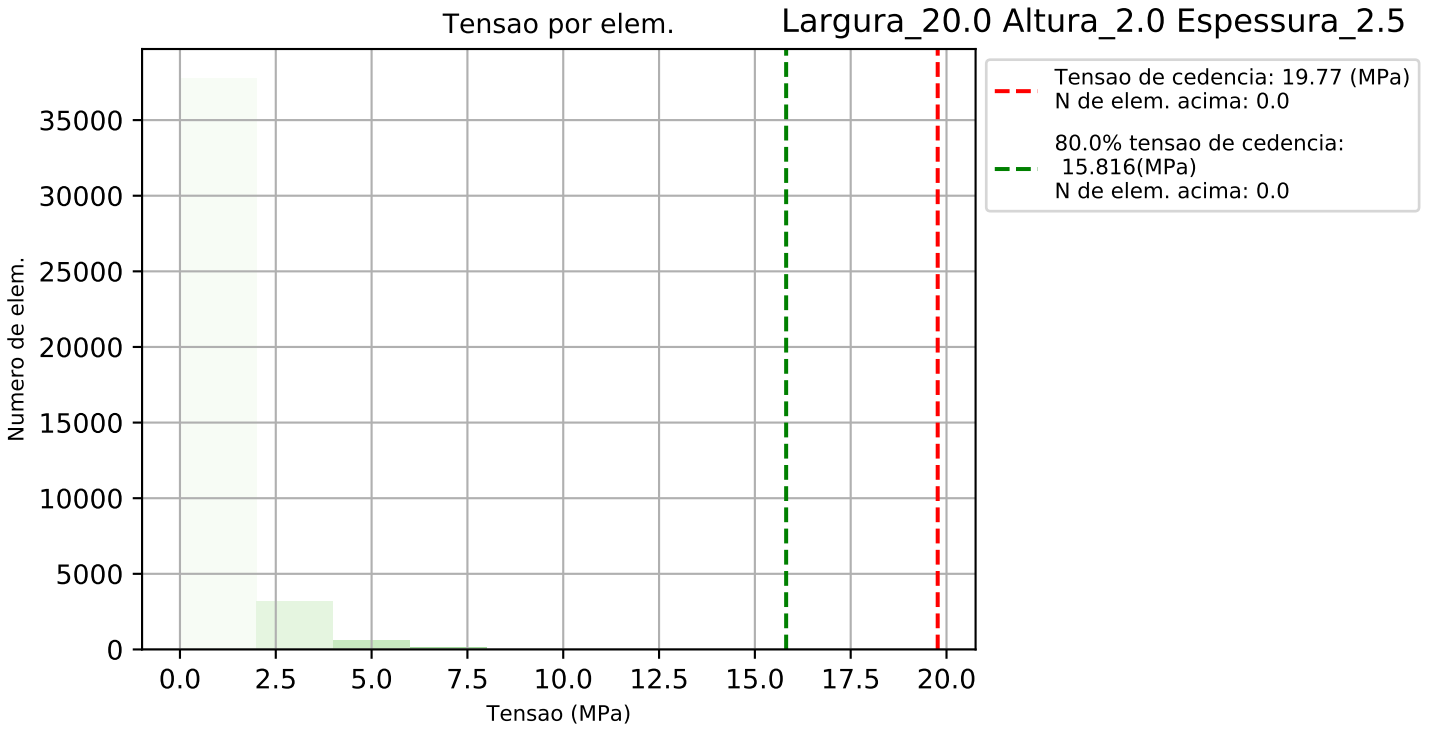

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

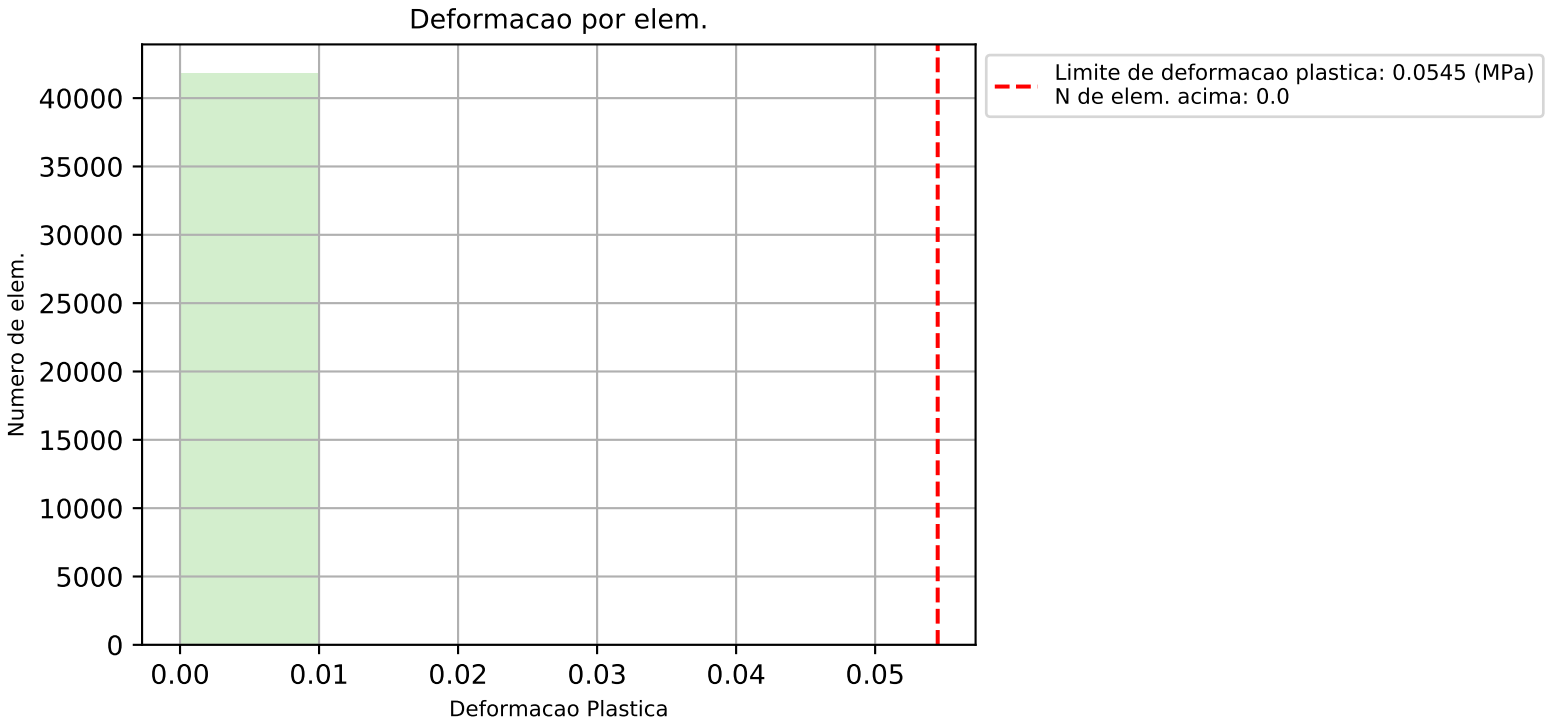

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.713e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.694e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.817e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.900e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

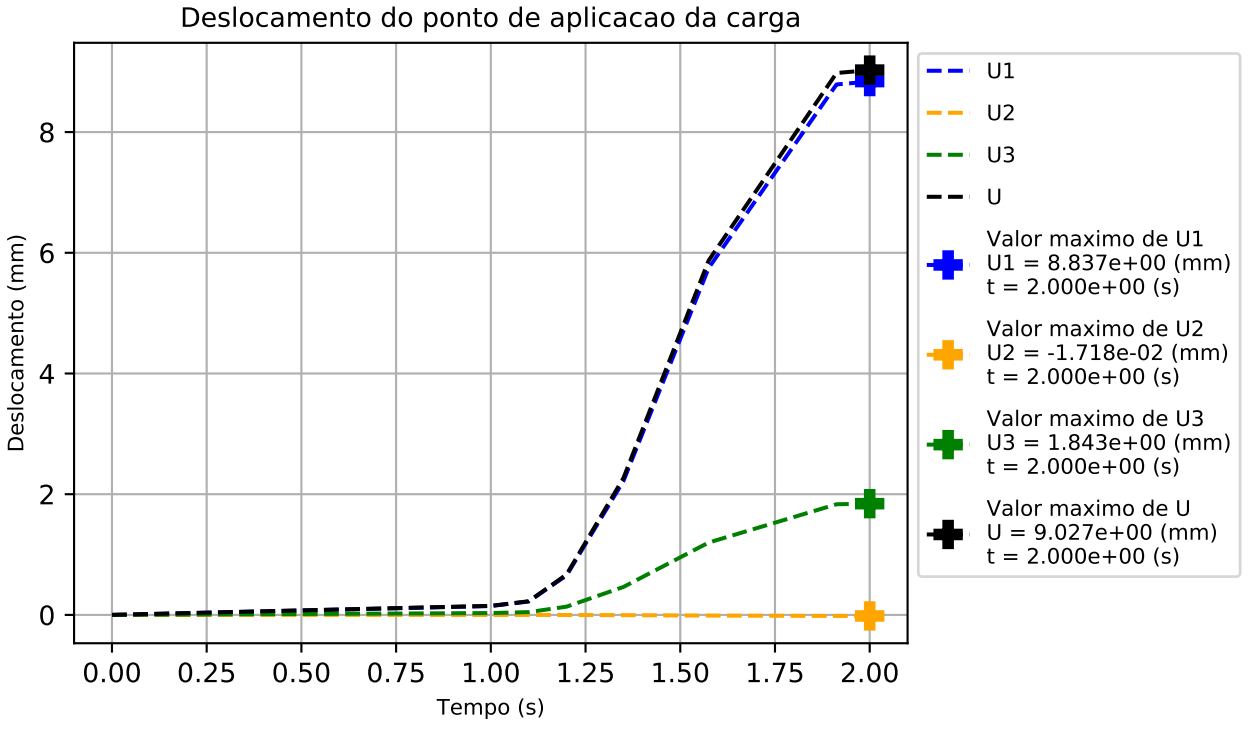

Tensao por elem.
Largura_20.0 Altura_2.0 Espessura_3.5
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

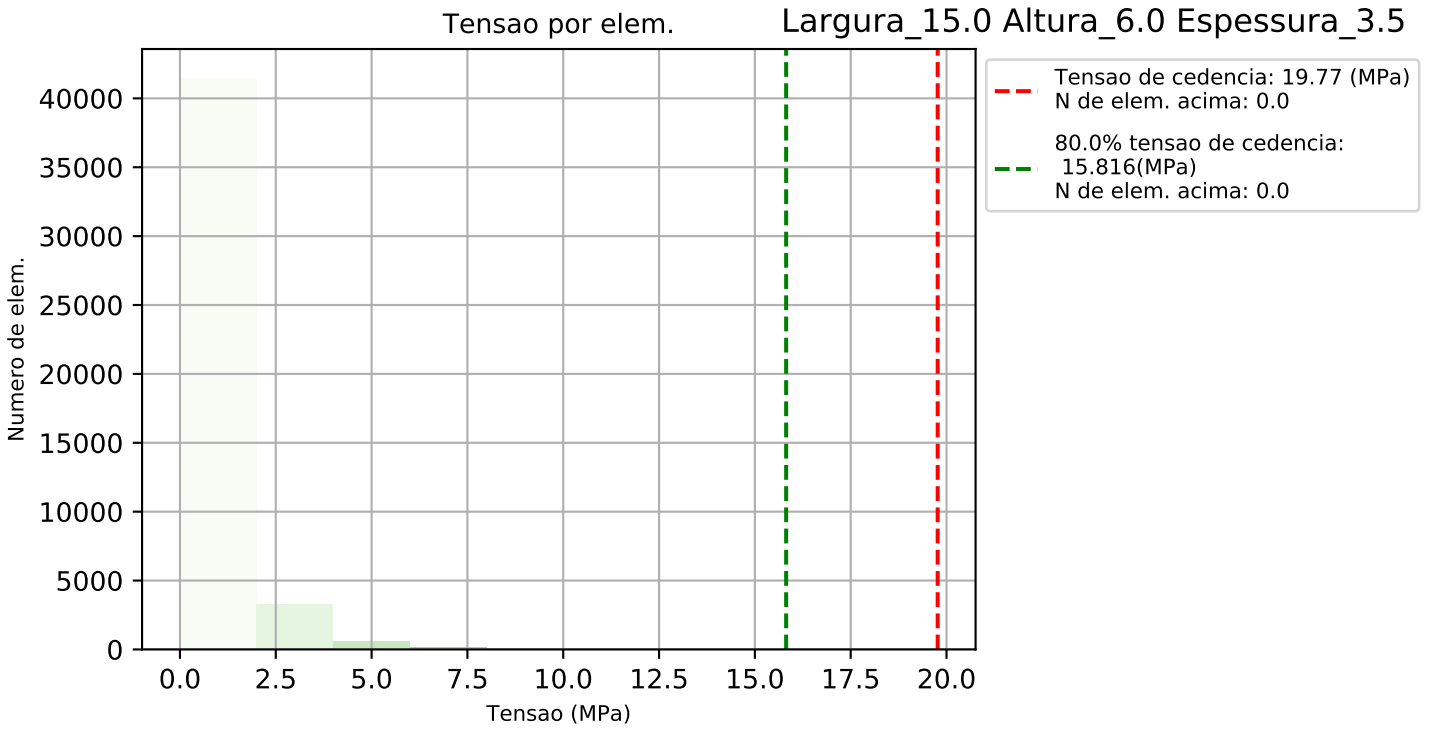

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

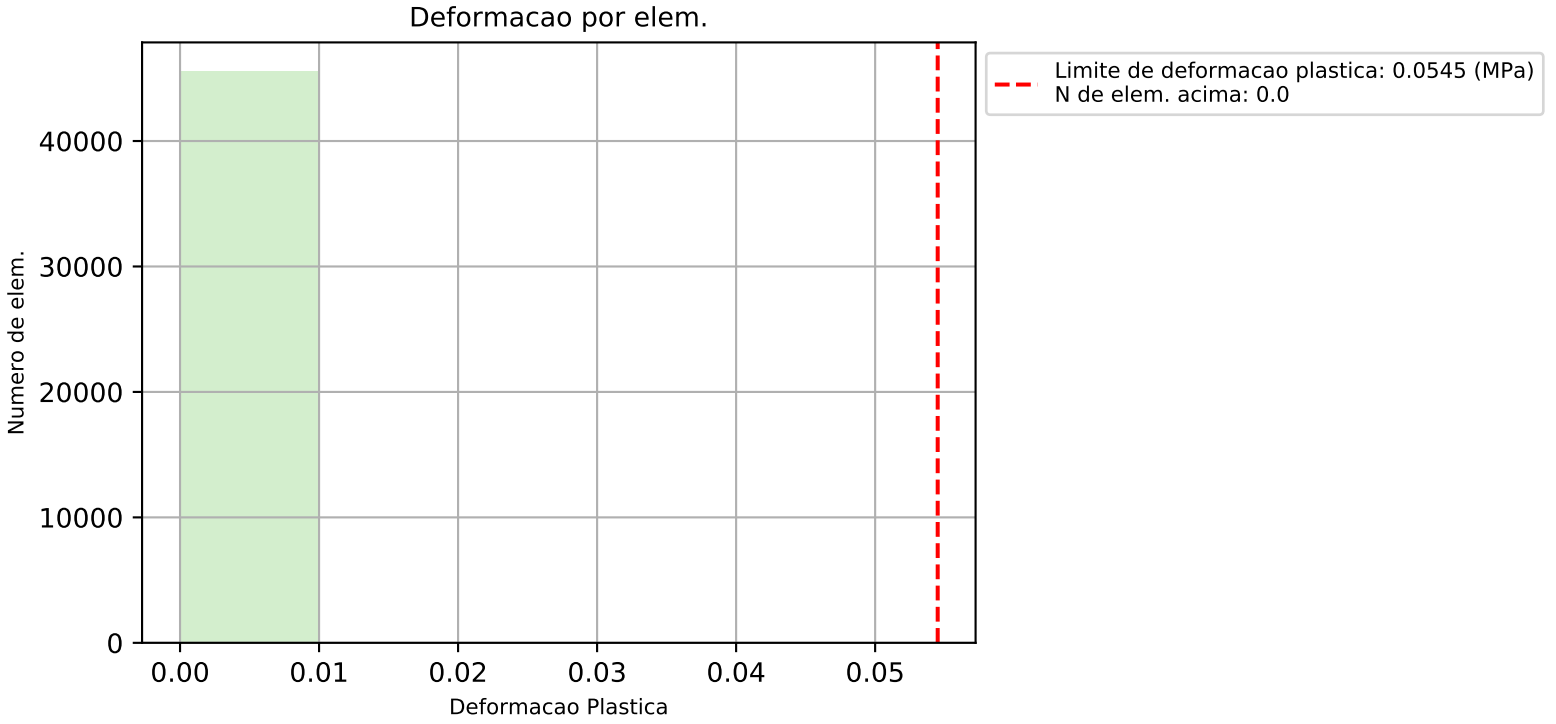

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.750e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.701e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.825e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.938e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

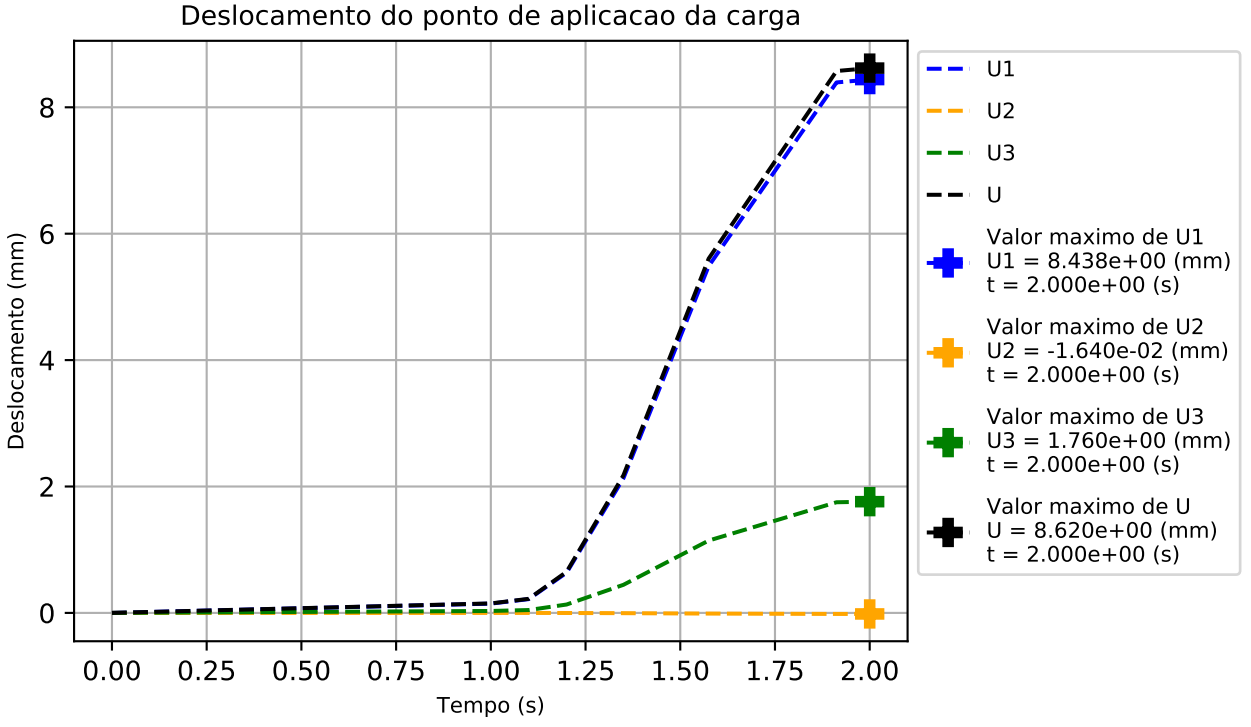

Tensao por elem.
Largura_20.0 Altura_2.0 Espessura_2.5
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

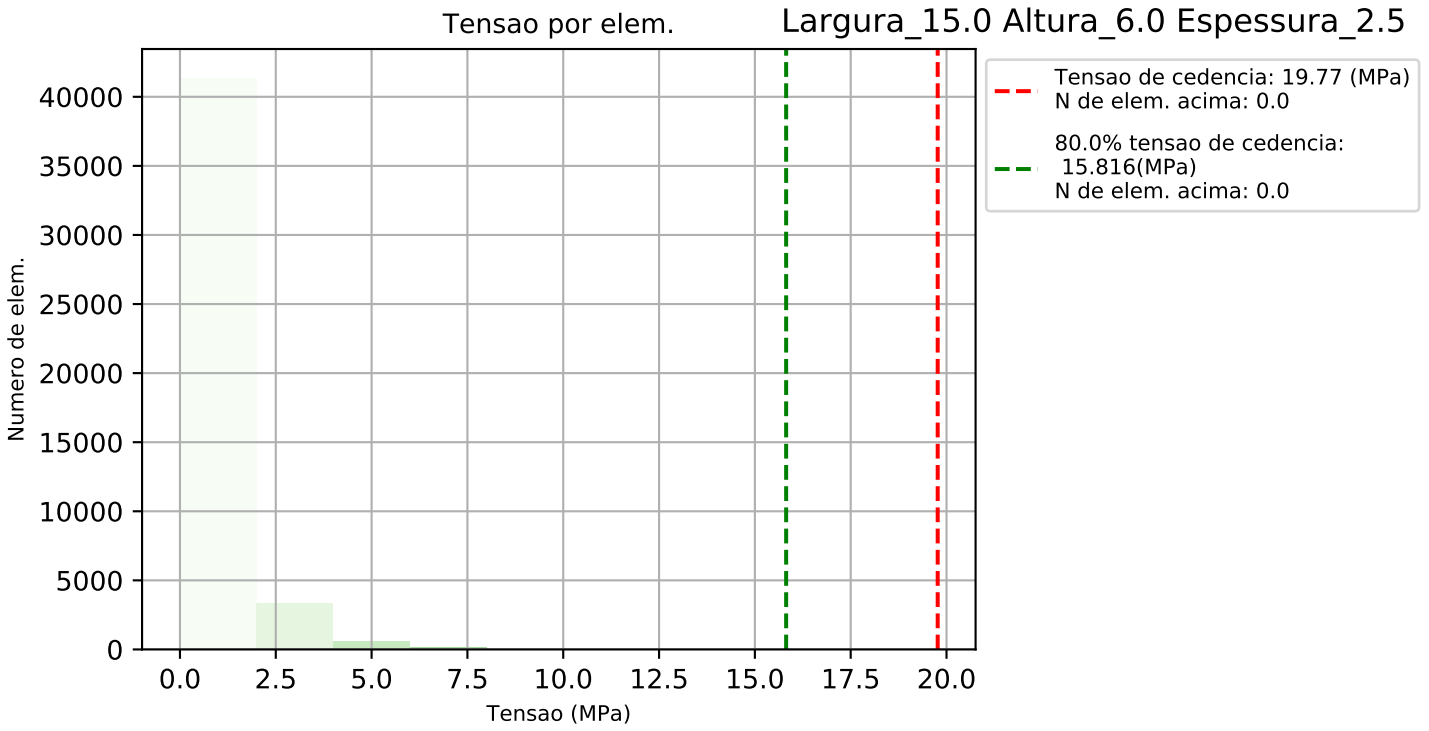

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

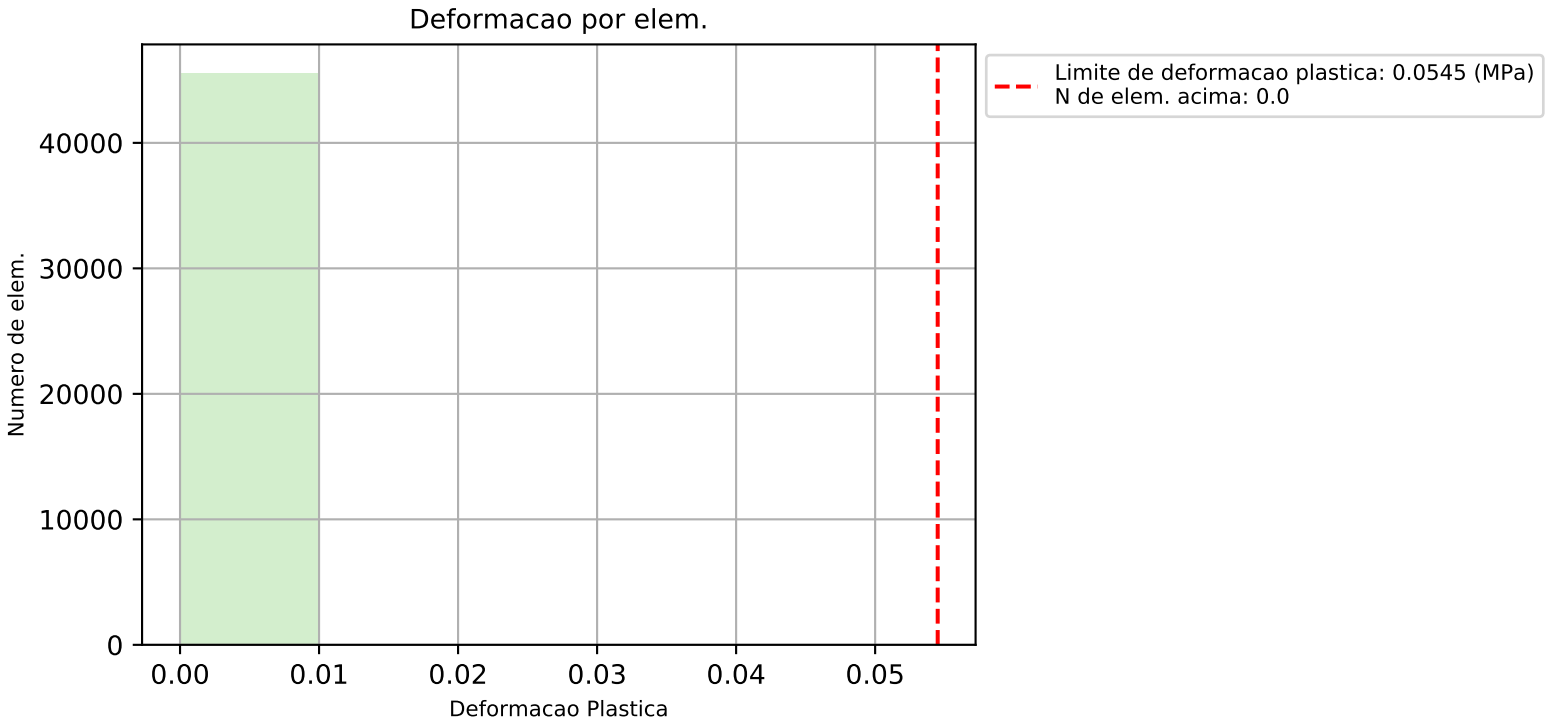

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.837e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.718e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.843e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 9.027e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

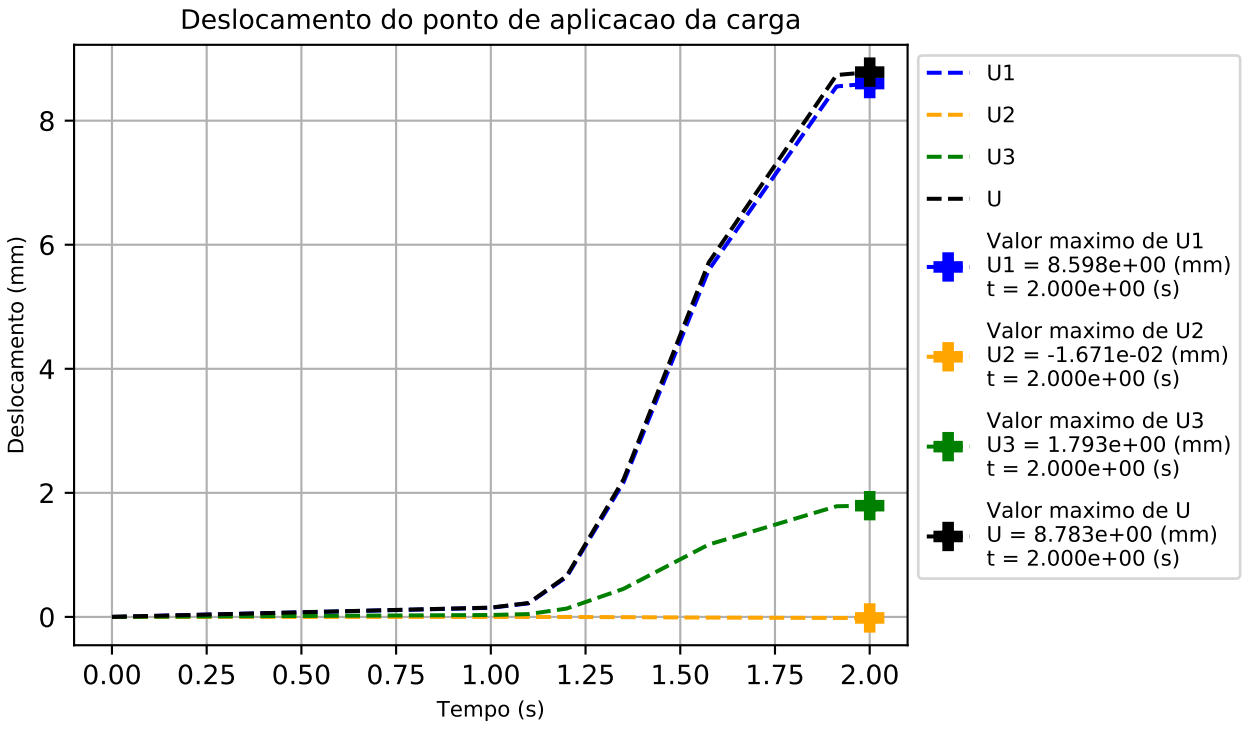

Tensao por elem.
Largura_15.0 Altura_6.0 Espessura_3.5
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

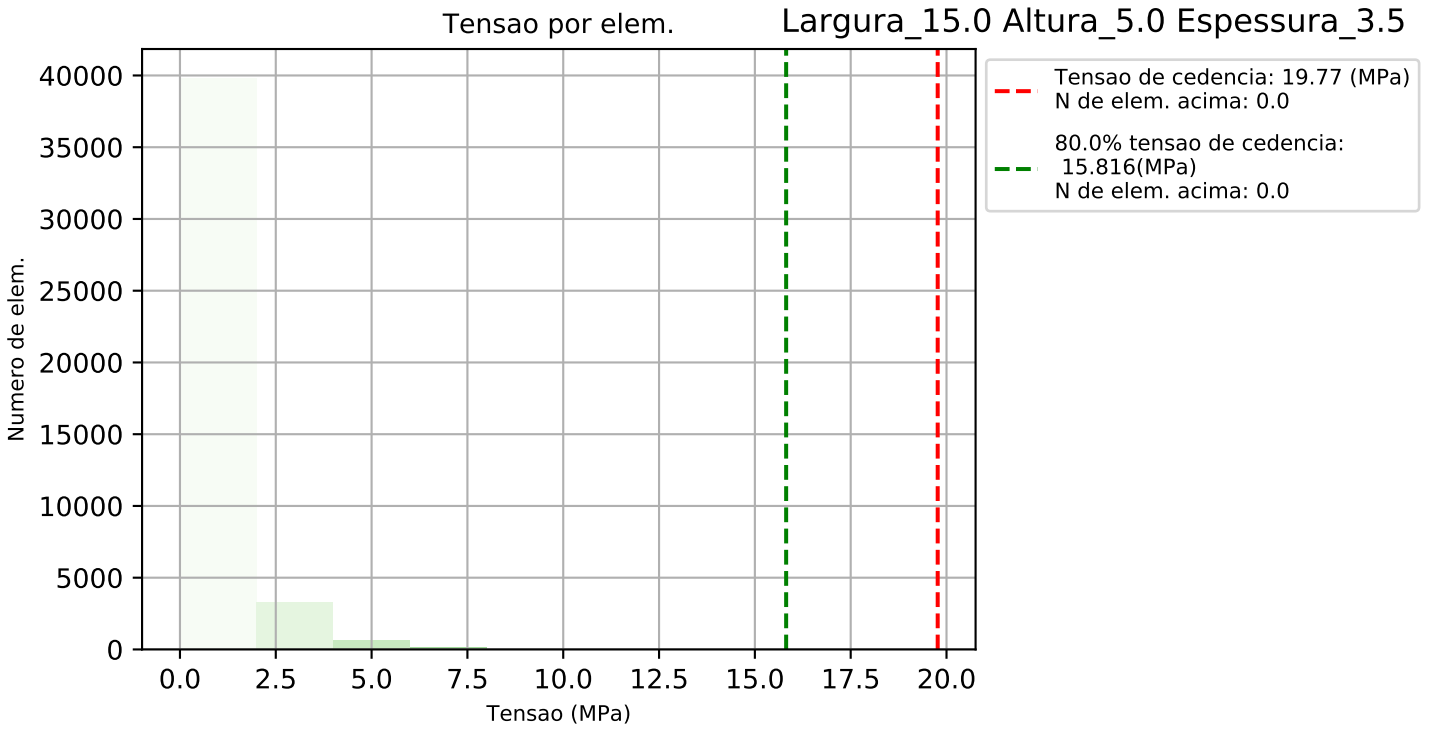

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

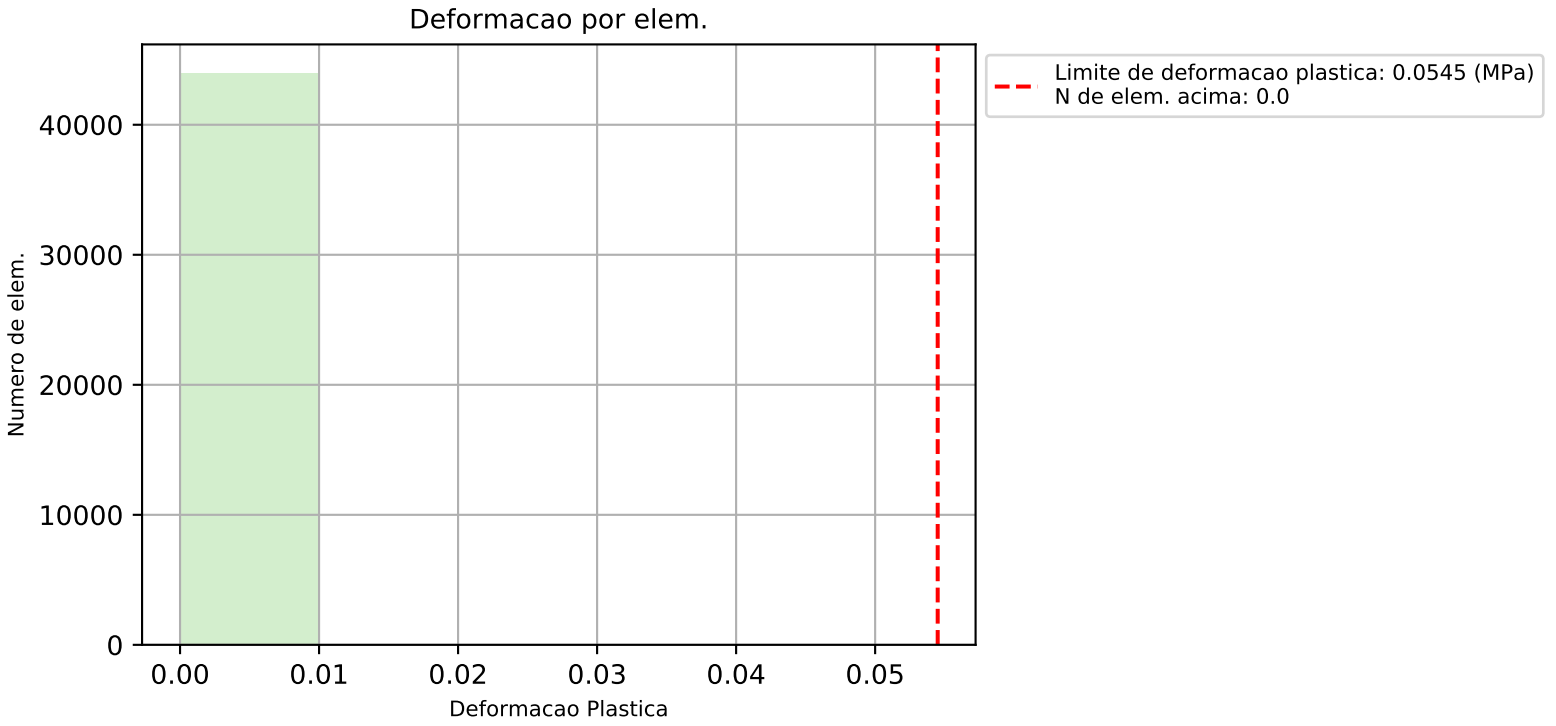

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.438e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.640e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.760e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.620e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

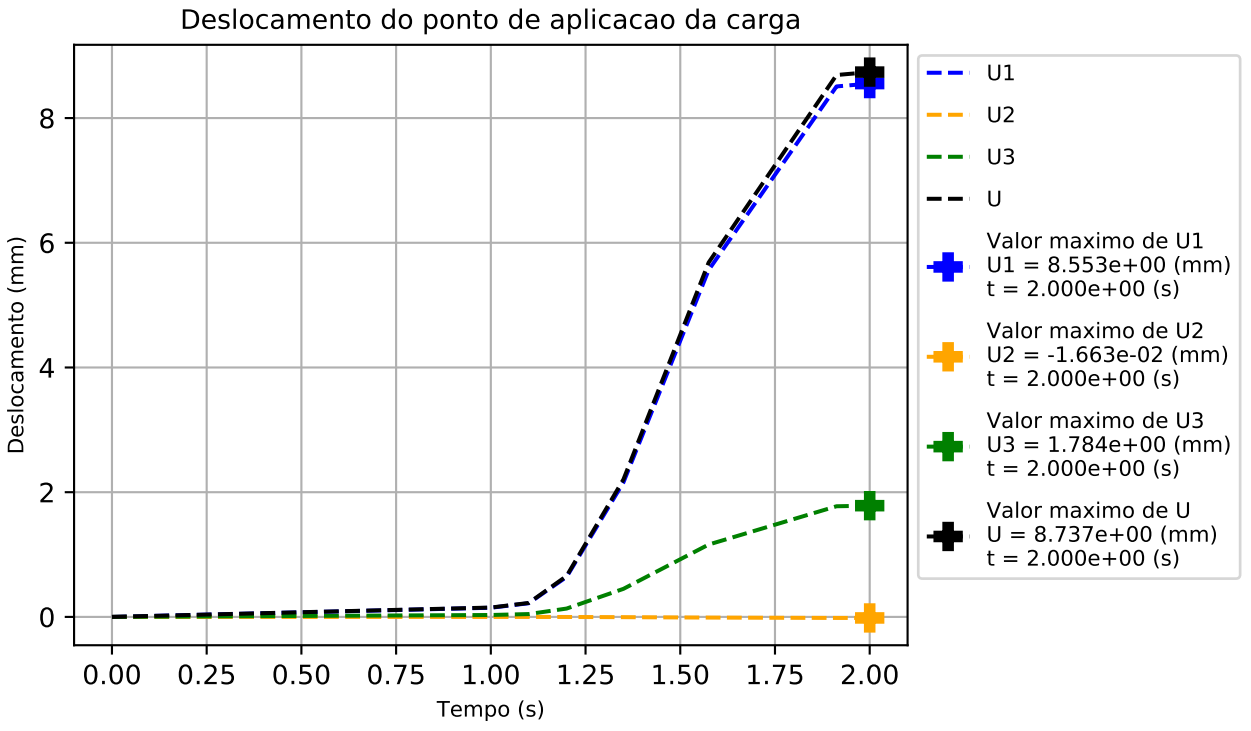

Tensao por elem.
Largura_15.0 Altura_6.0 Espessura_2.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

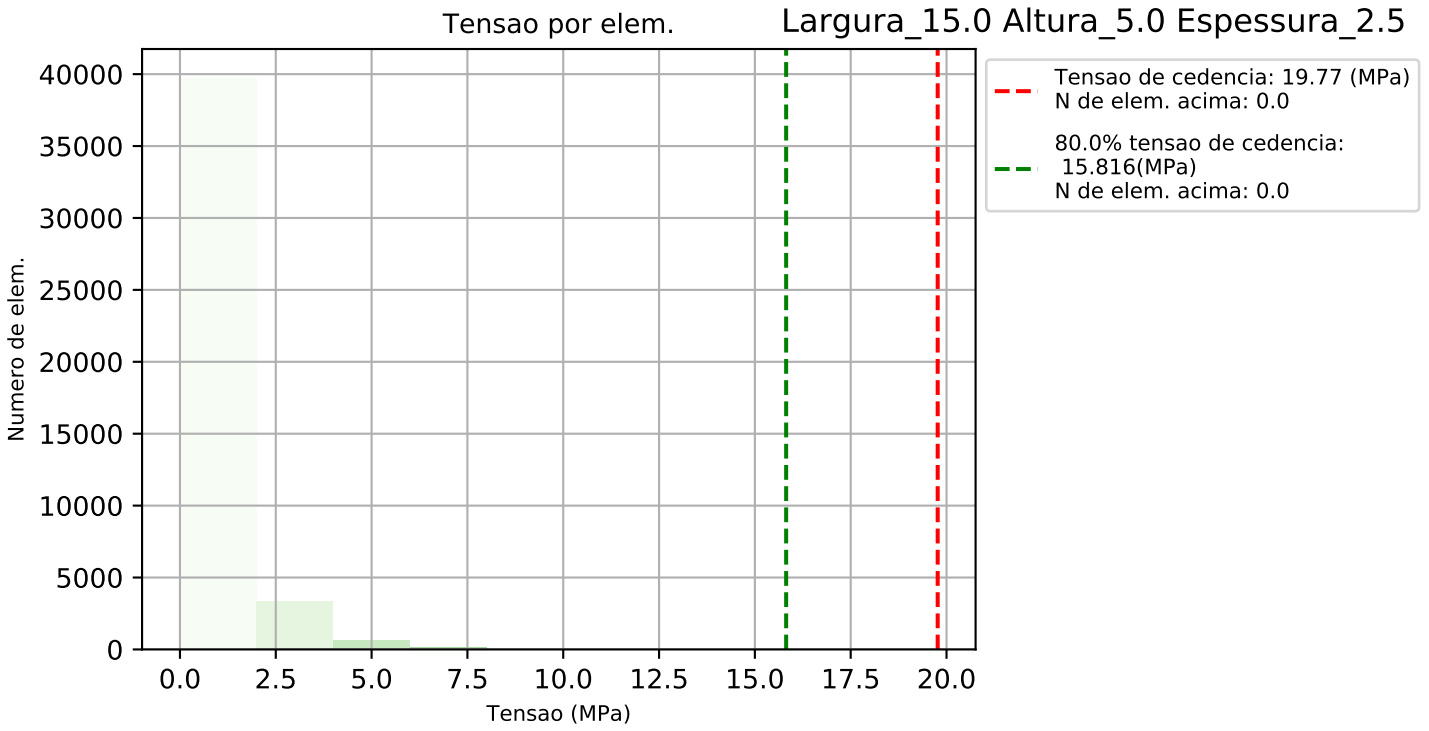

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

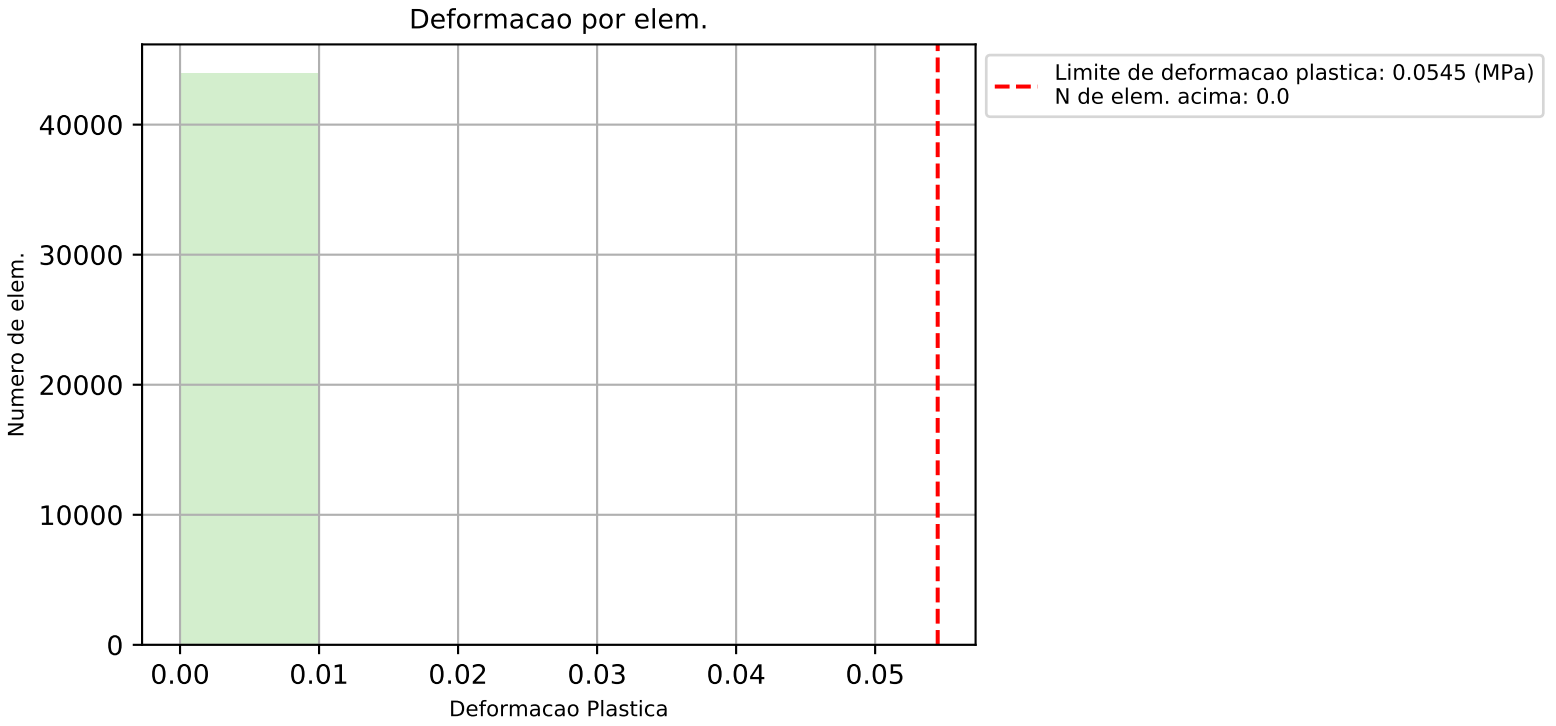

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.598e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.671e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.793e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.783e+00 (mm)
t = 2.000e+00 (s)

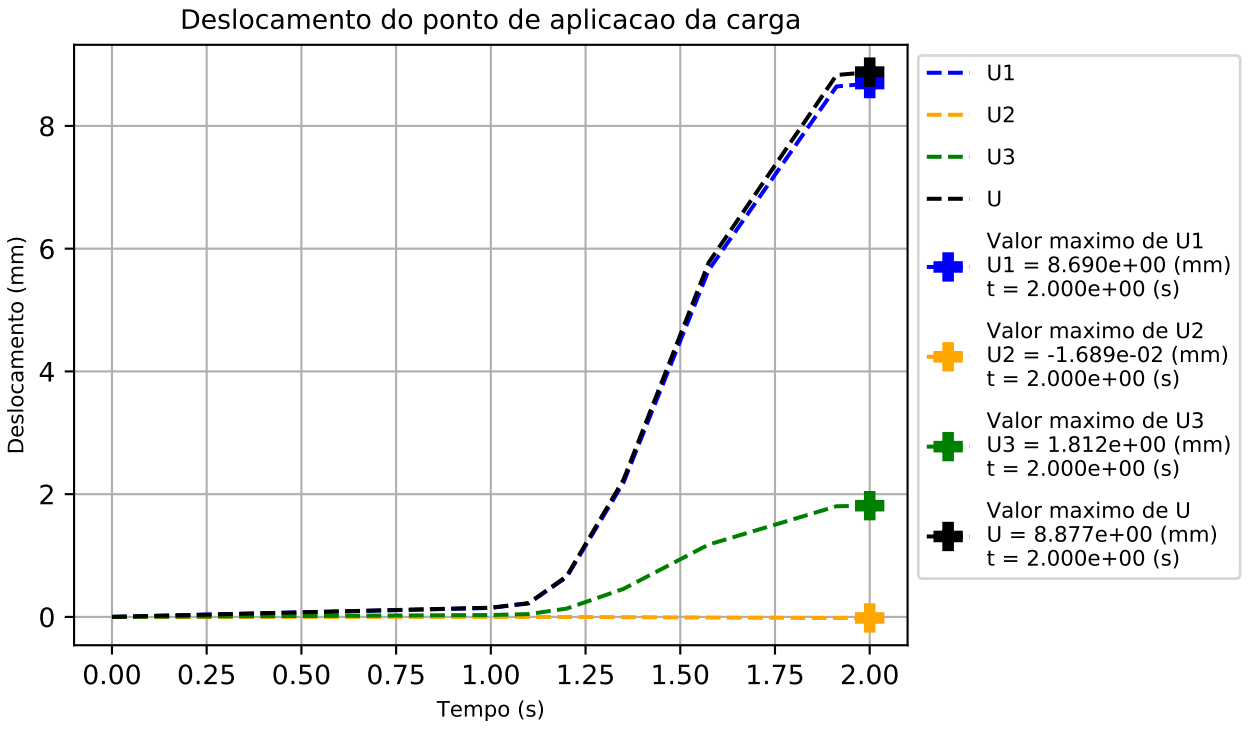

Tensao por elem.
Largura_15.0 Altura_5.0 Espessura_3.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

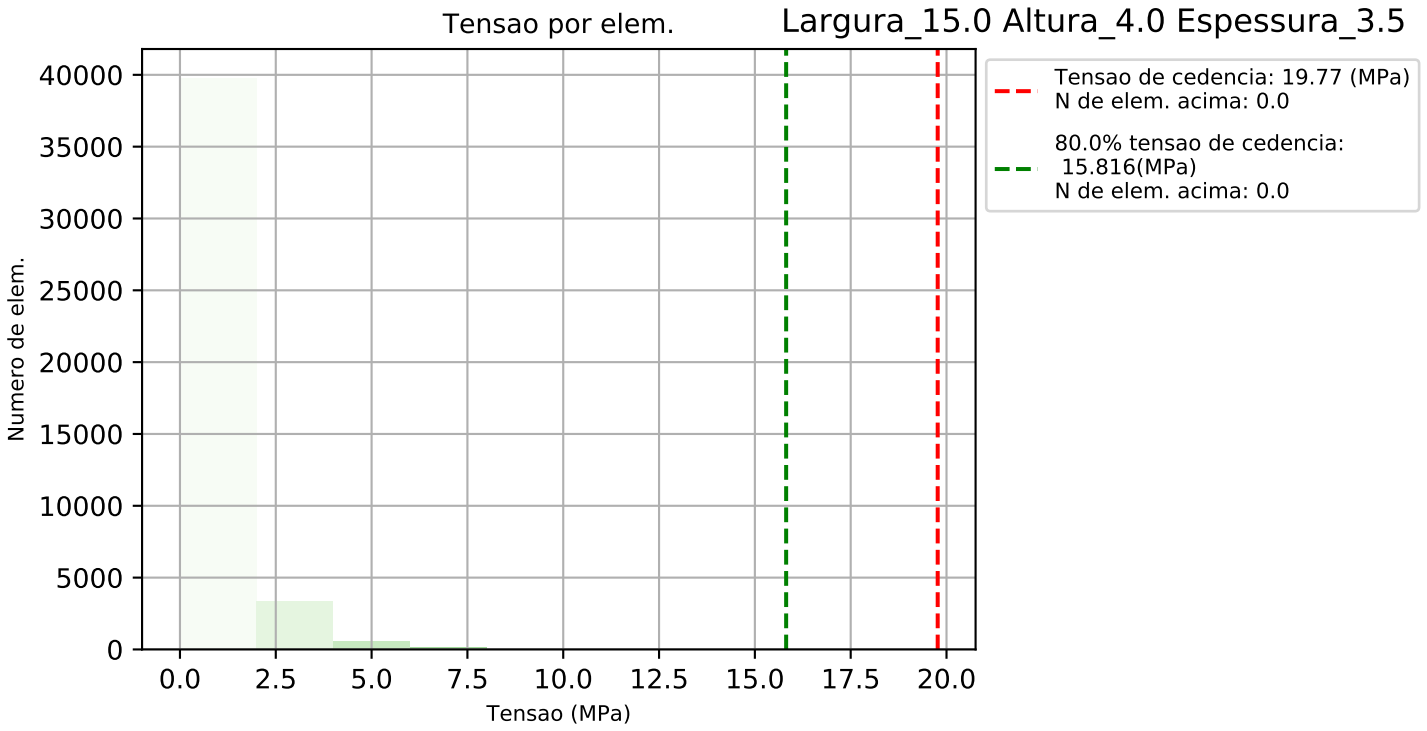

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

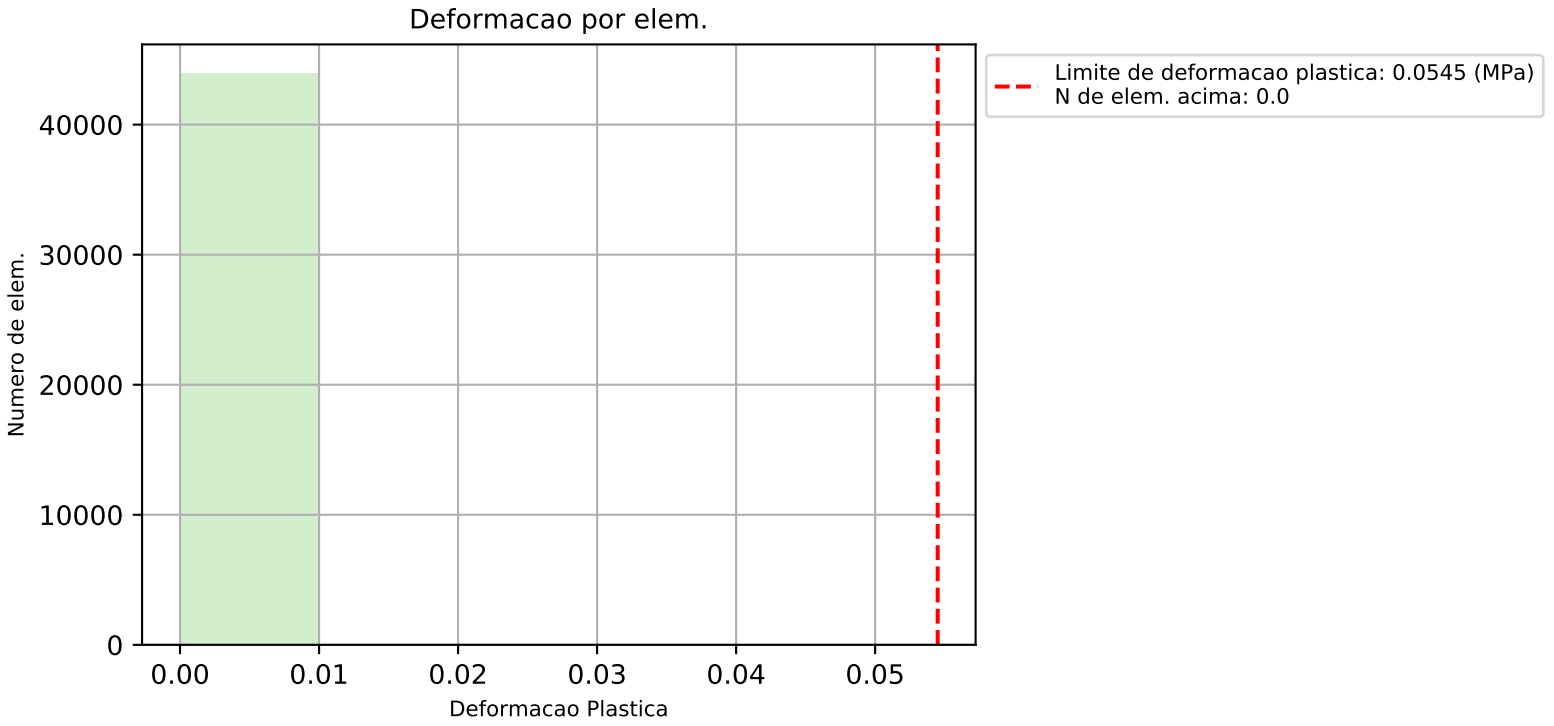

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.553e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.663e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.784e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.737e+00 (mm)
t = 2.000e+00 (s)

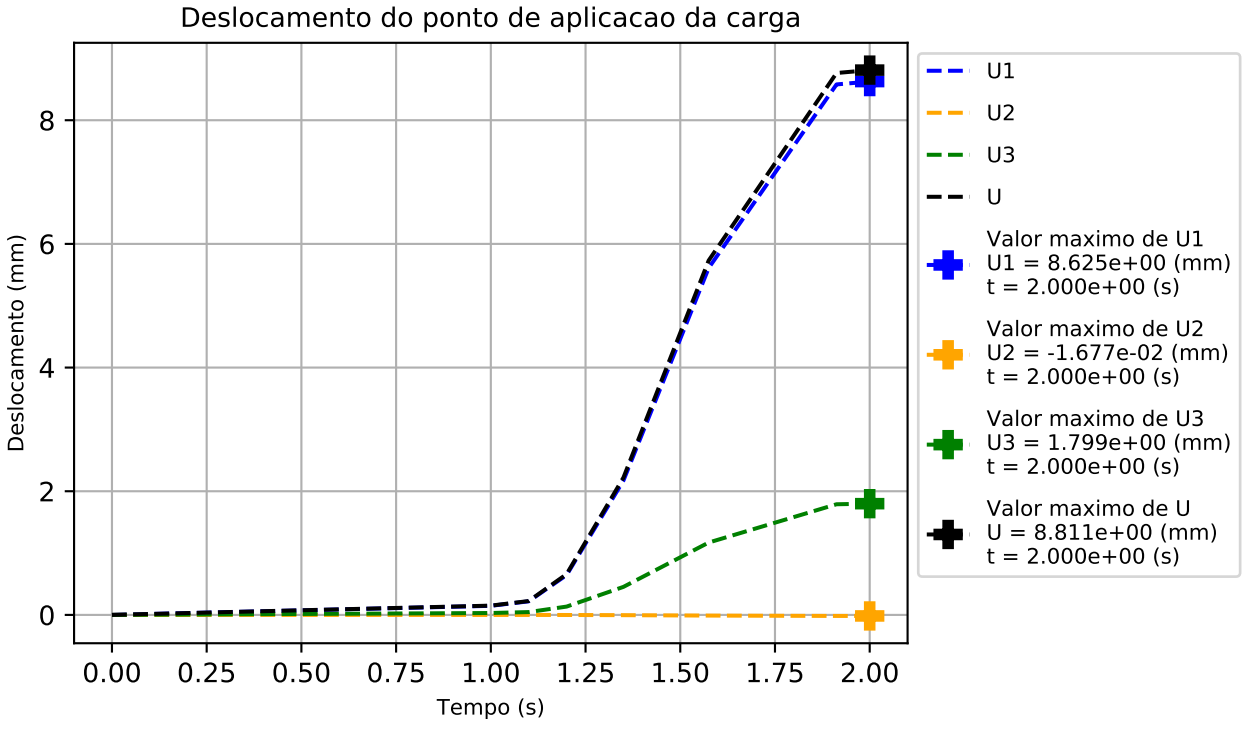

Tensao por elem.
Largura_15.0 Altura_5.0 Espessura_2.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

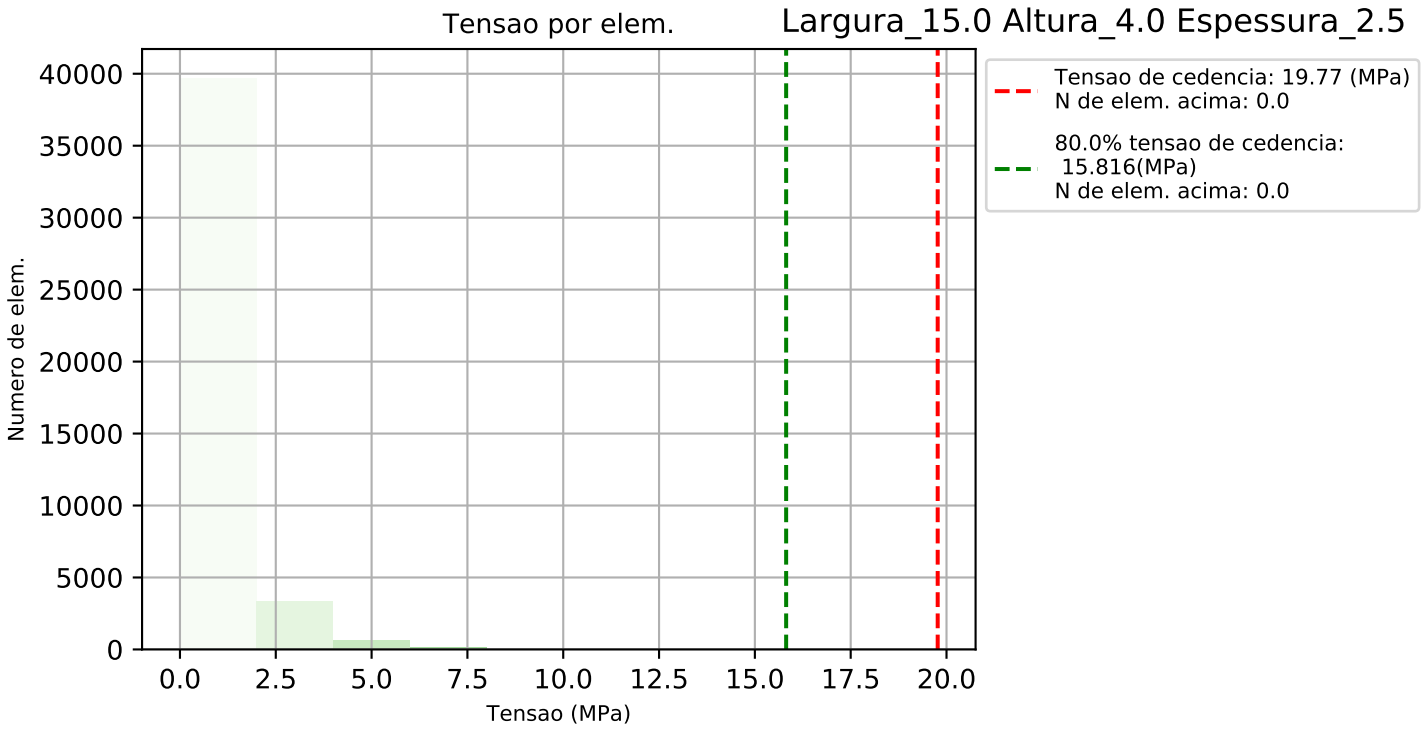

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

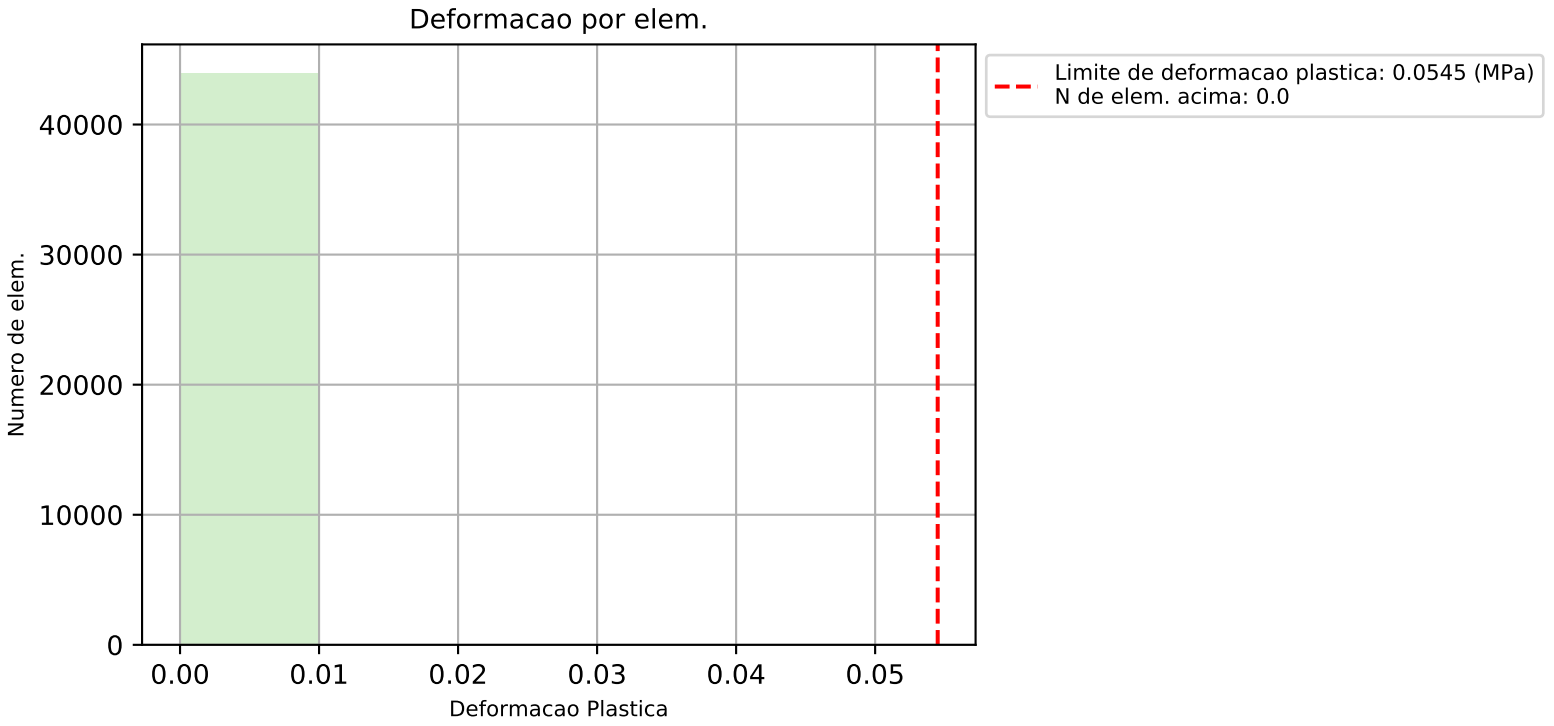

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.690e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.689e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.812e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.877e+00 (mm)
t = 2.000e+00 (s)

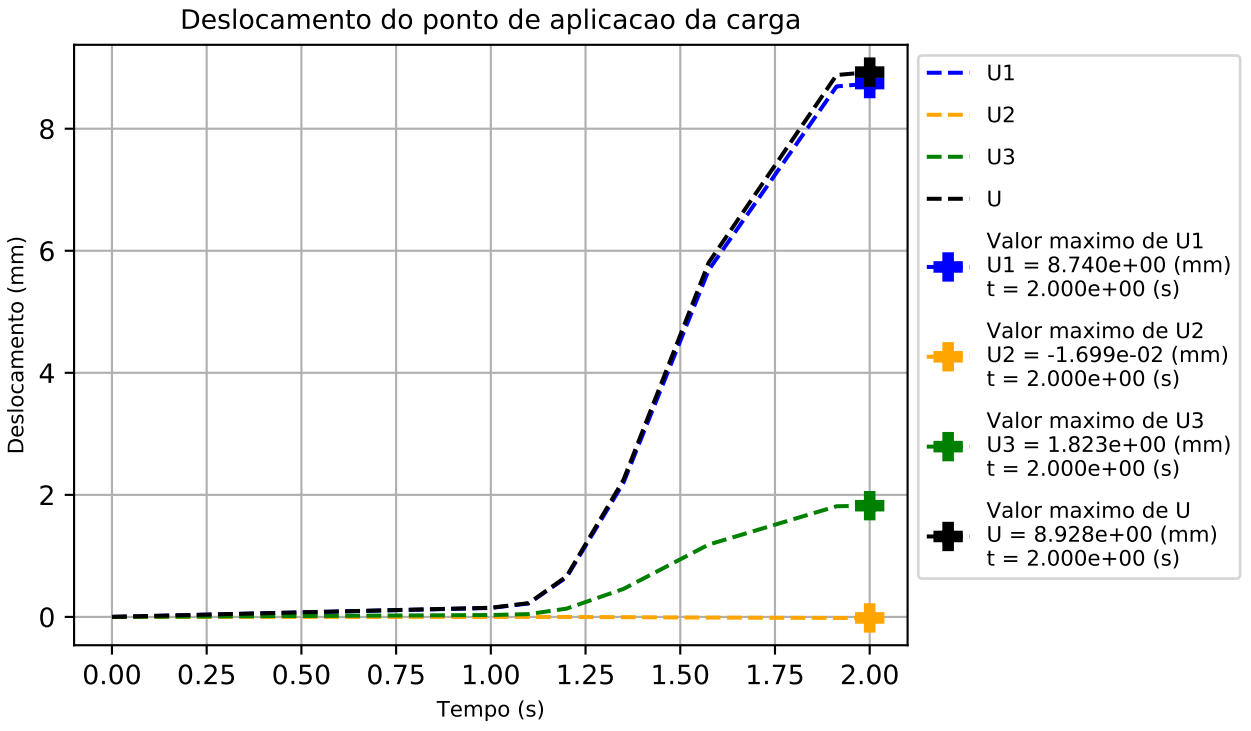

Tensao por elem.
Largura_15.0 Altura_4.0 Espessura_3.5
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

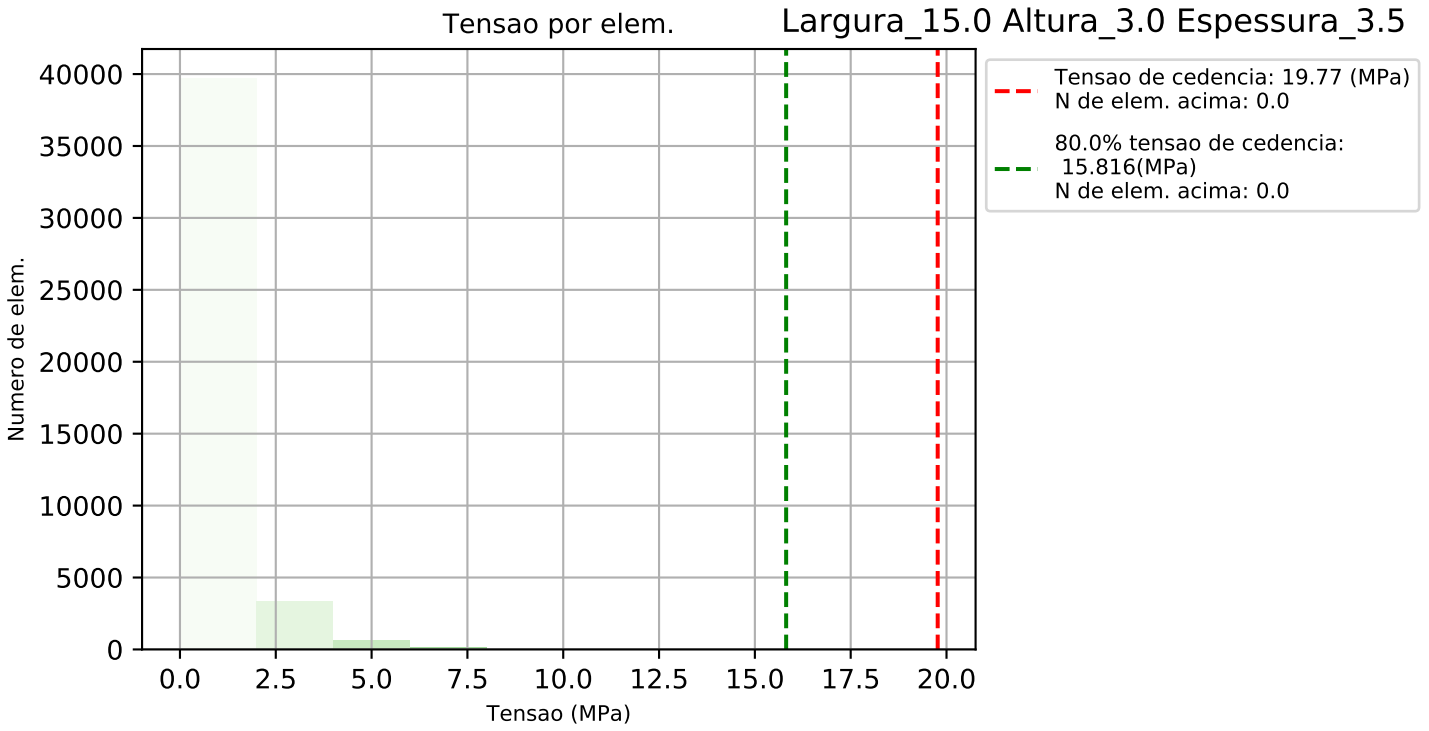

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

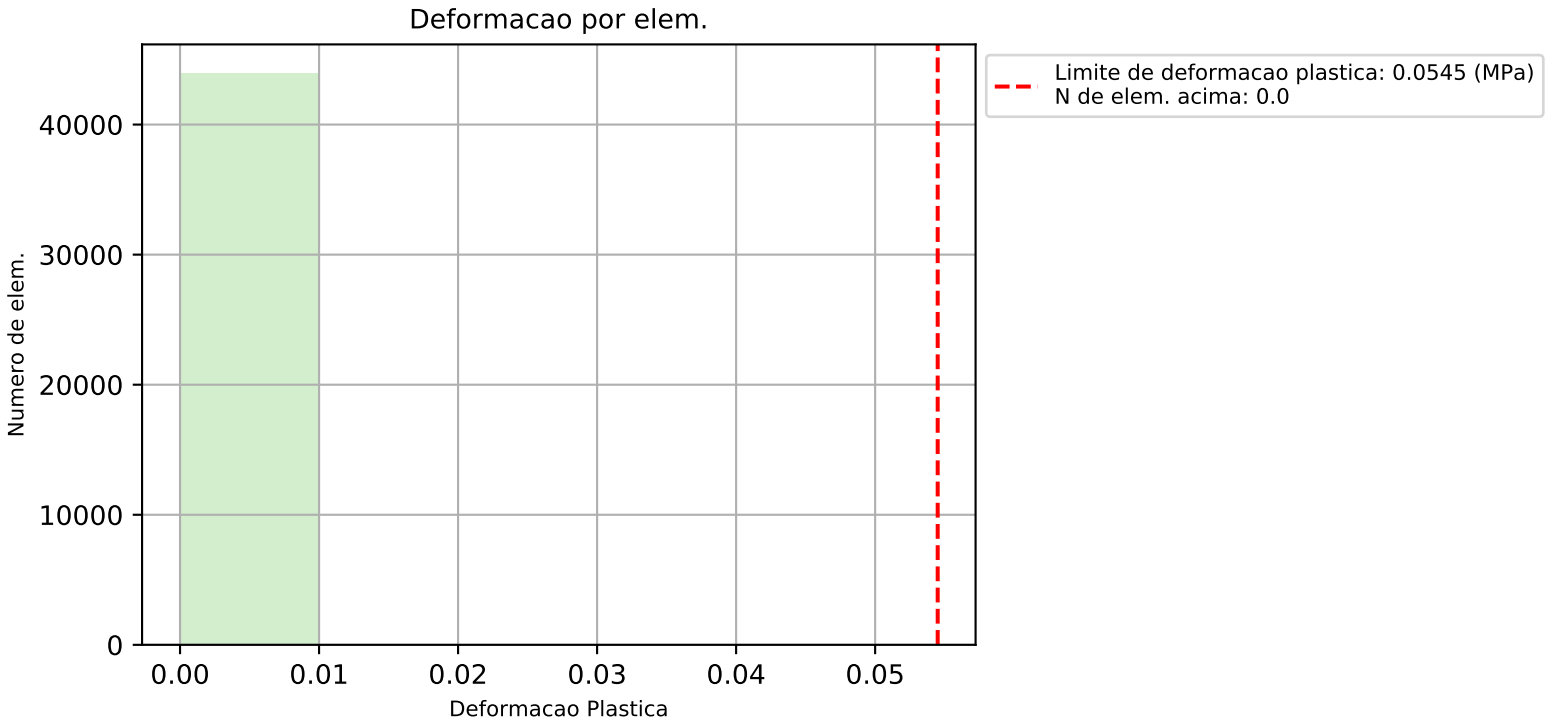

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.625e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.677e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.799e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.811e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

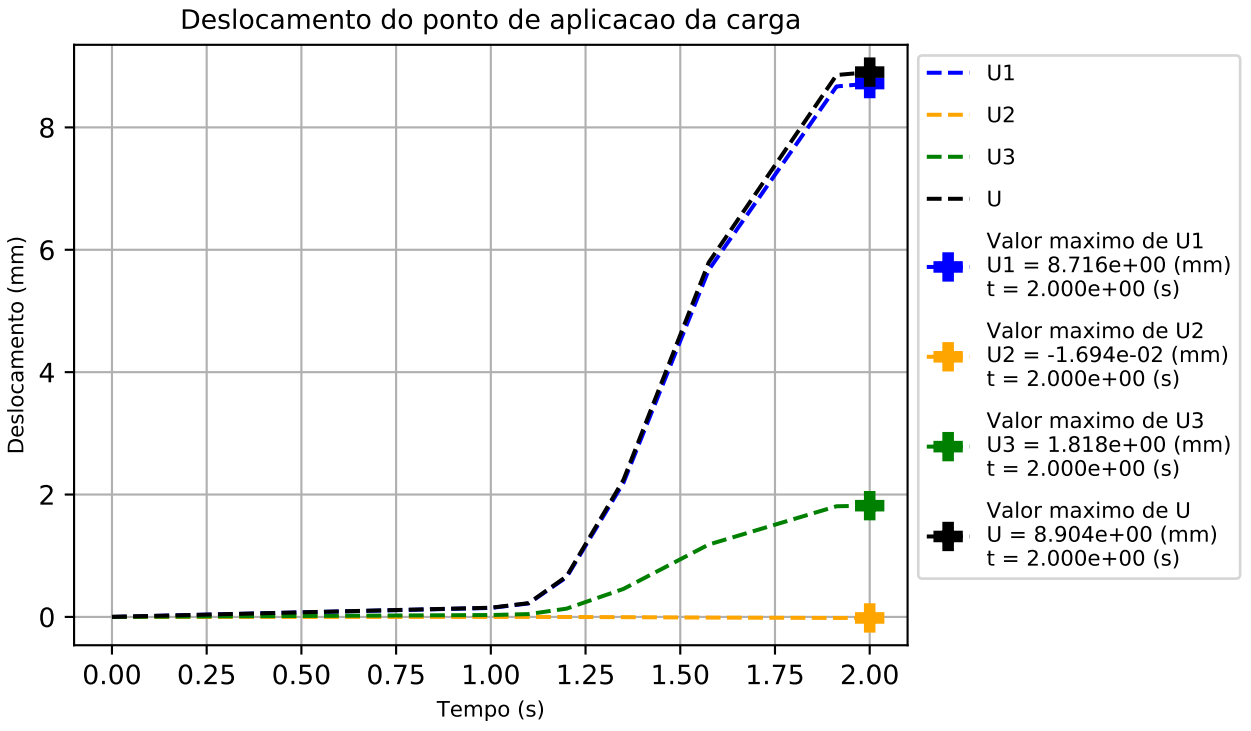

Tensao por elem.
Largura_15.0 Altura_4.0 Espessura_2.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

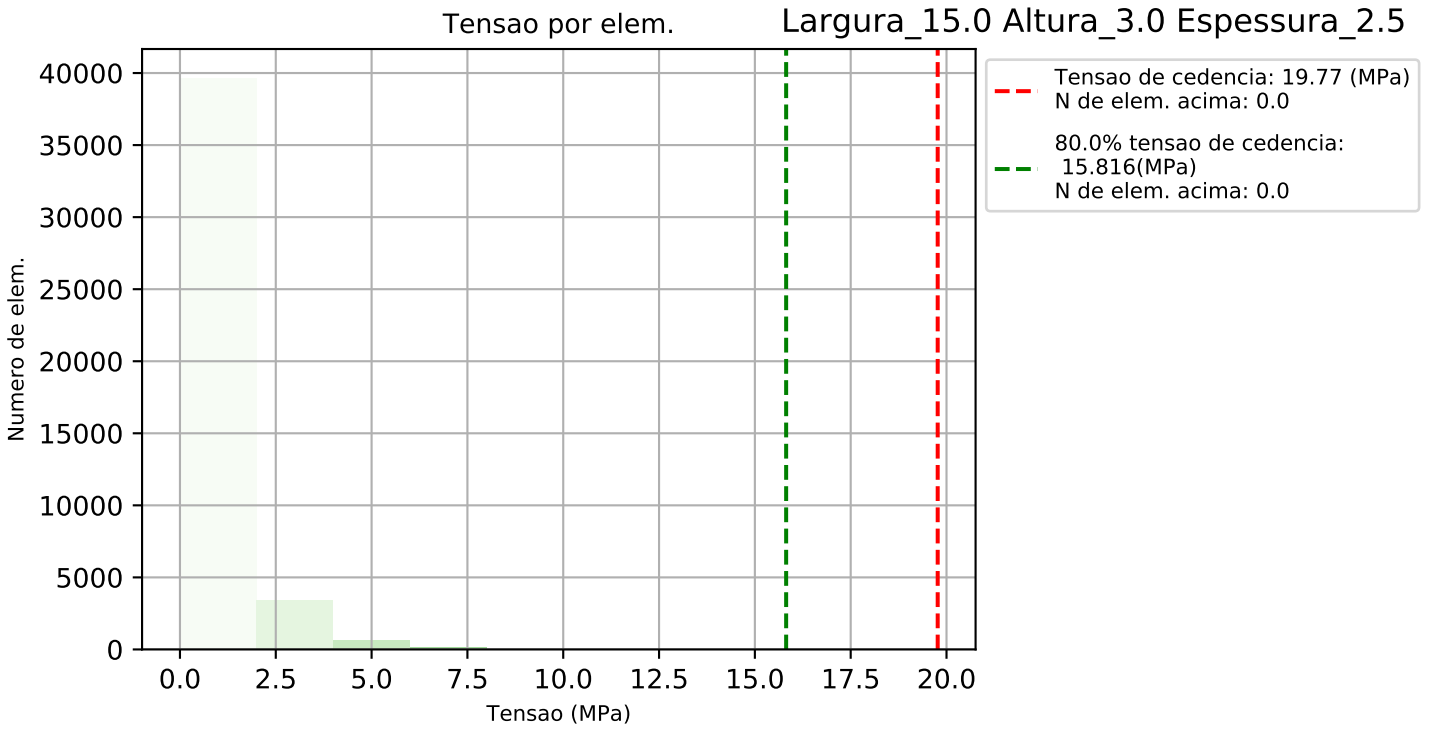

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

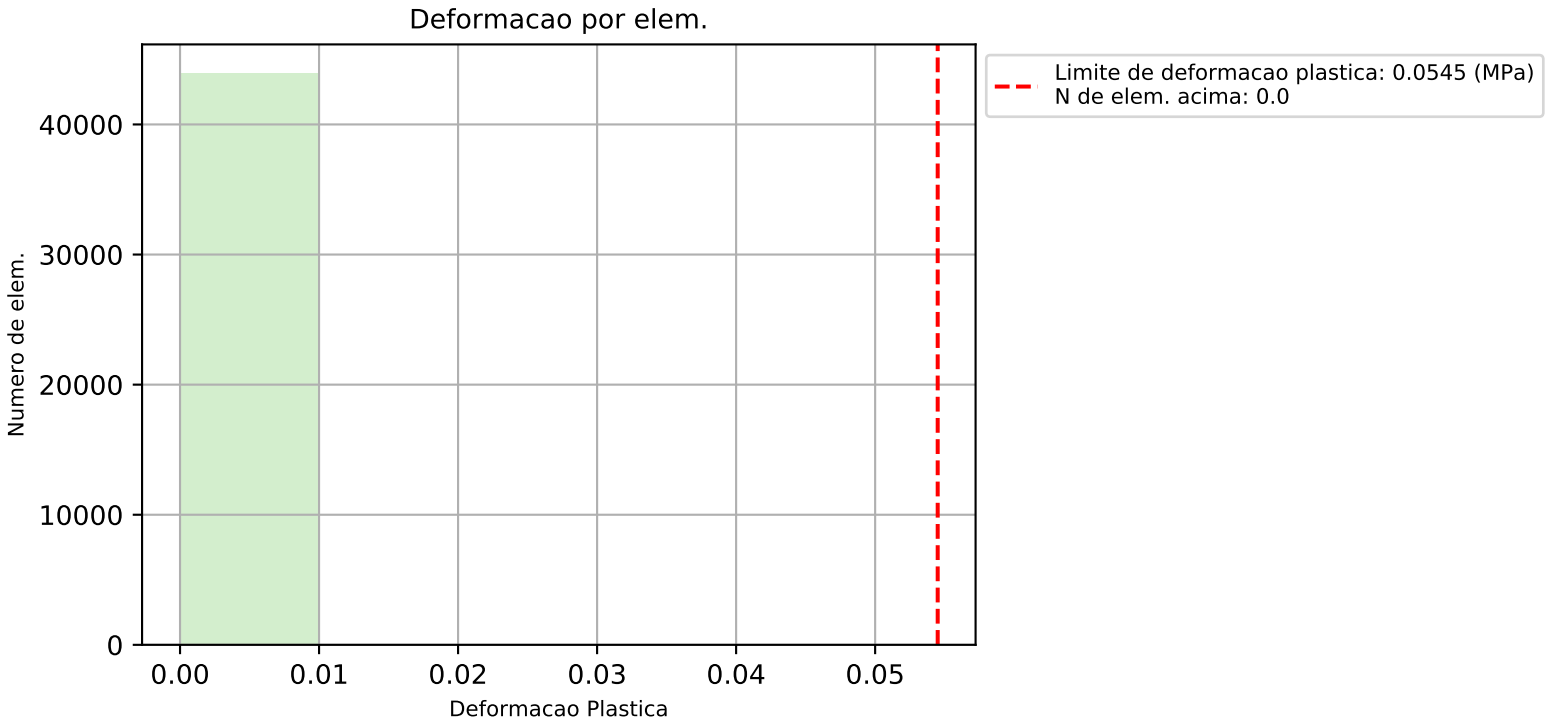

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.740e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.699e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.823e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.928e+00 (mm)
t = 2.000e+00 (s)

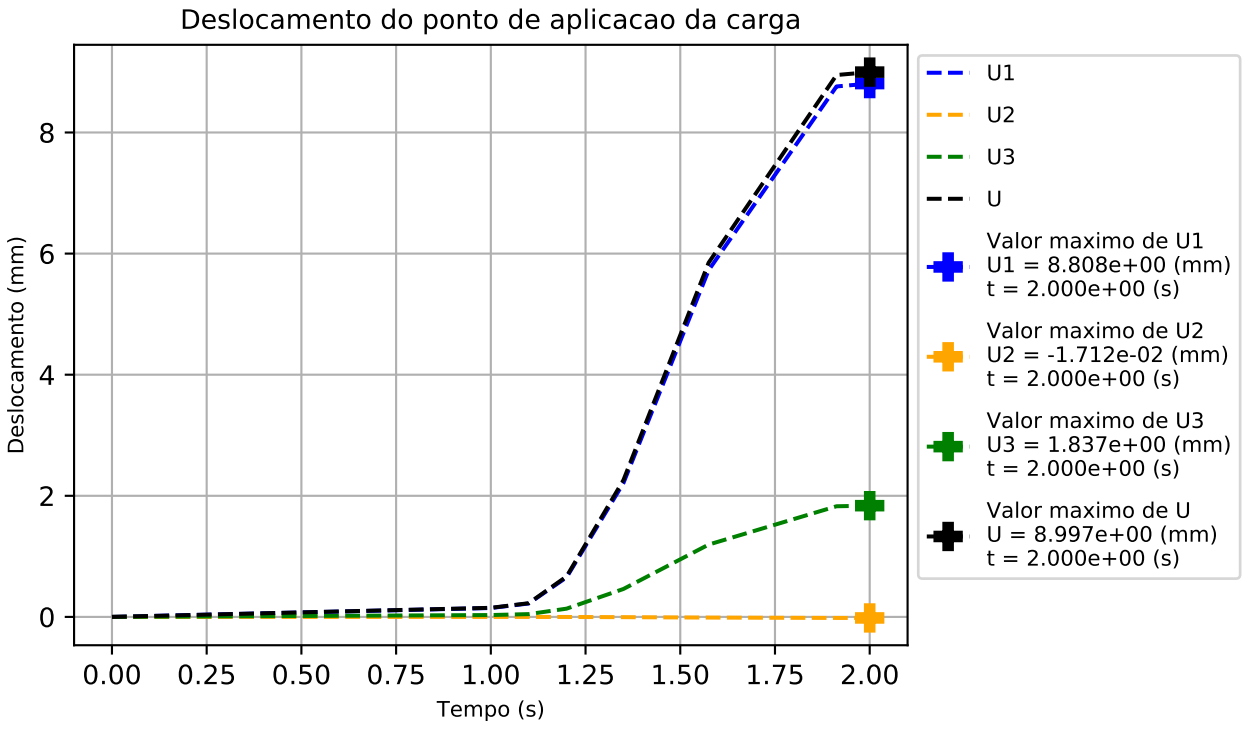

Tensao por elem.
Largura_15.0 Altura_3.0 Espessura_3.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

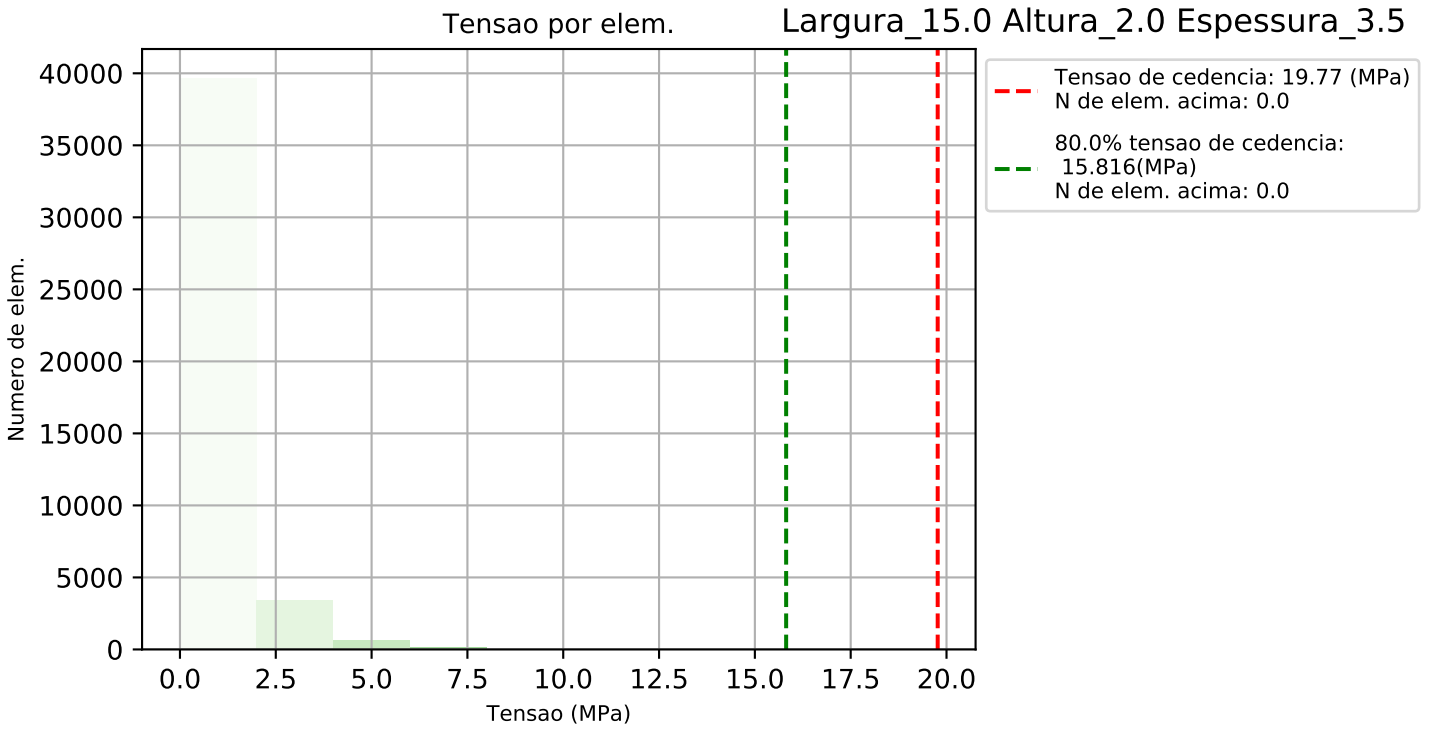

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

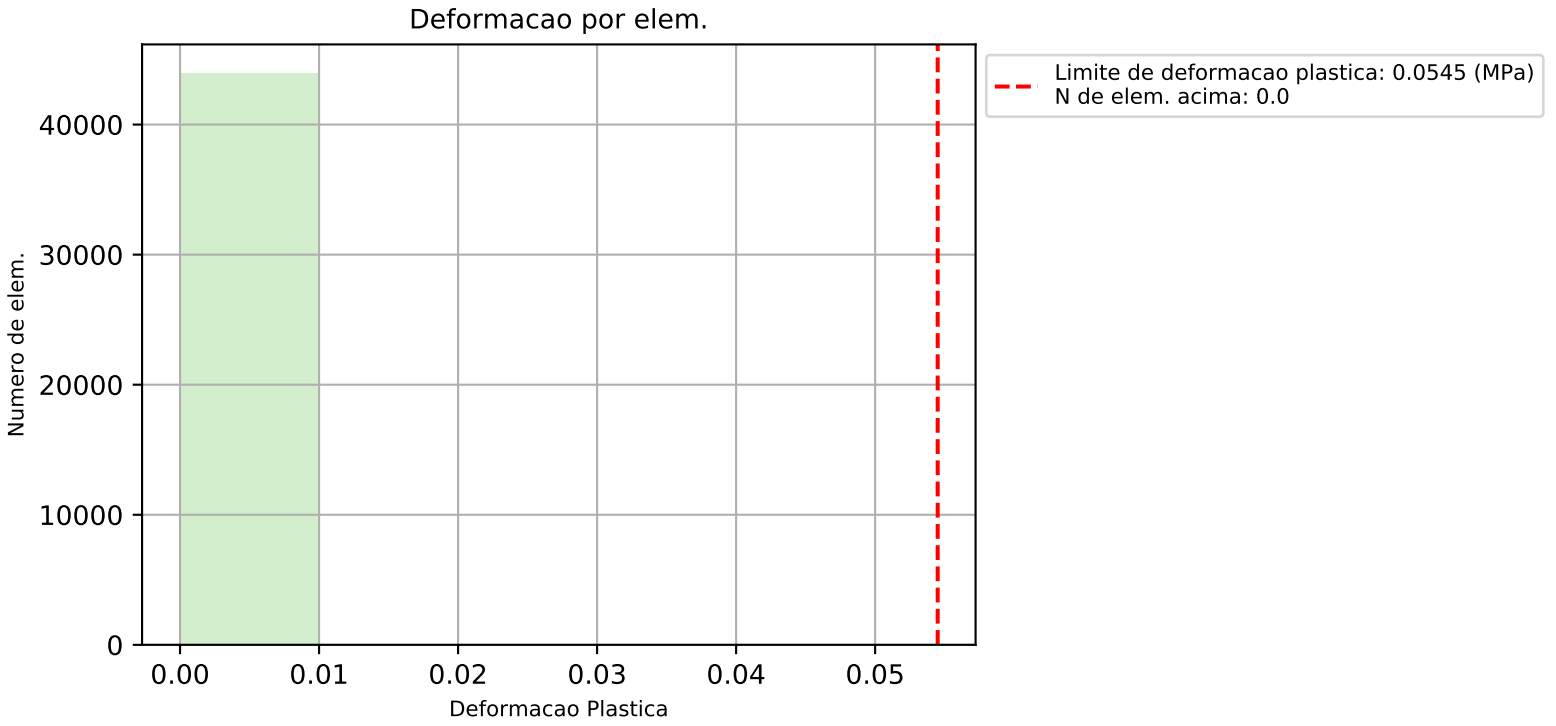

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.716e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.694e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.818e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.904e+00 (mm)
t = 2.000e+00 (s)

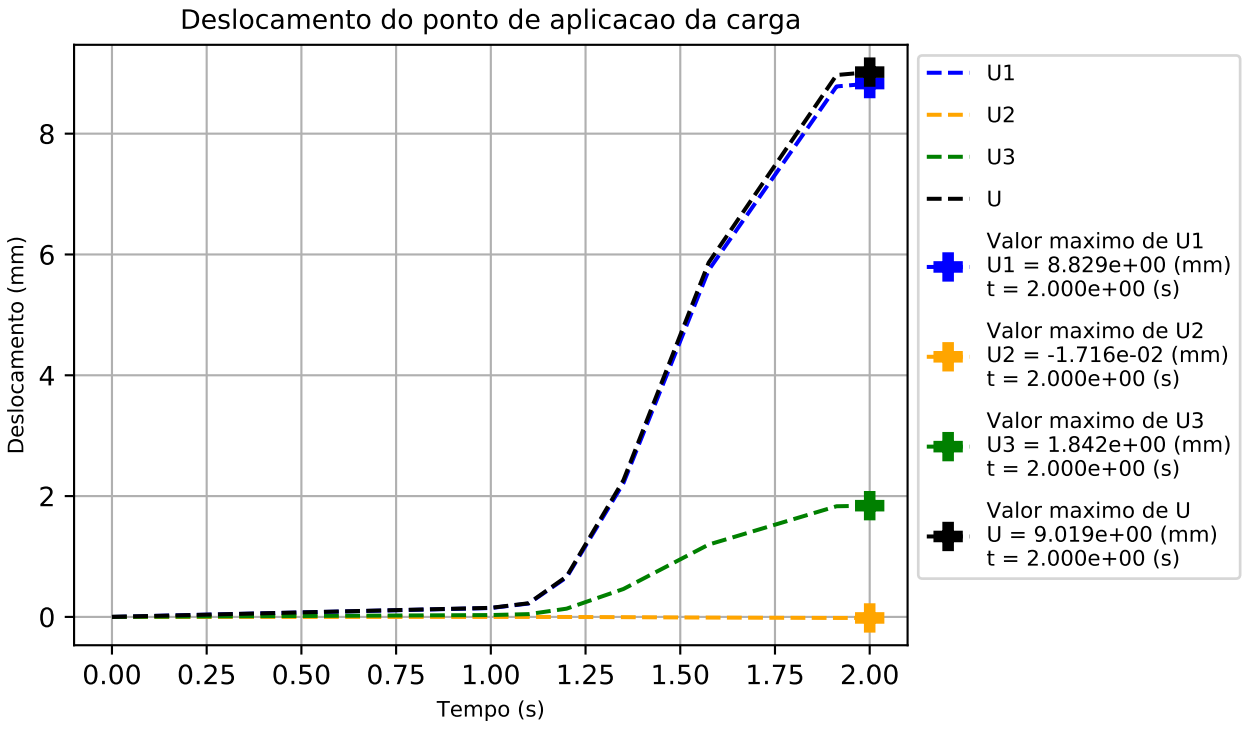

Tensao por elem.
Largura_15.0 Altura_3.0 Espessura_2.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

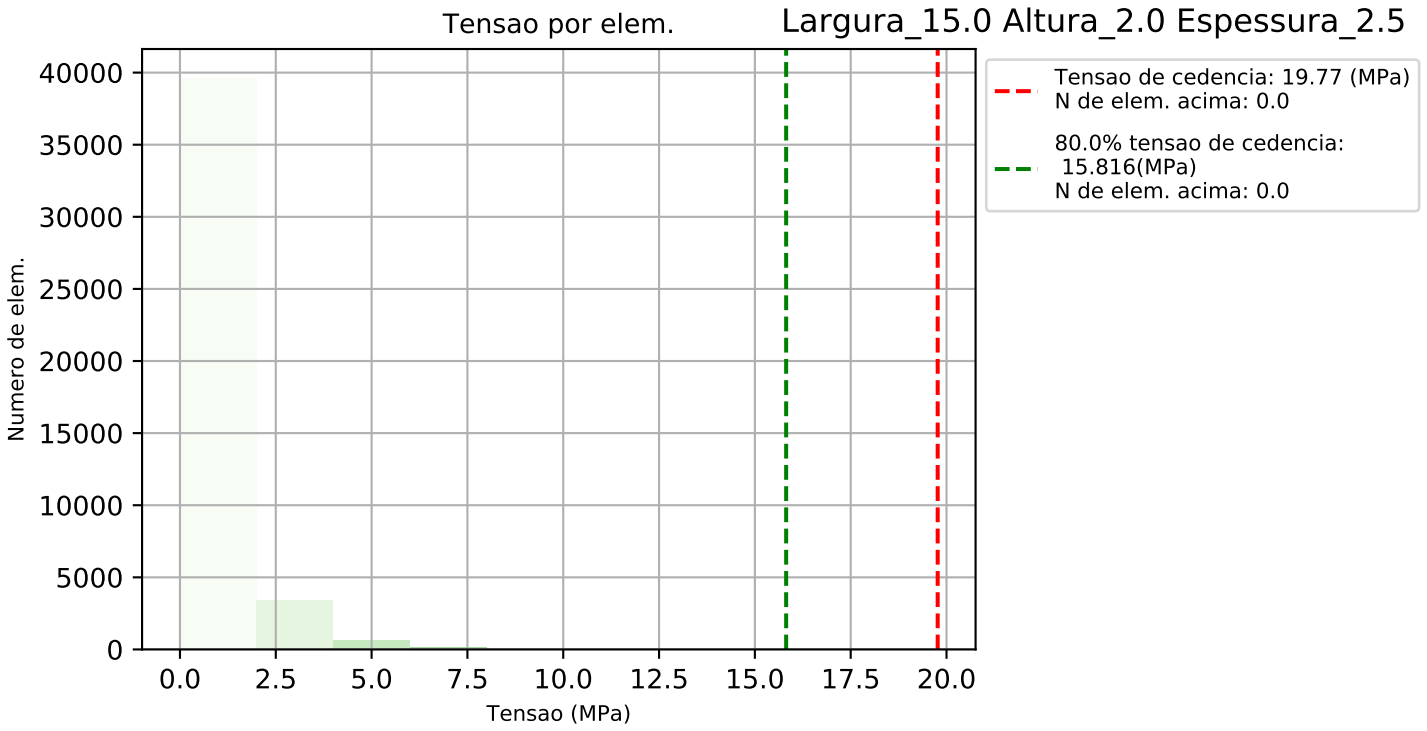

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

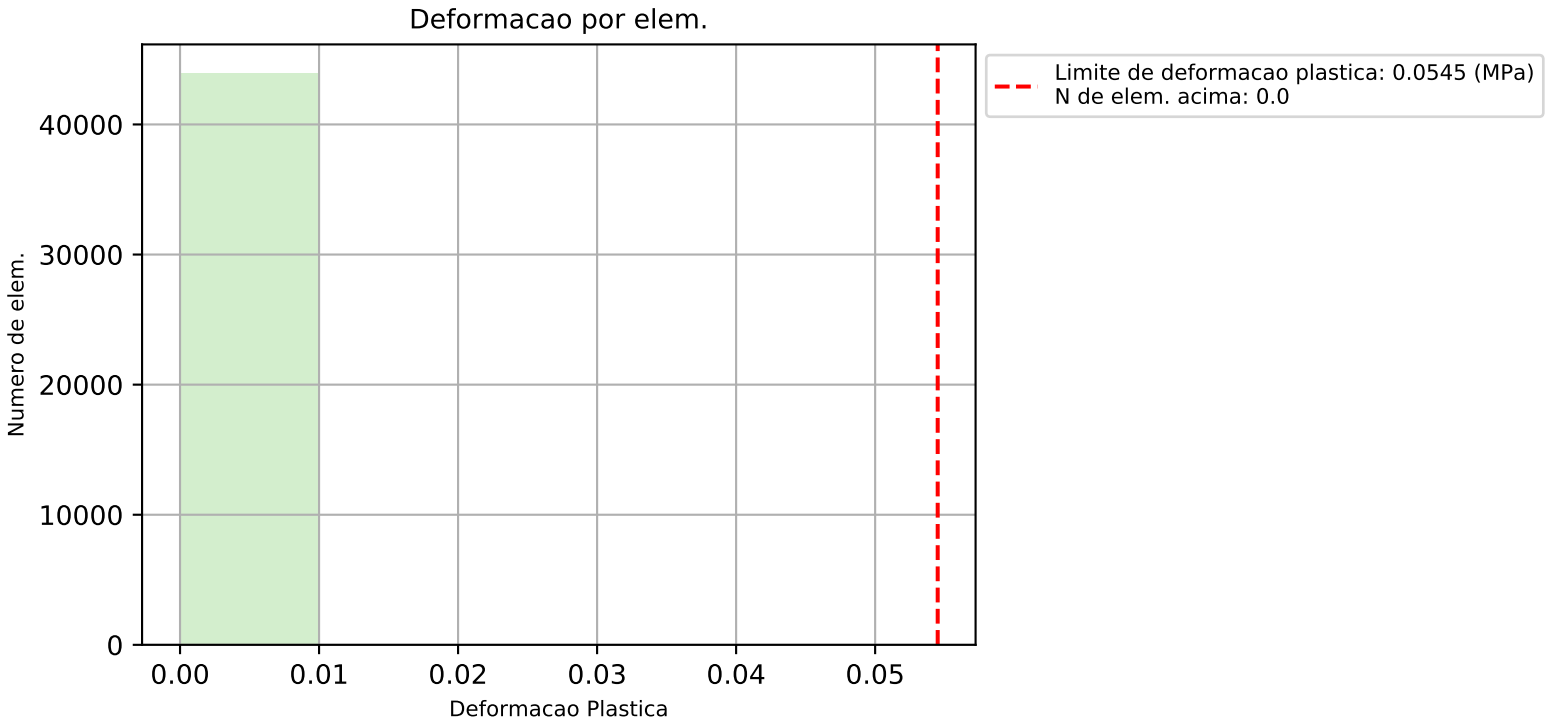

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.808e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.712e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.837e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.997e+00 (mm)
t = 2.000e+00 (s)

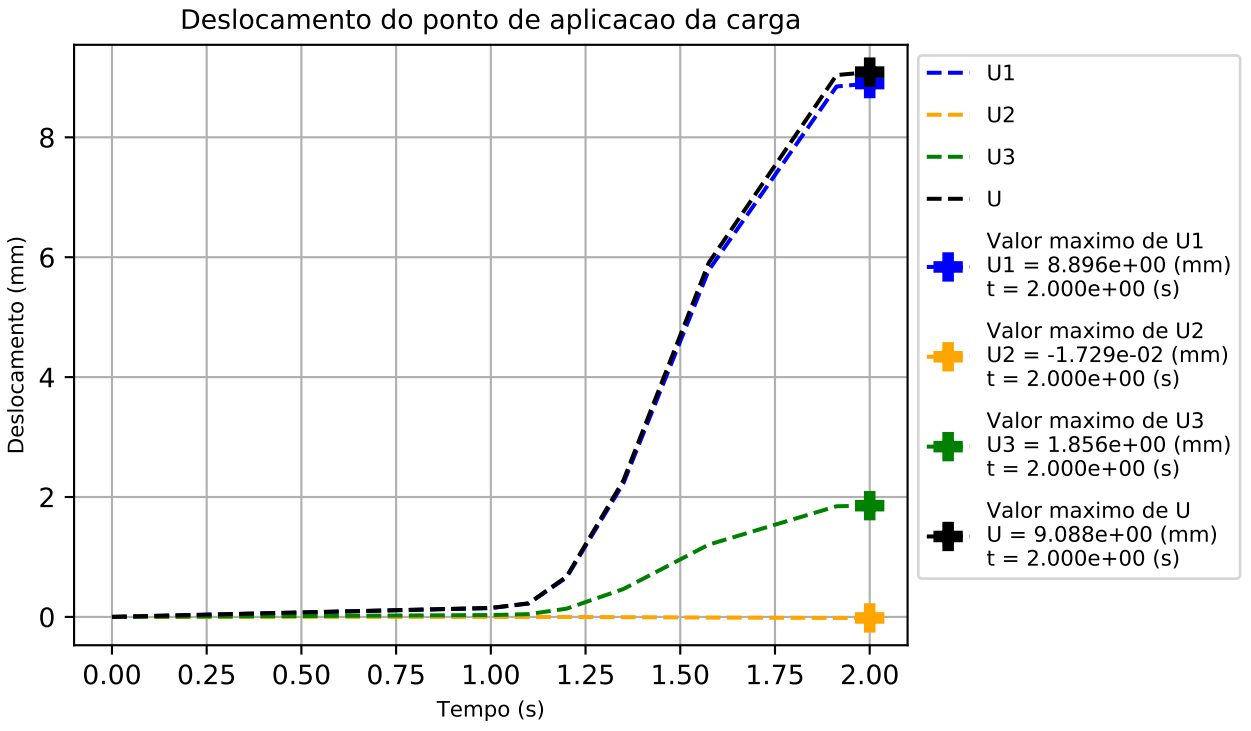

Tensao por elem.
Largura_15.0 Altura_2.0 Espessura_3.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

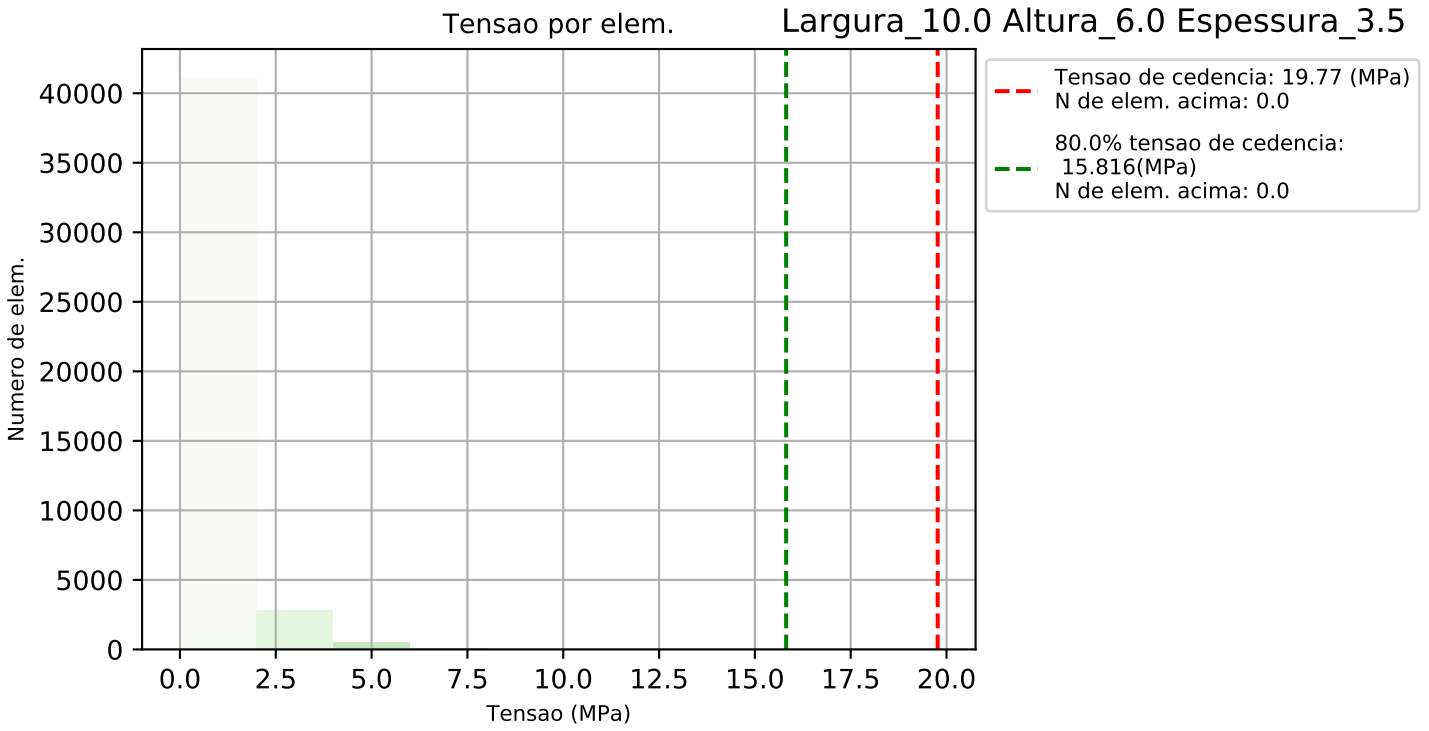

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

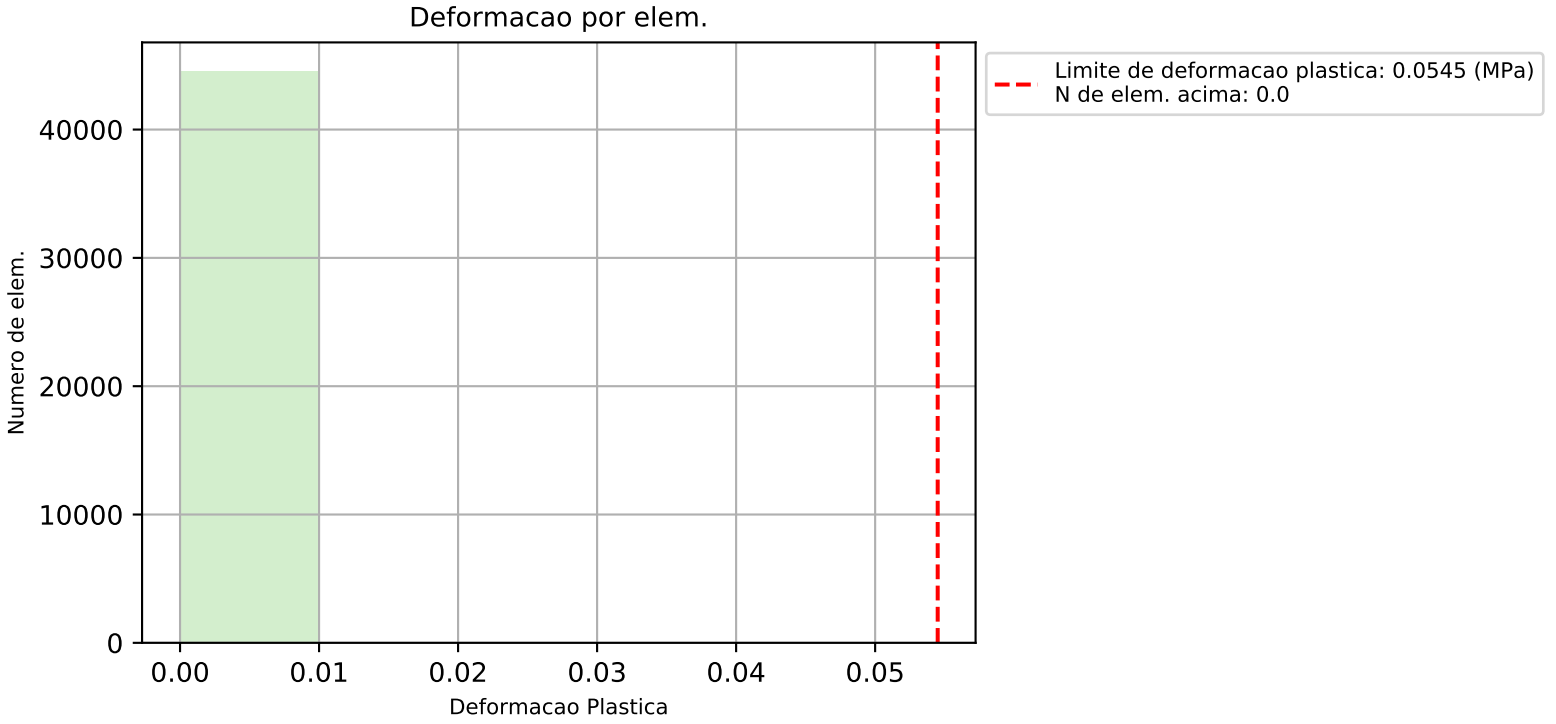

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.829e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.716e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.842e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 9.019e+00 (mm)
t = 2.000e+00 (s)

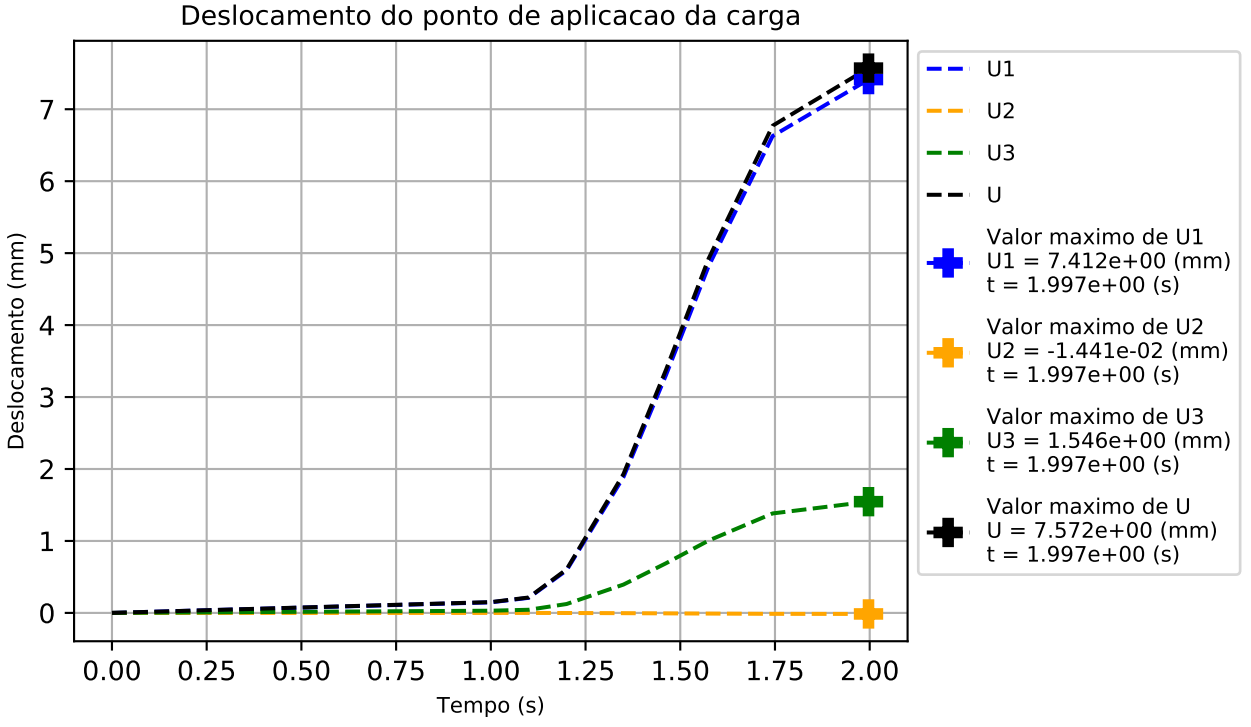

Tensao por elem.
Largura_15.0 Altura_2.0 Espessura_2.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

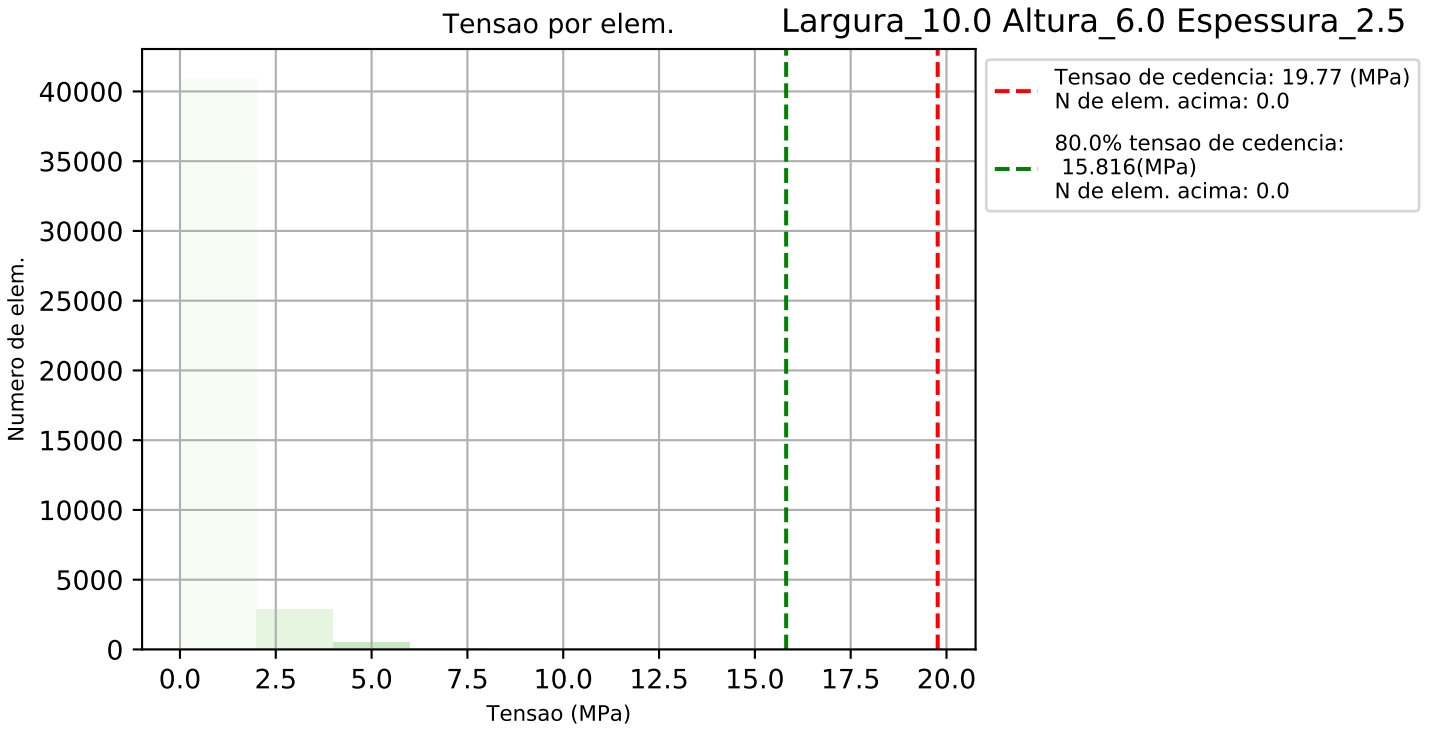

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

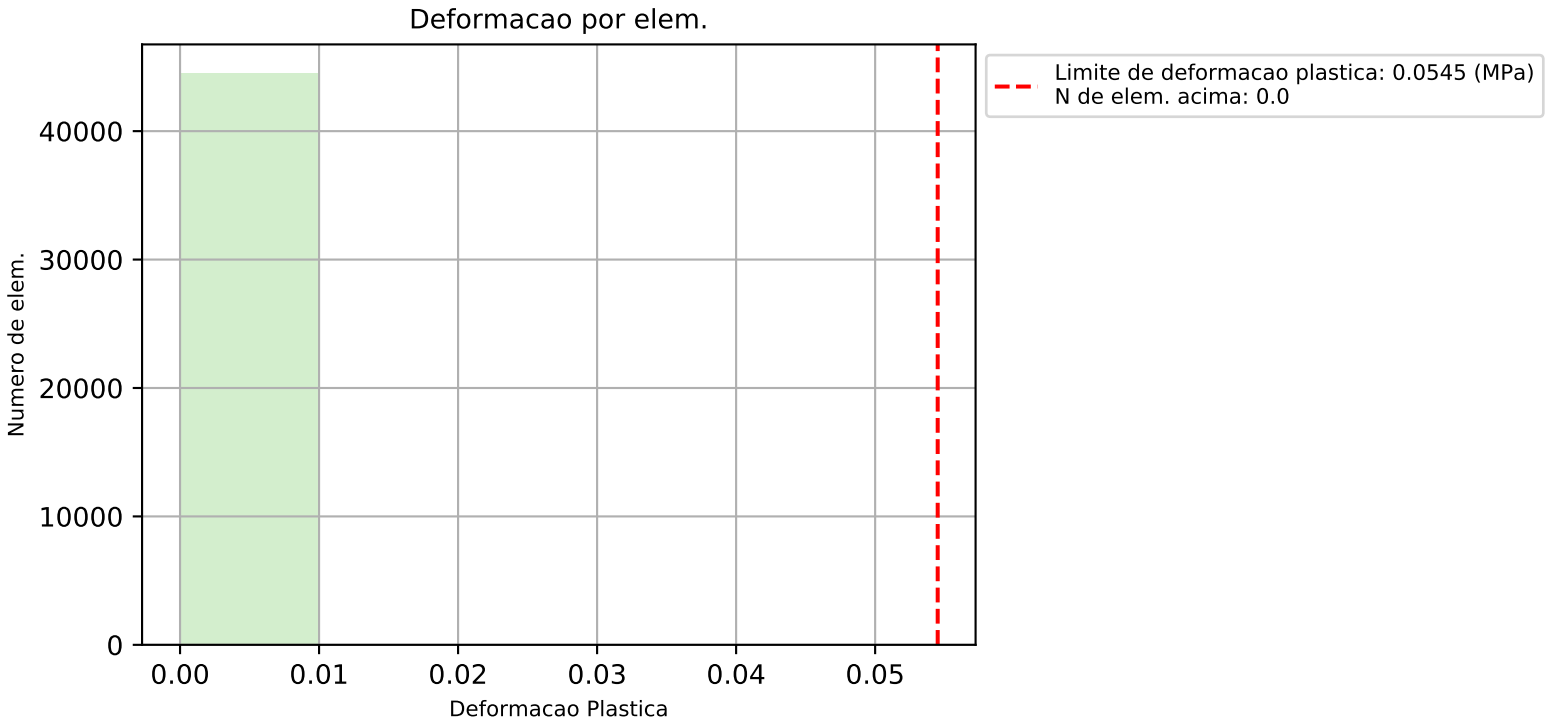

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.896e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.729e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.856e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 9.088e+00 (mm)
t = 2.000e+00 (s)

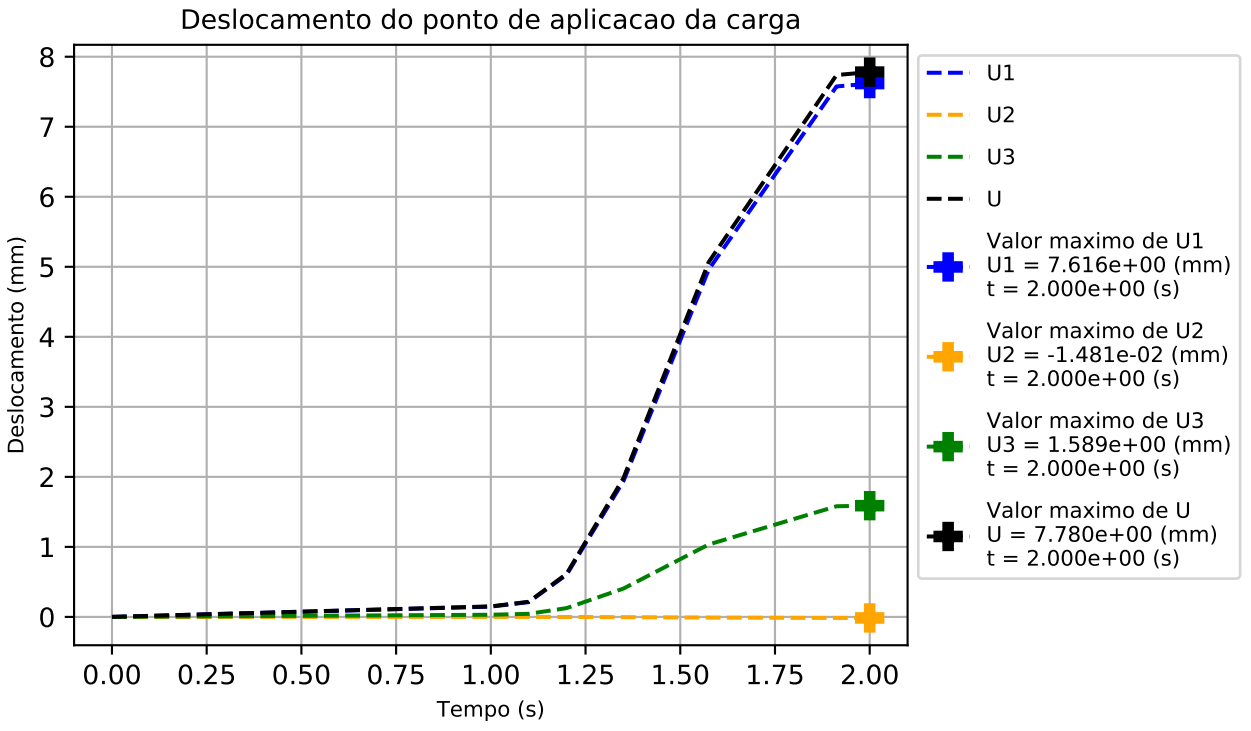

Tensao por elem.
Largura_10.0 Altura_6.0 Espessura_3.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

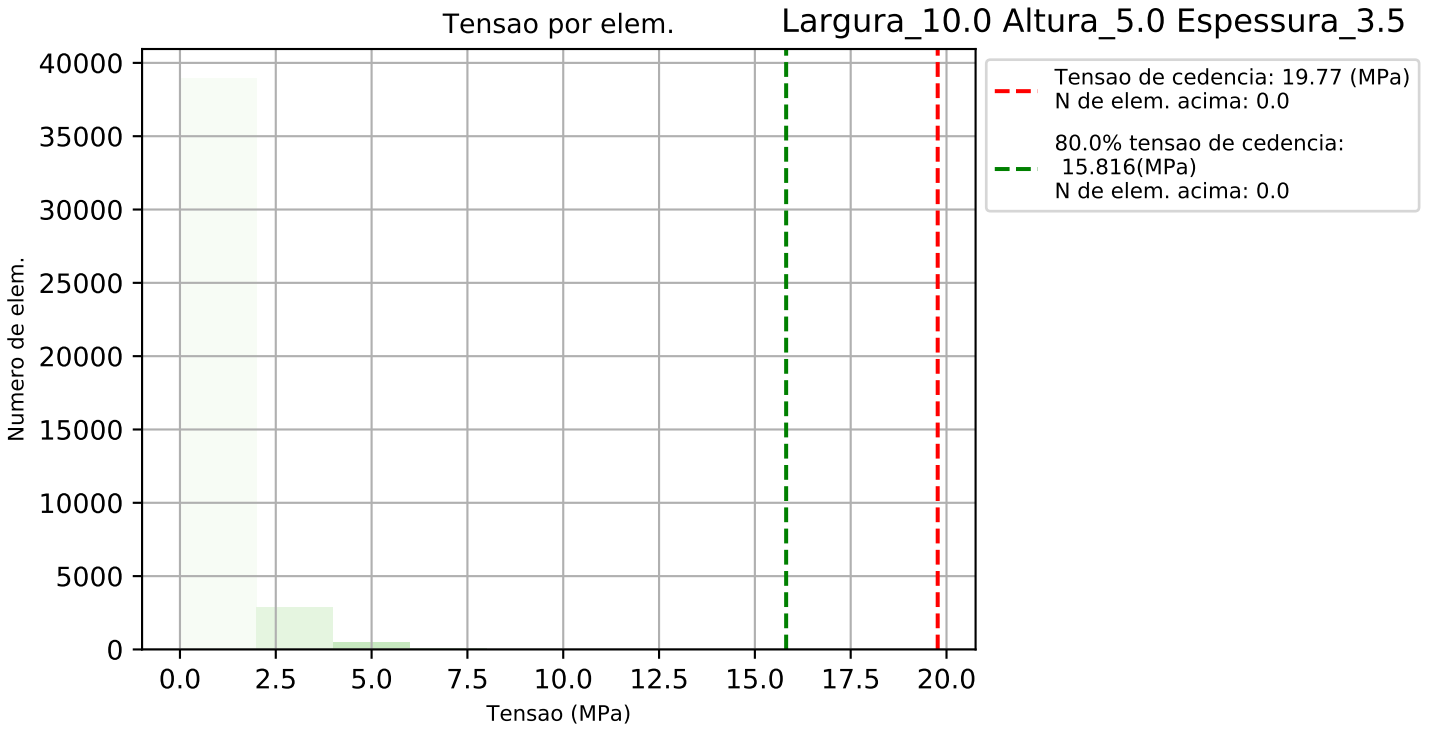

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

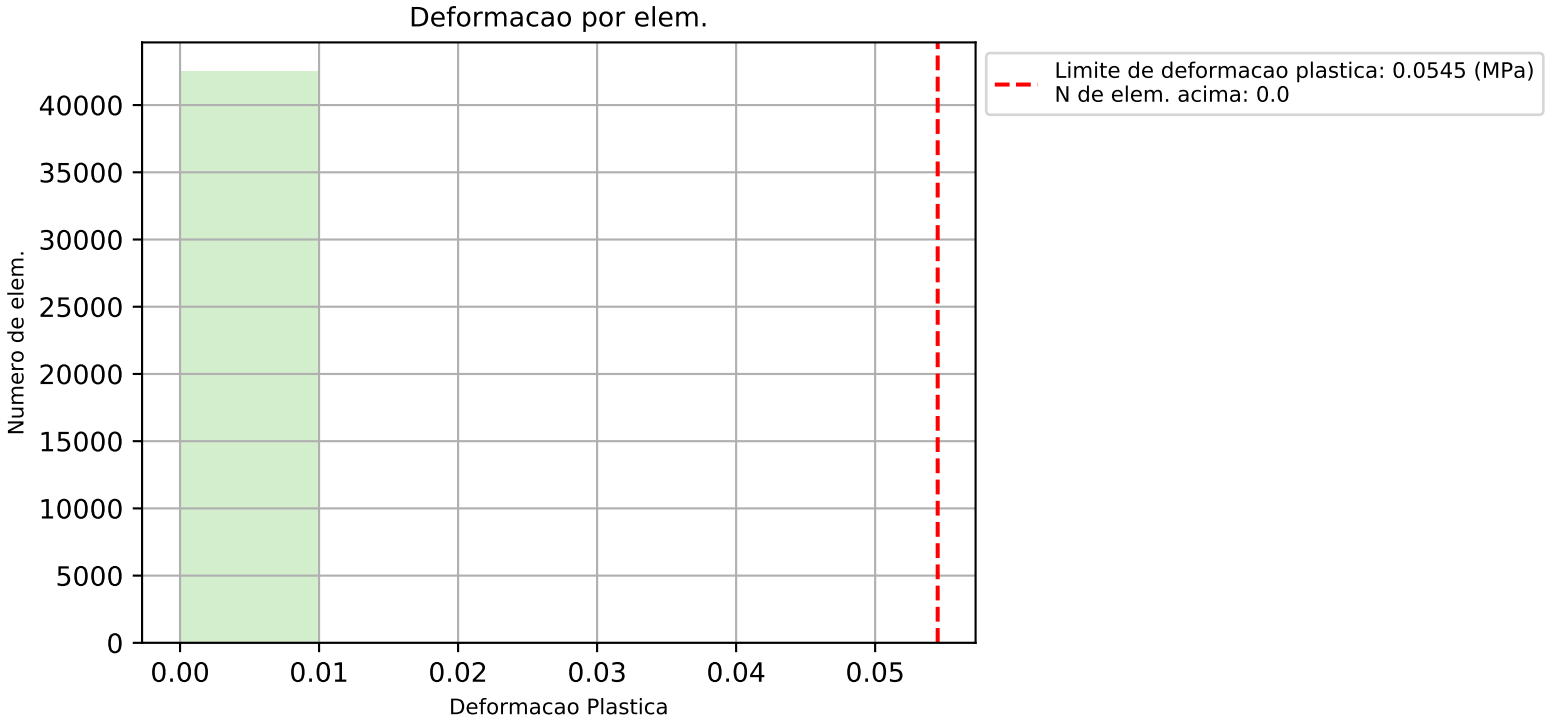

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 7.412e+00 (mm)
t = 1.997e+00 (s)
Valor maximo de U2
U2 = -1.441e-02 (mm)
t = 1.997e+00 (s)
Valor maximo de U3
U3 = 1.546e+00 (mm)
t = 1.997e+00 (s)
Valor maximo de U
U = 7.572e+00 (mm)
t = 1.997e+00 (s)

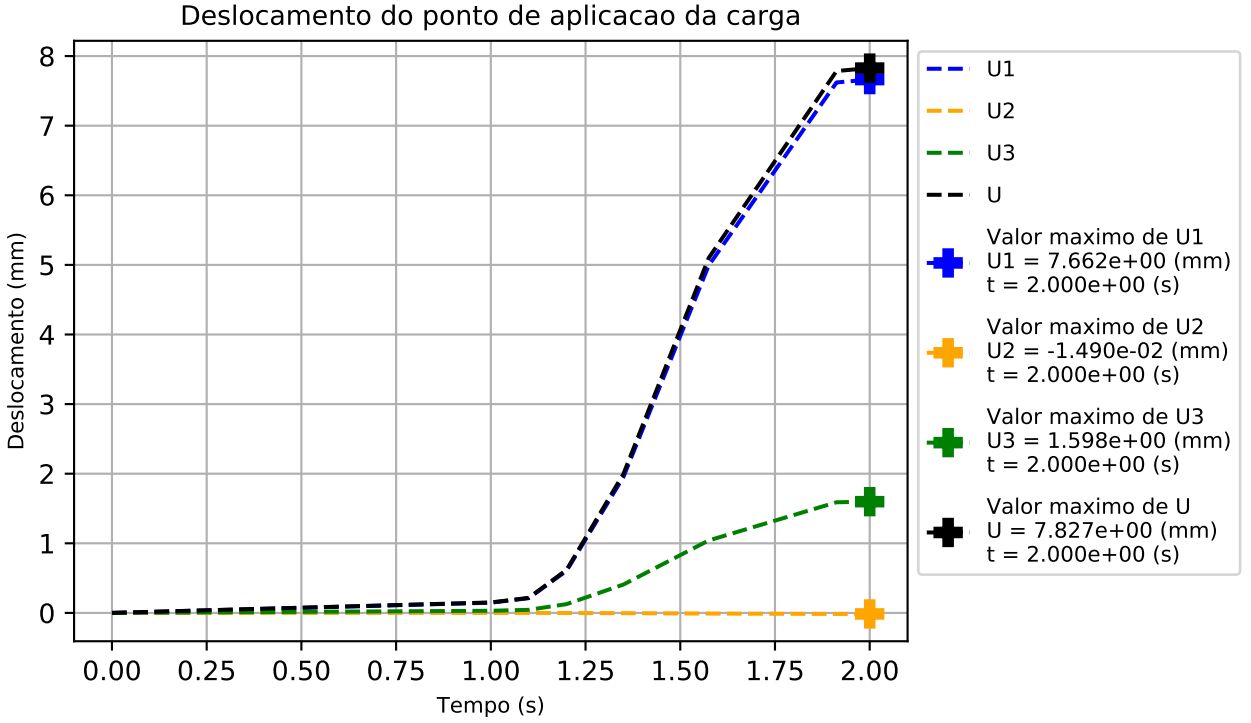

Tensao por elem.
Largura_10.0 Altura_6.0 Espessura_2.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

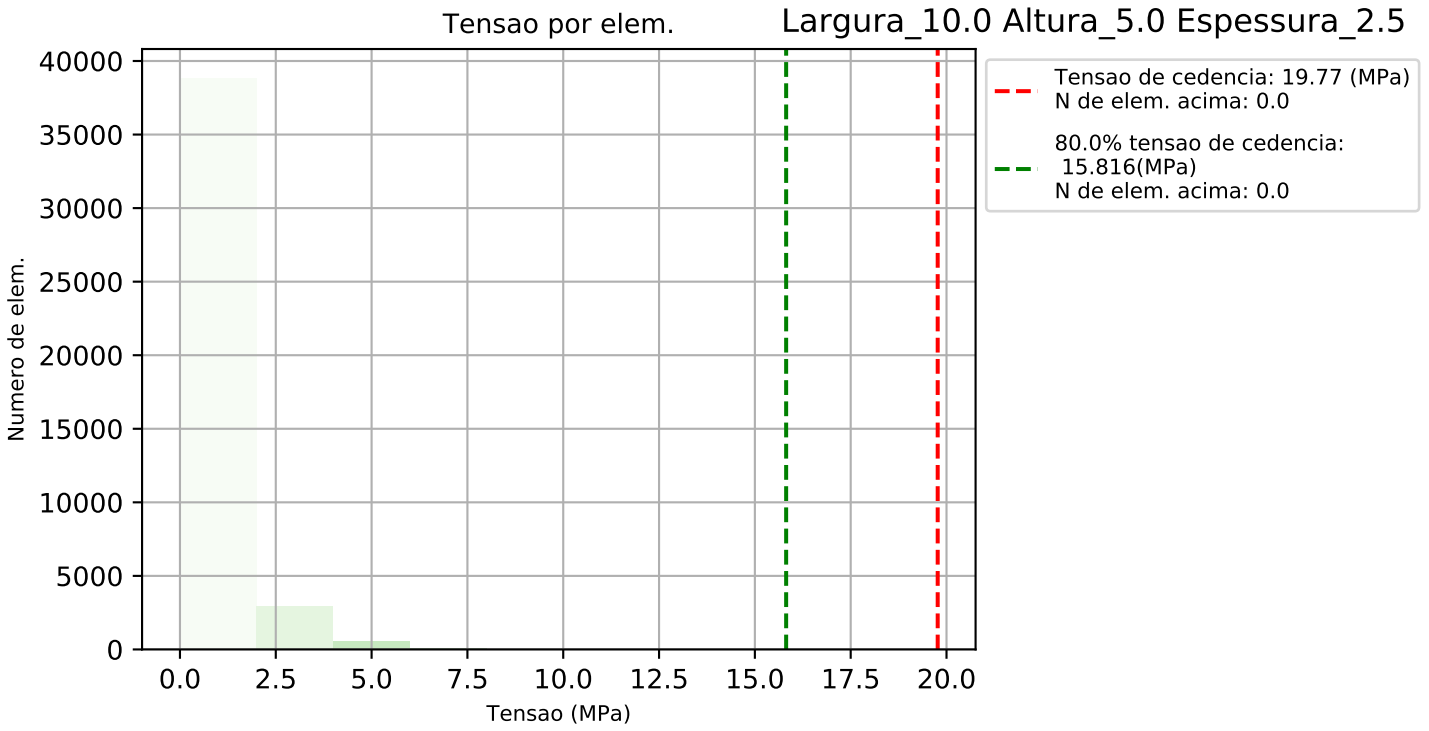

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

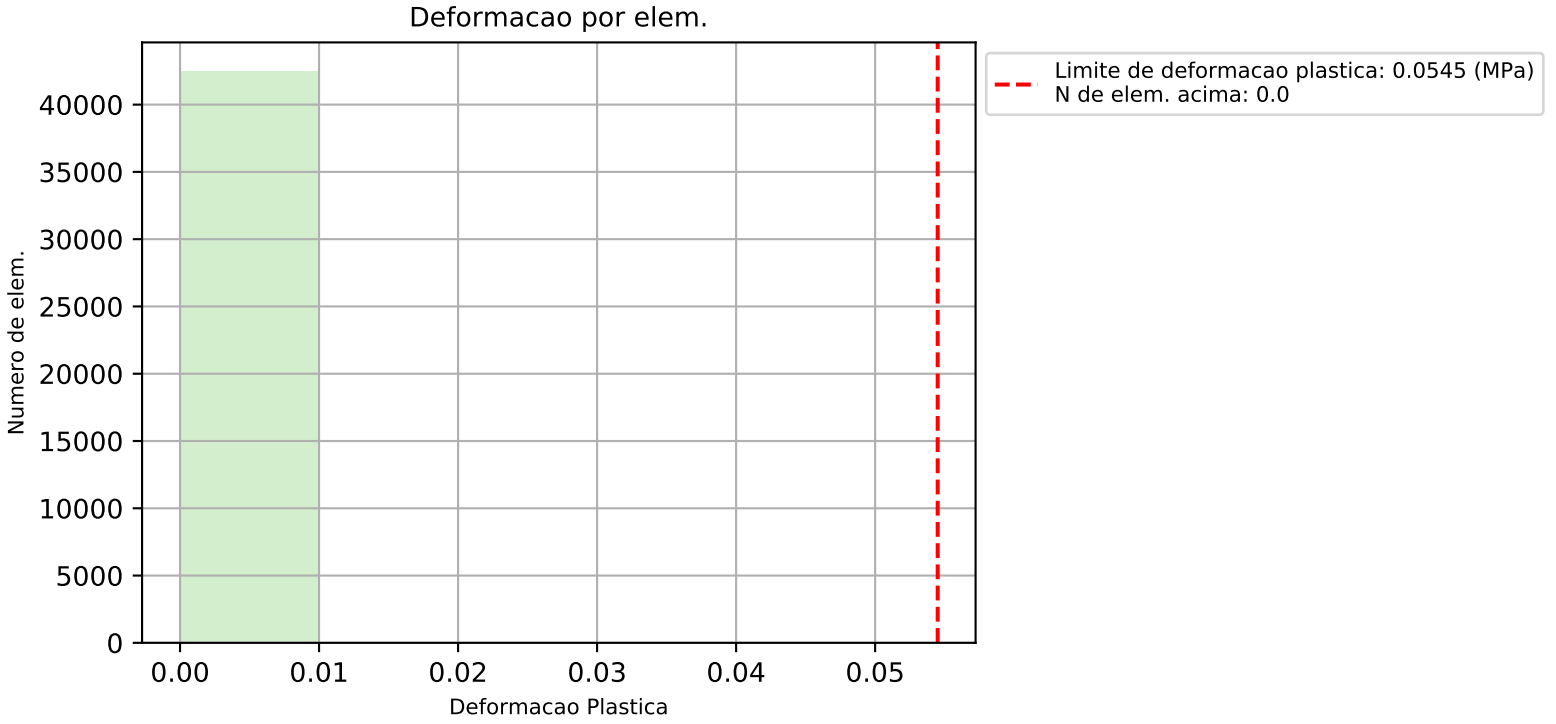

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 7.616e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.481e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.589e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 7.780e+00 (mm)
t = 2.000e+00 (s)

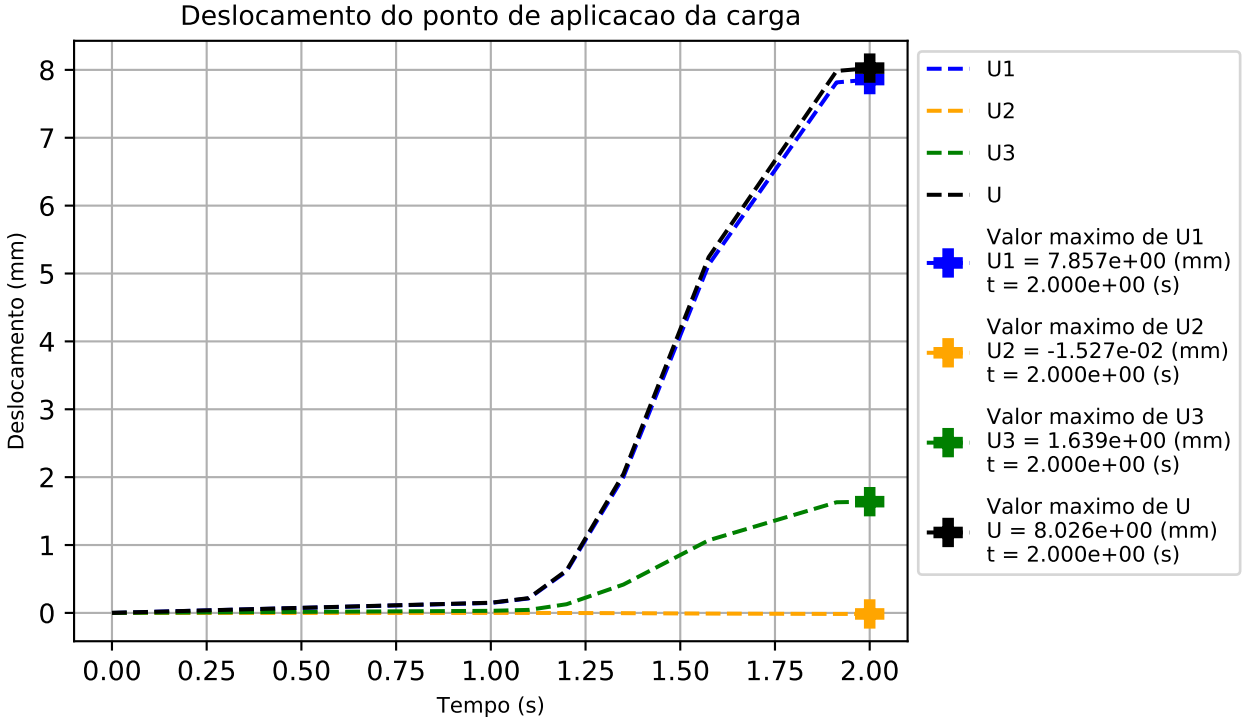

Tensao por elem.
Largura_10.0 Altura_5.0 Espessura_3.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

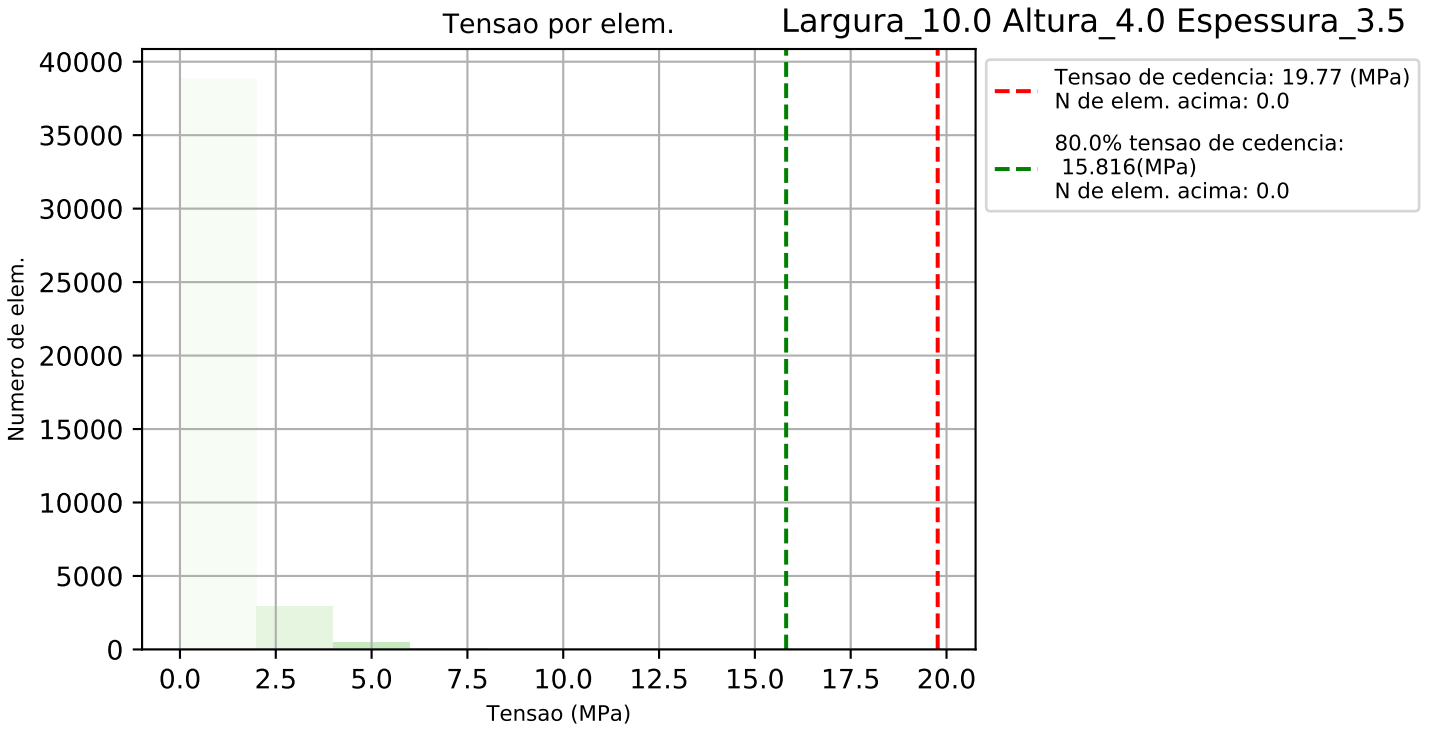

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

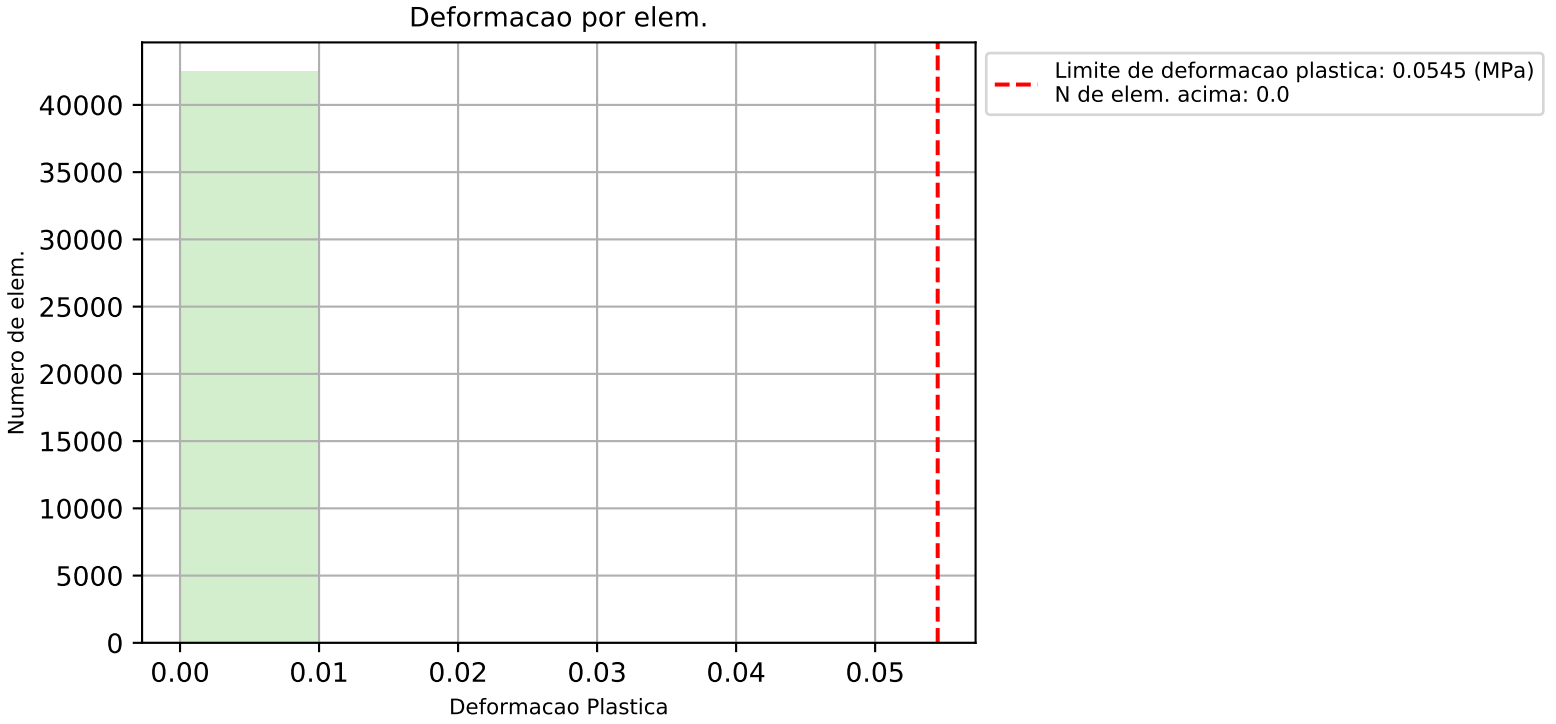

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 7.662e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.490e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.598e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 7.827e+00 (mm)
t = 2.000e+00 (s)

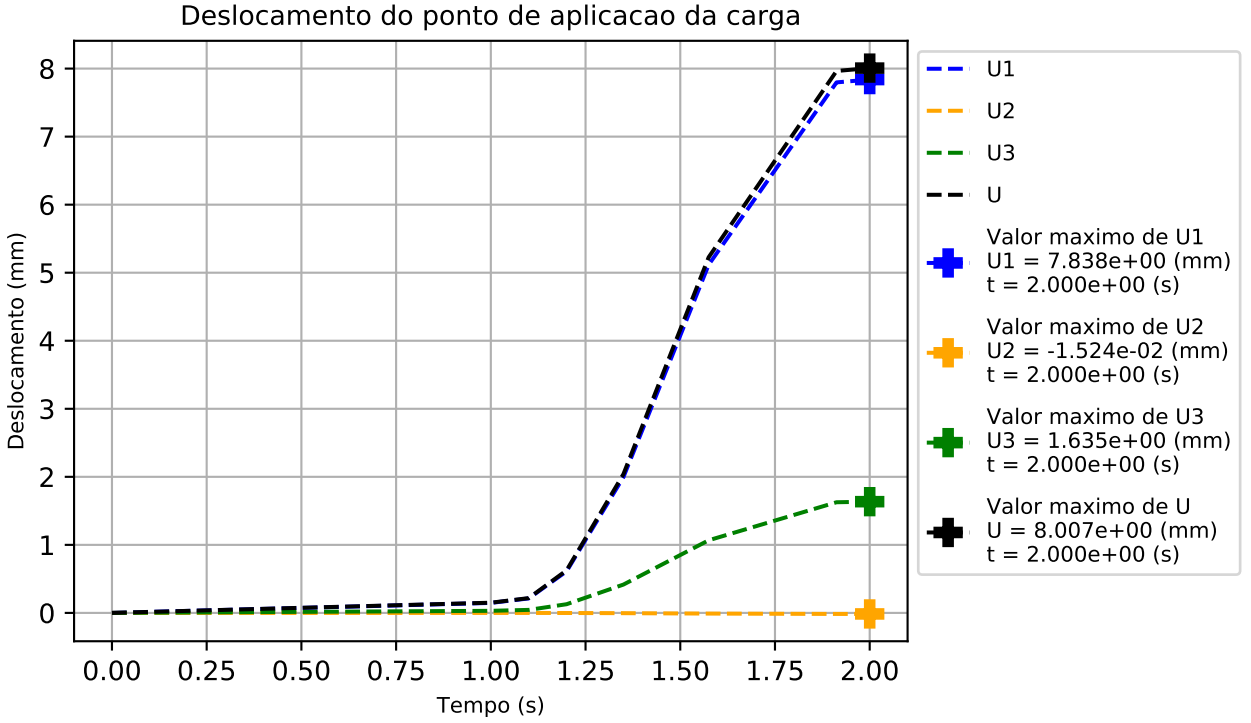

Tensao por elem.
Largura_10.0 Altura_5.0 Espessura_2.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

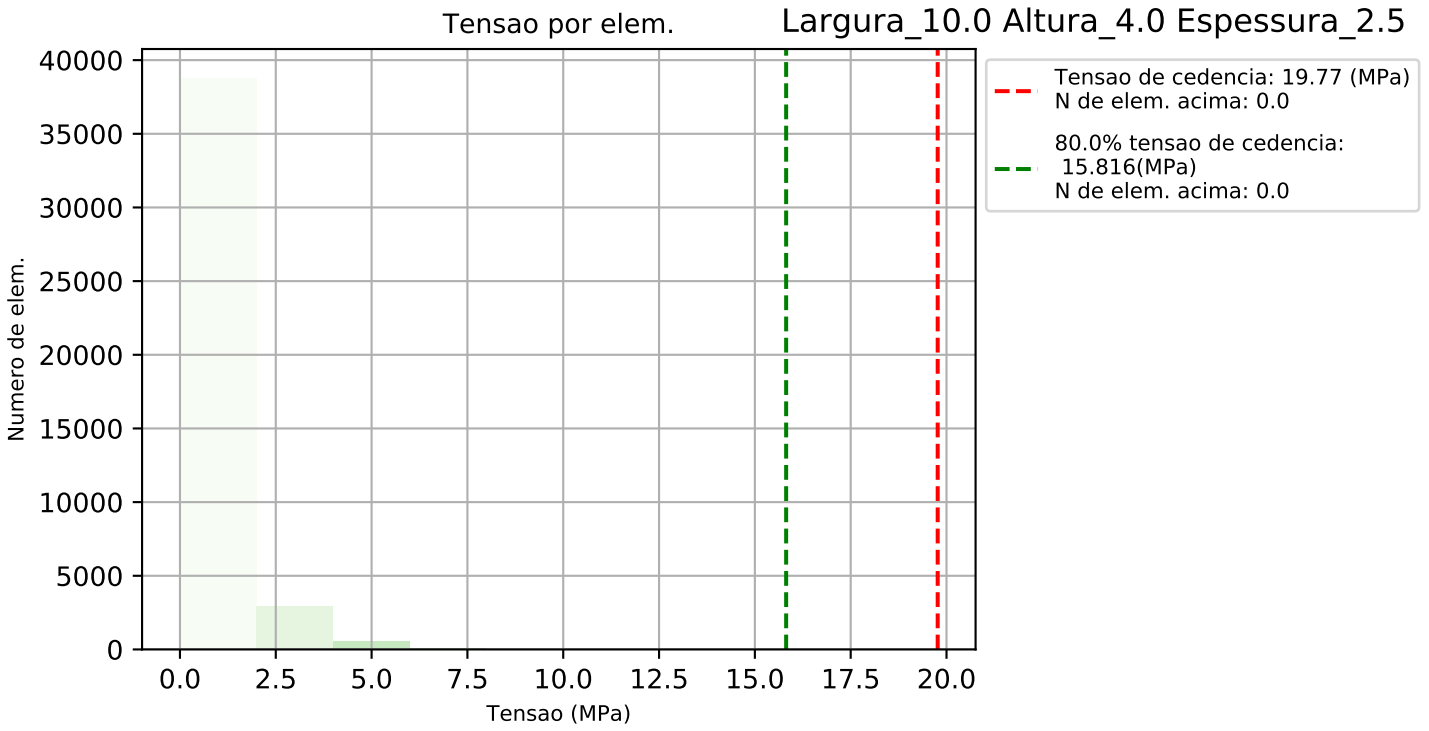

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

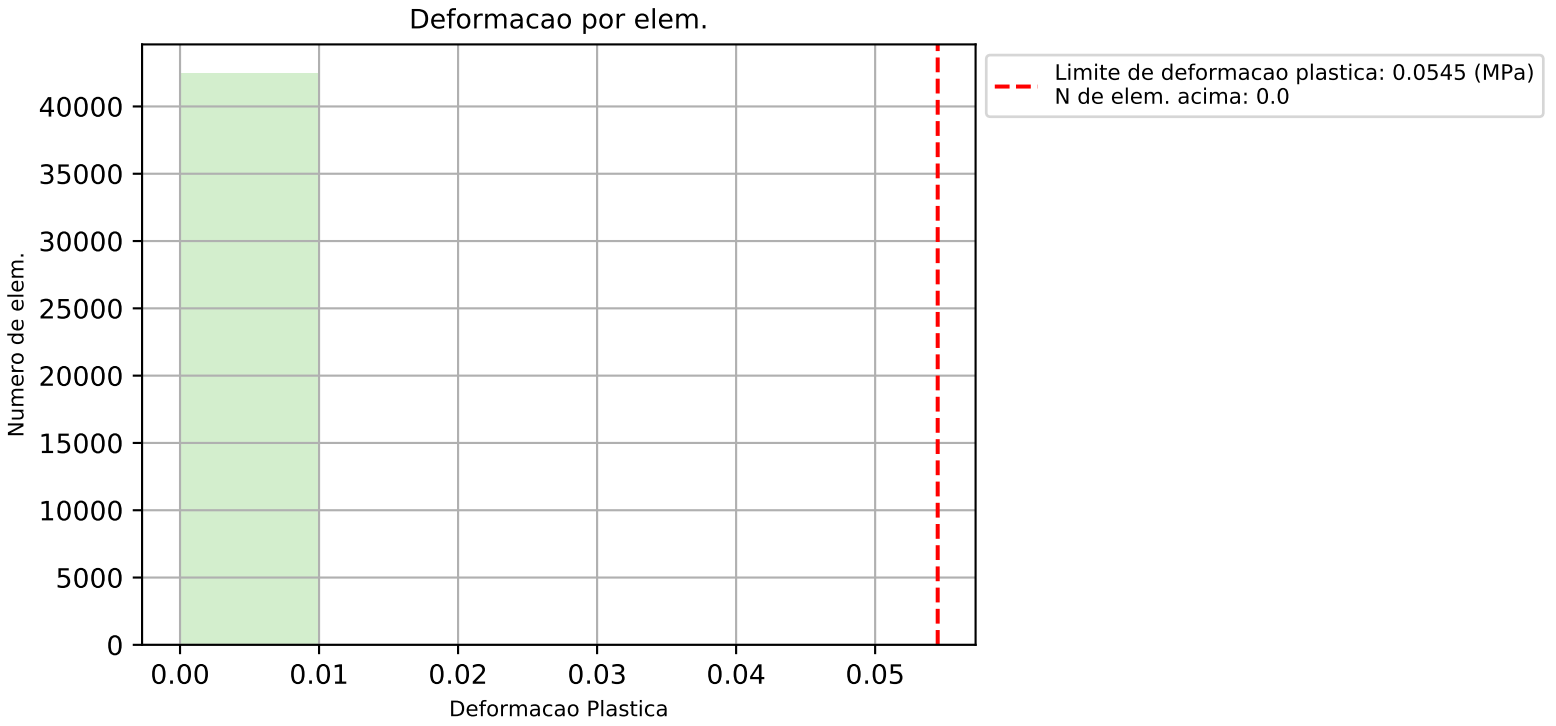

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 7.857e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.527e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.639e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.026e+00 (mm)
t = 2.000e+00 (s)

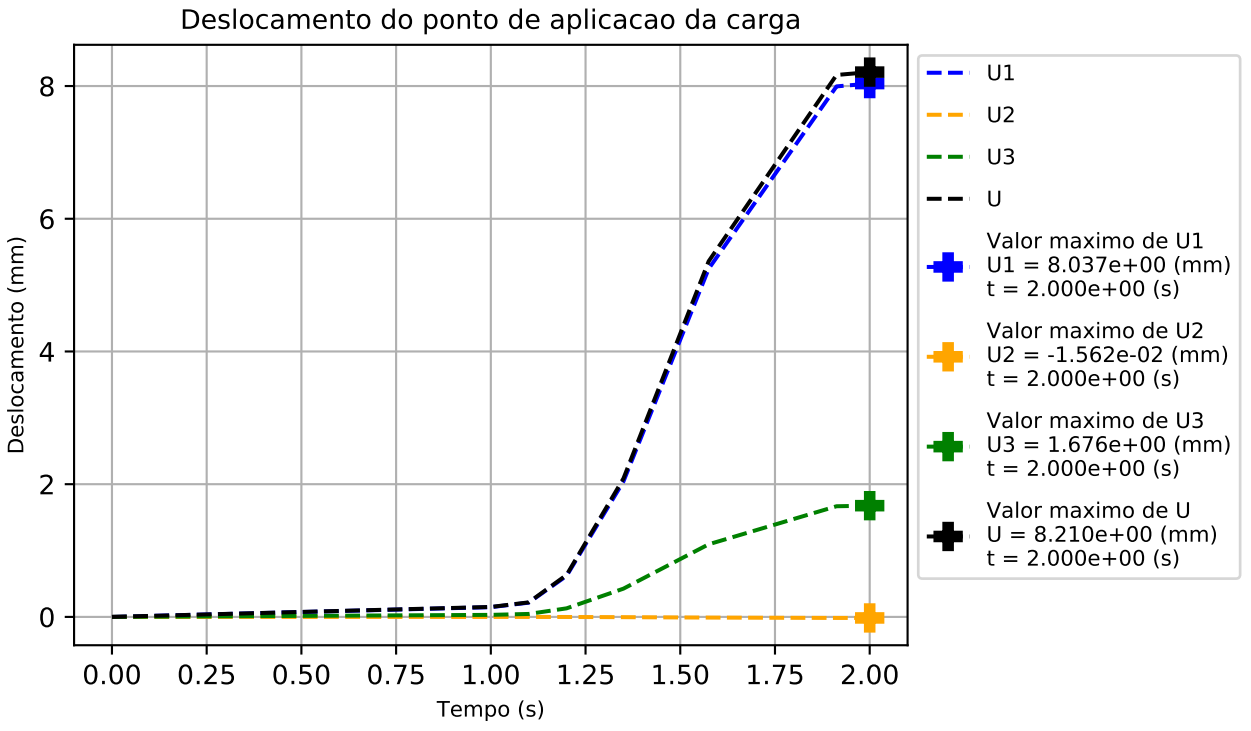

Tensao por elem.
Largura_10.0 Altura_4.0 Espessura_3.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

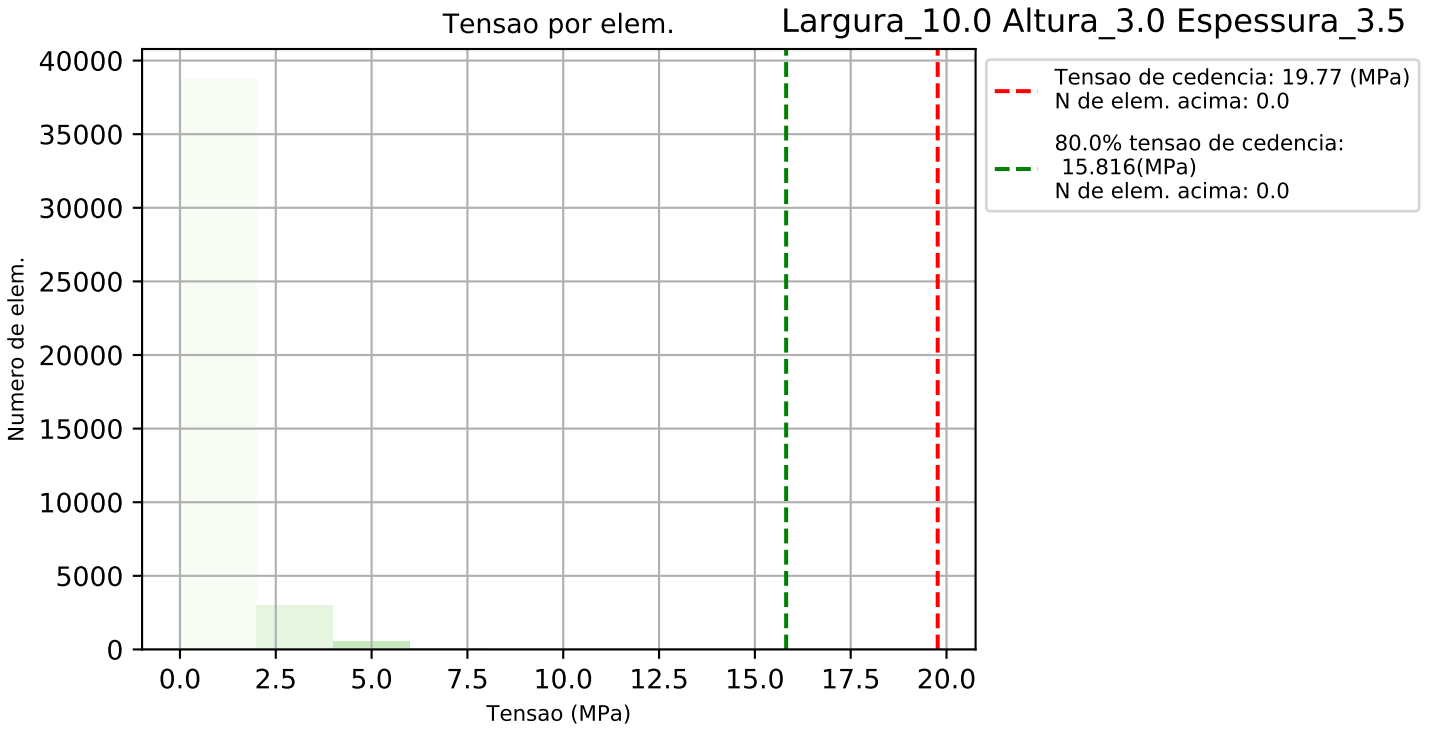

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

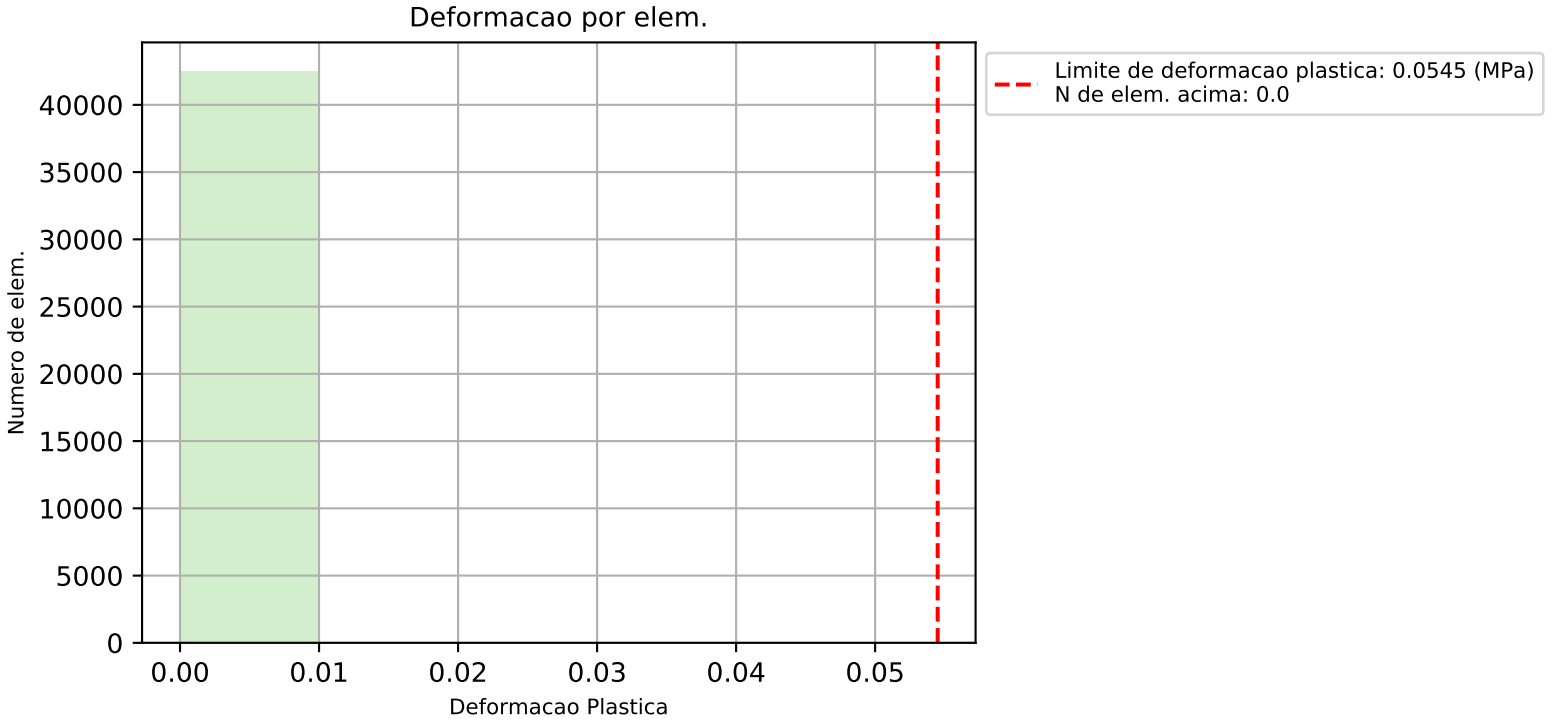

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 7.838e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.524e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.635e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.007e+00 (mm)
t = 2.000e+00 (s)

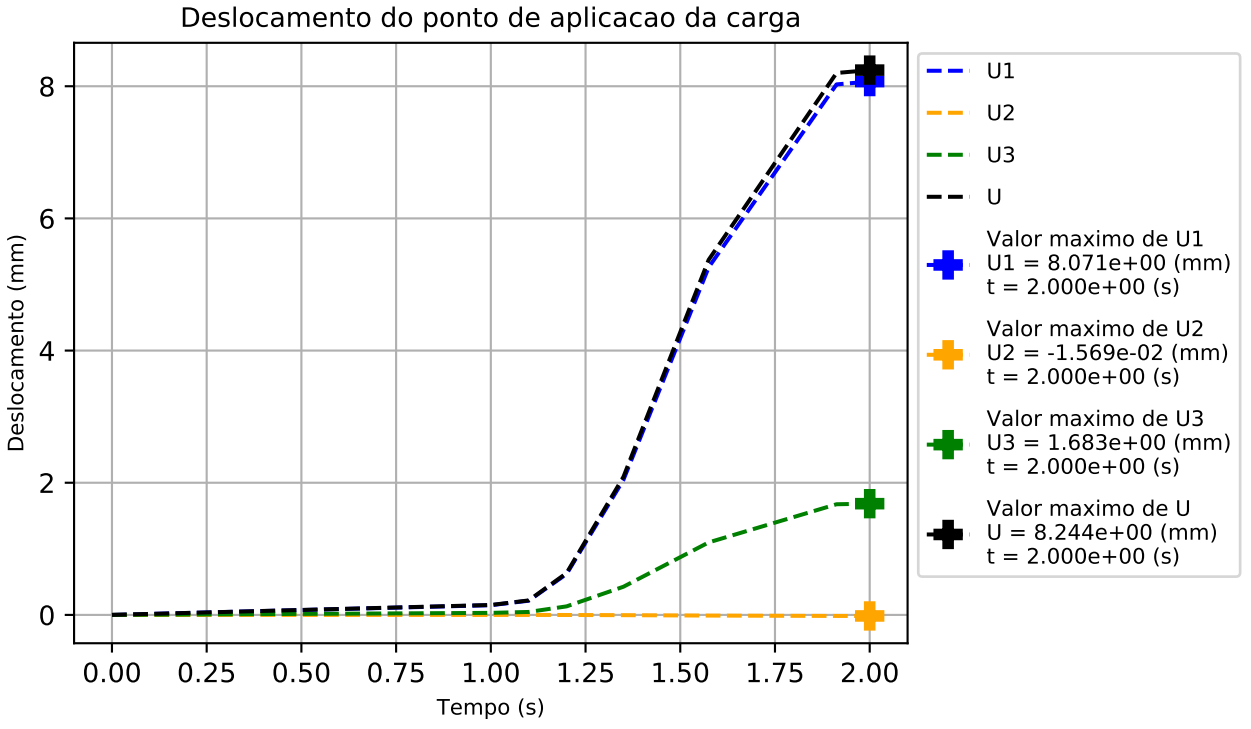

Tensao por elem.
Largura_10.0 Altura_4.0 Espessura_2.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

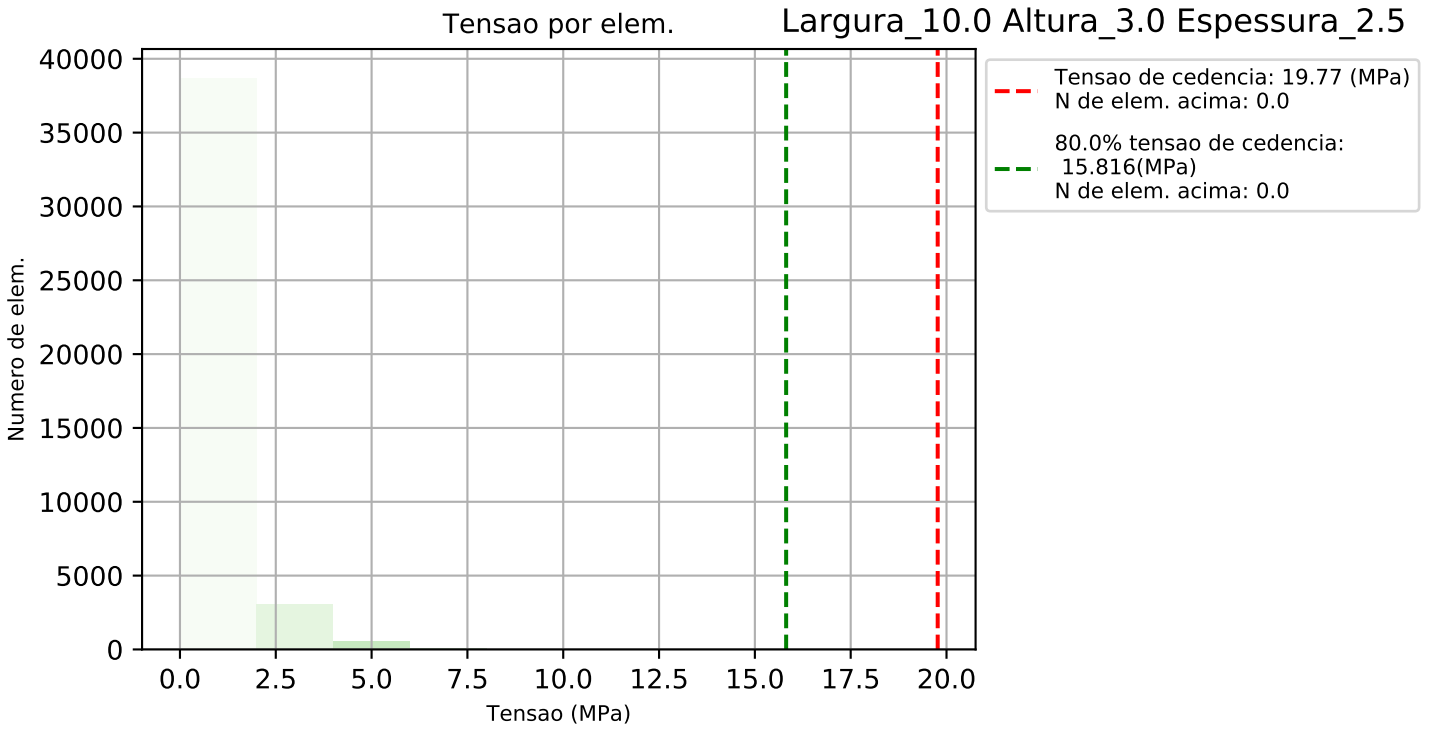

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

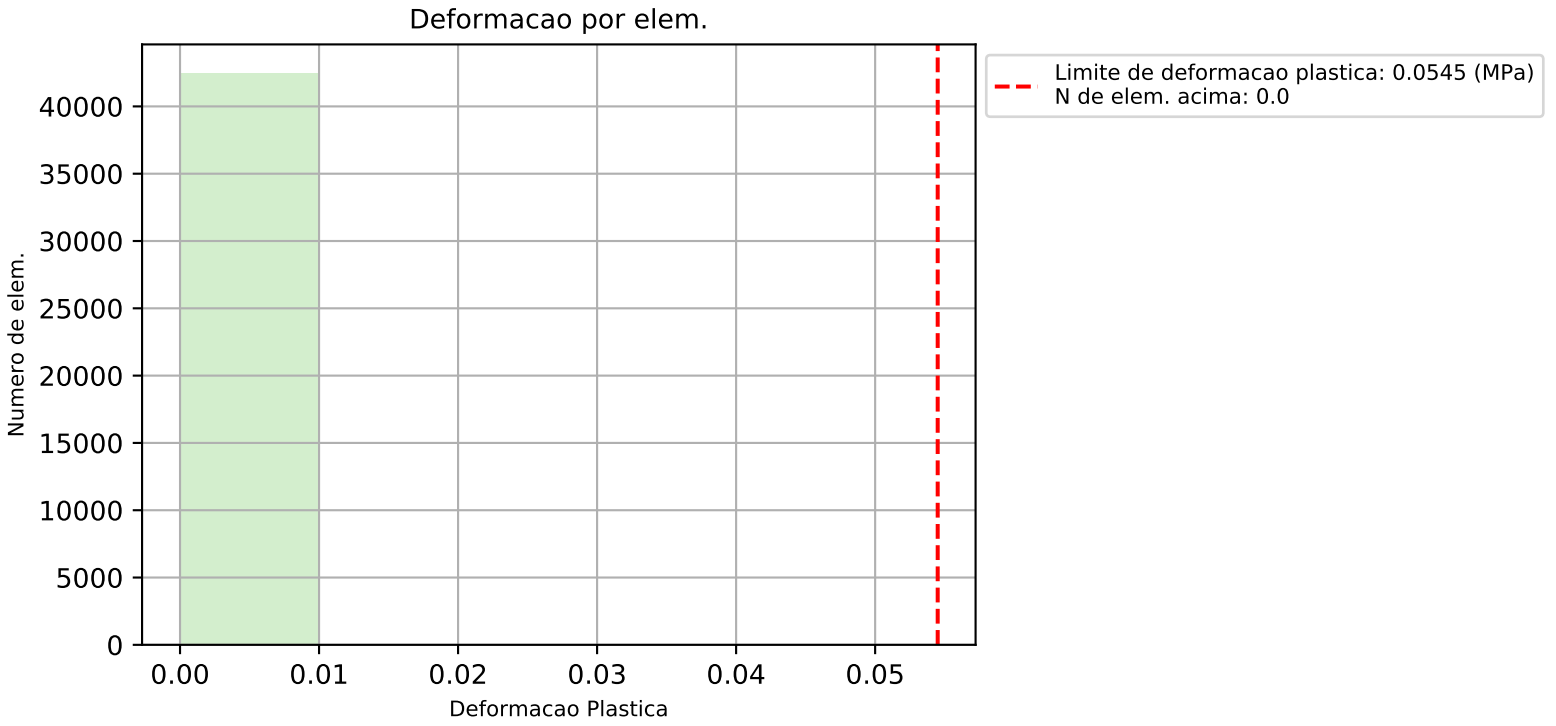

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.037e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.562e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.676e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.210e+00 (mm)
t = 2.000e+00 (s)

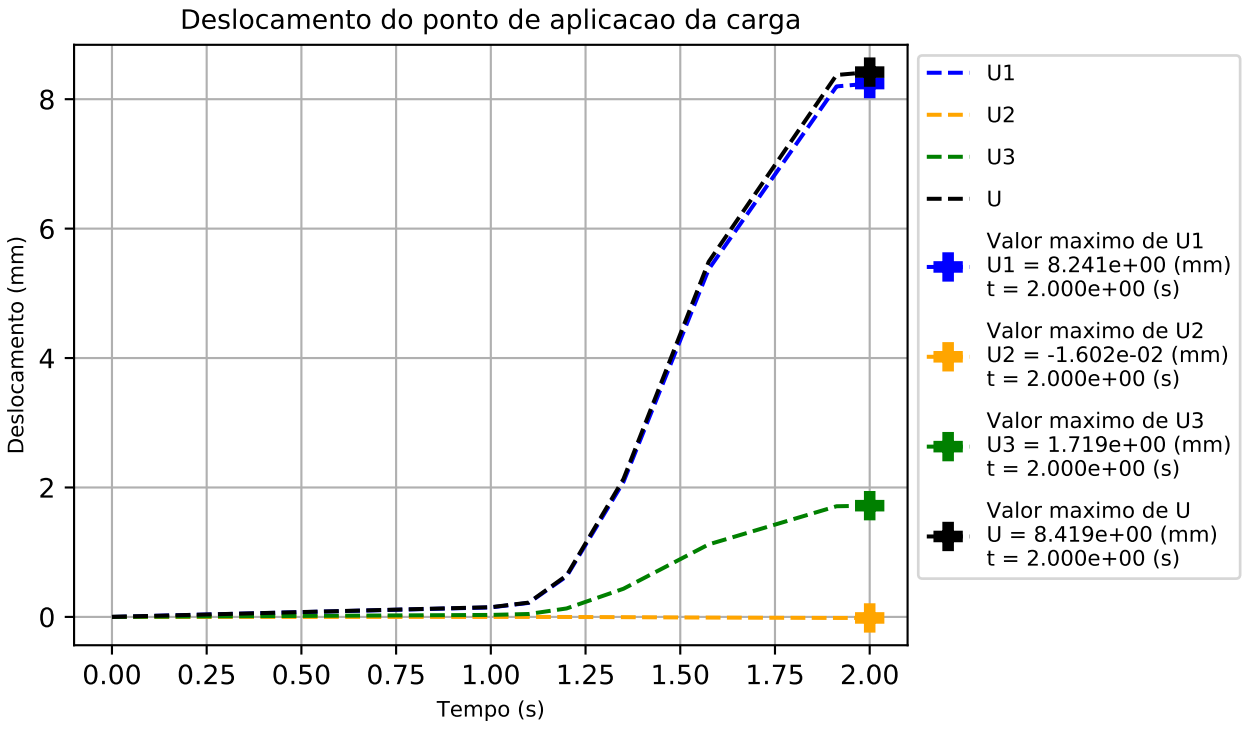

Tensao por elem.
Largura_10.0 Altura_3.0 Espessura_3.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

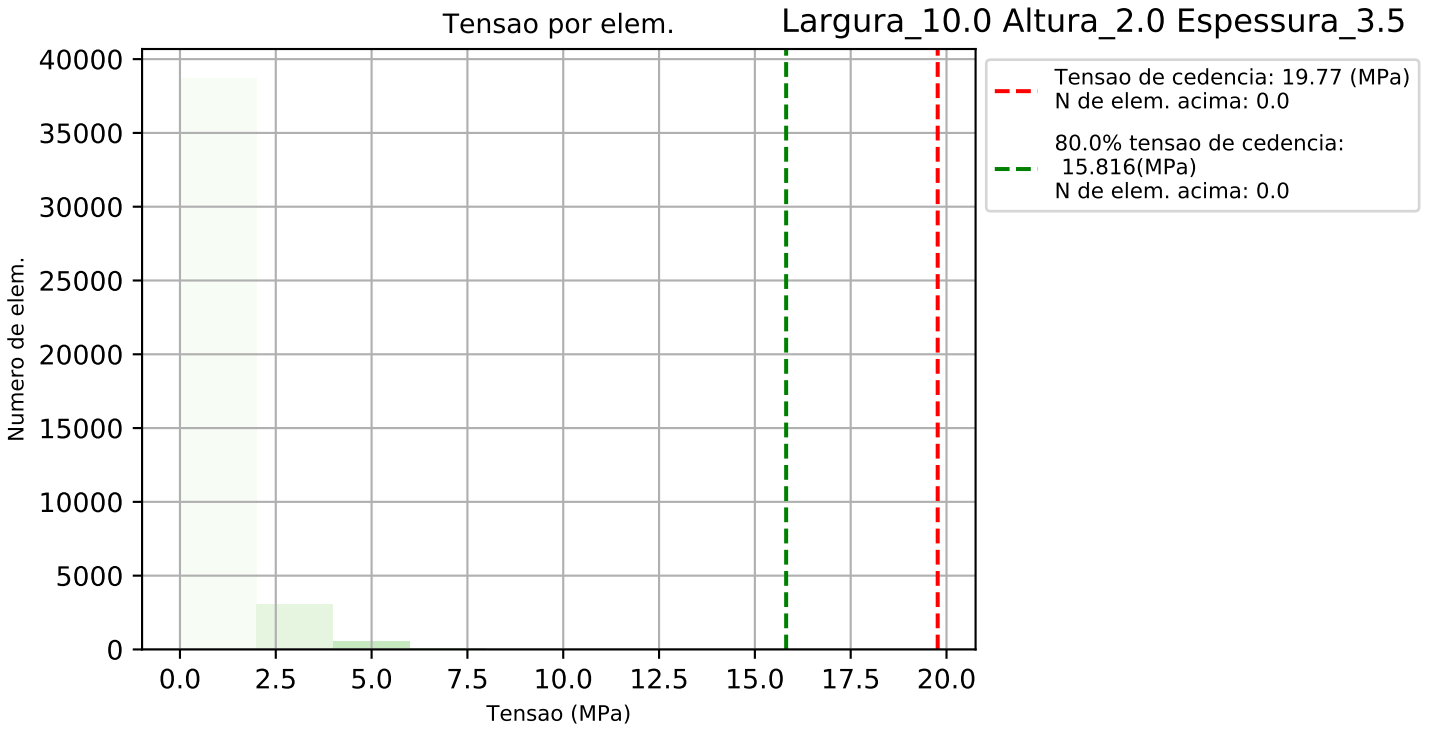

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

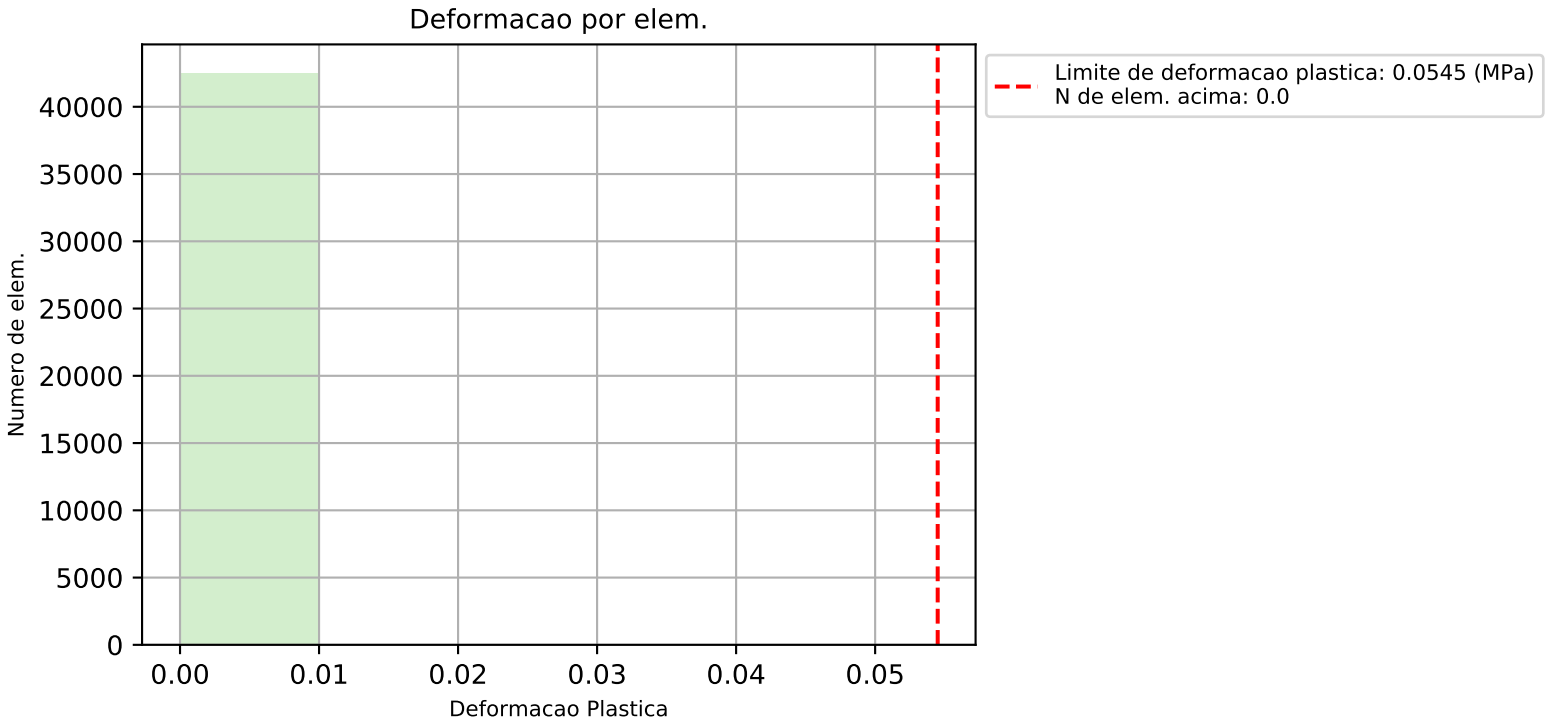

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.071e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.569e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.683e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.244e+00 (mm)
t = 2.000e+00 (s)

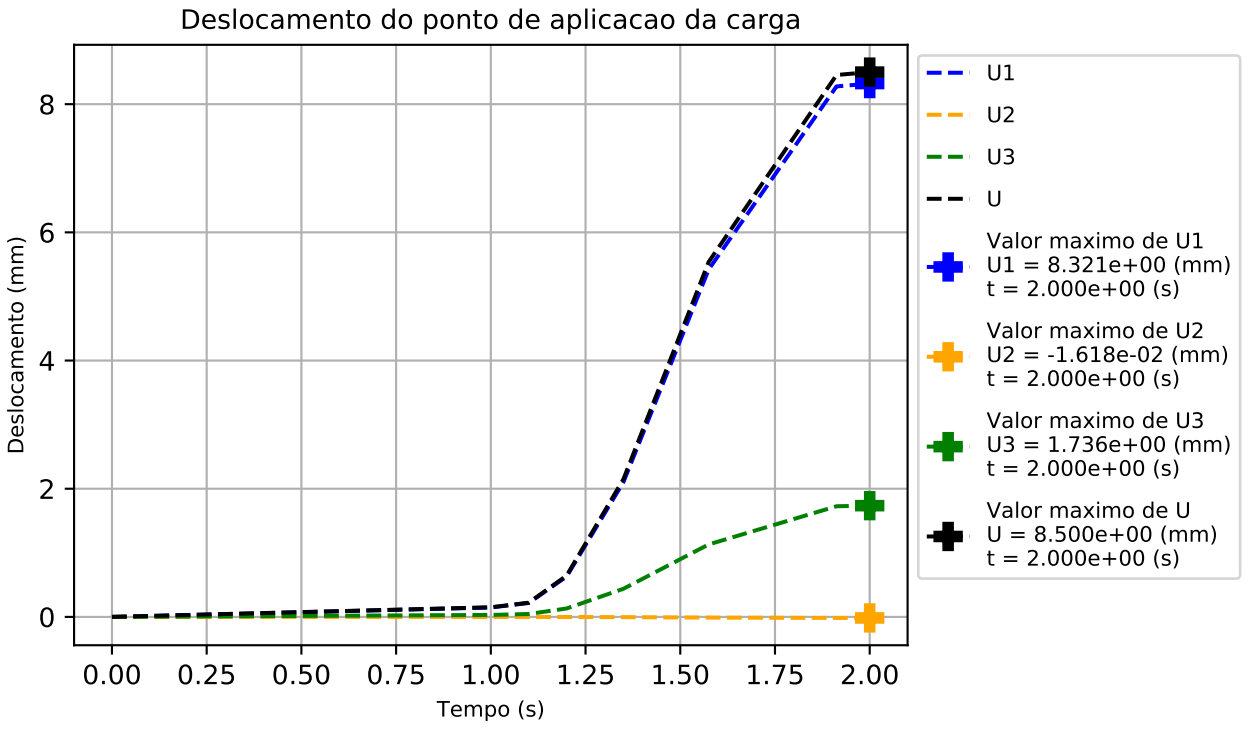

Tensao por elem.
Largura_10.0 Altura_3.0 Espessura_2.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

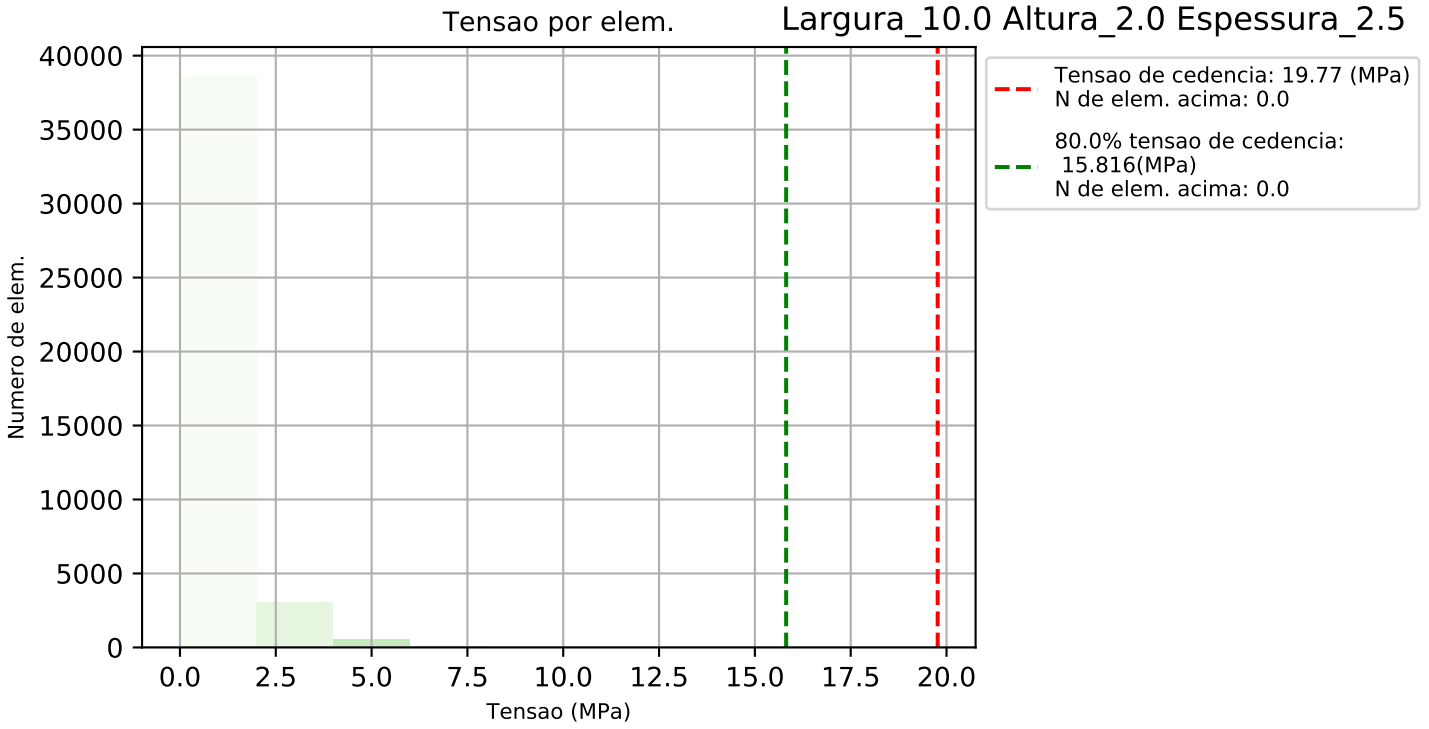

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

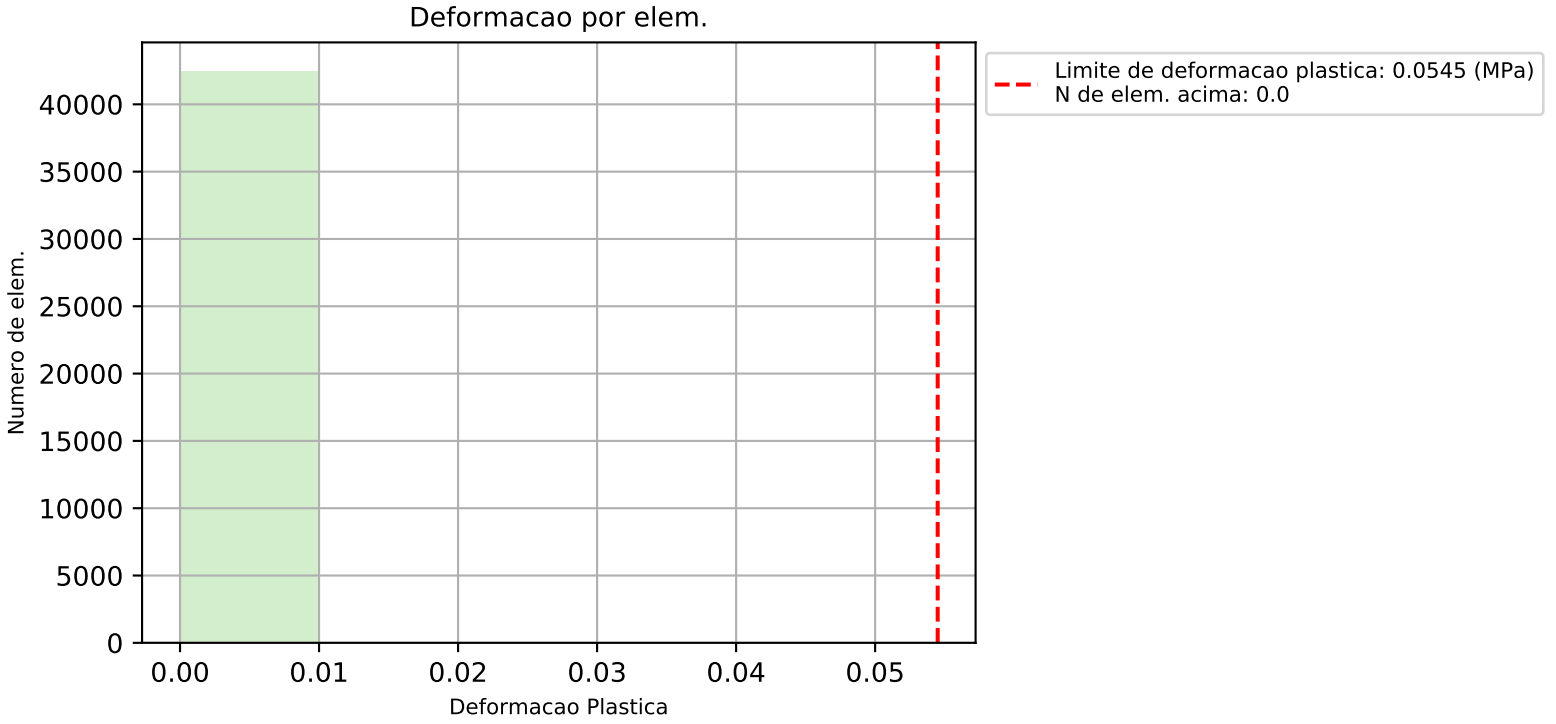

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.241e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.602e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.719e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.419e+00 (mm)
t = 2.000e+00 (s)

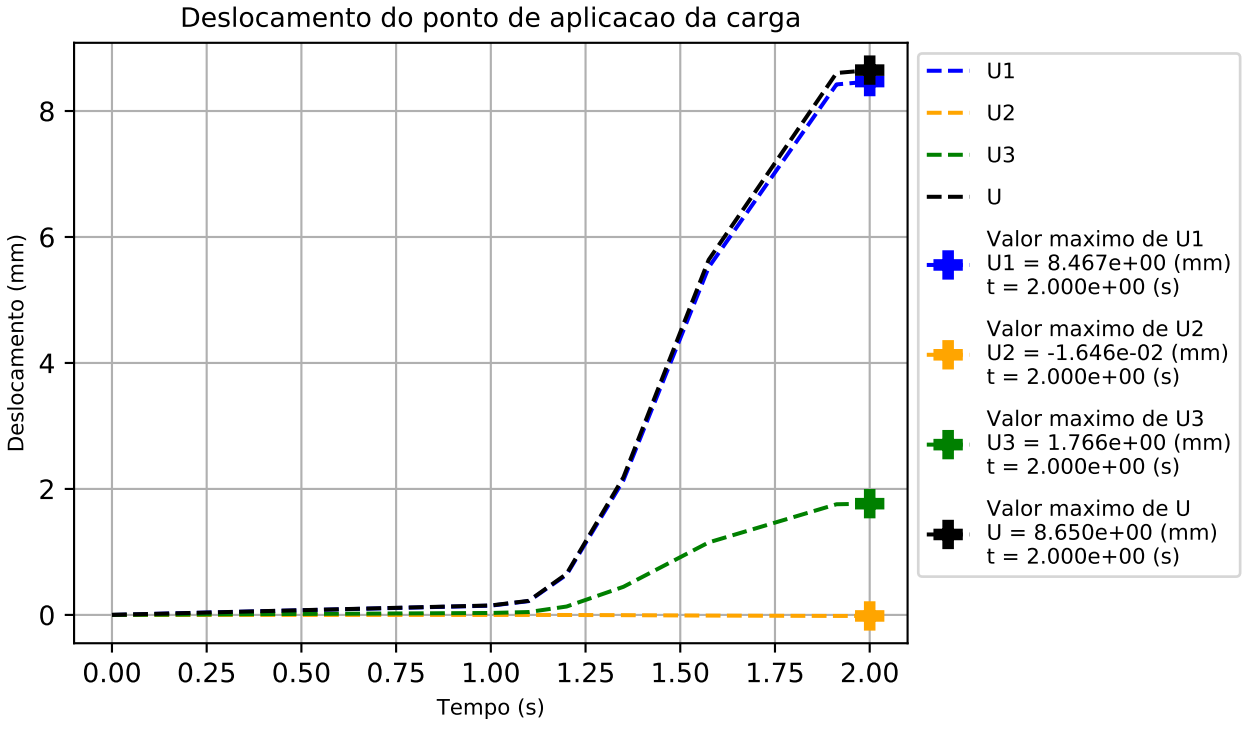

Tensao por elem.
Largura_10.0 Altura_2.0 Espessura_3.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0


Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

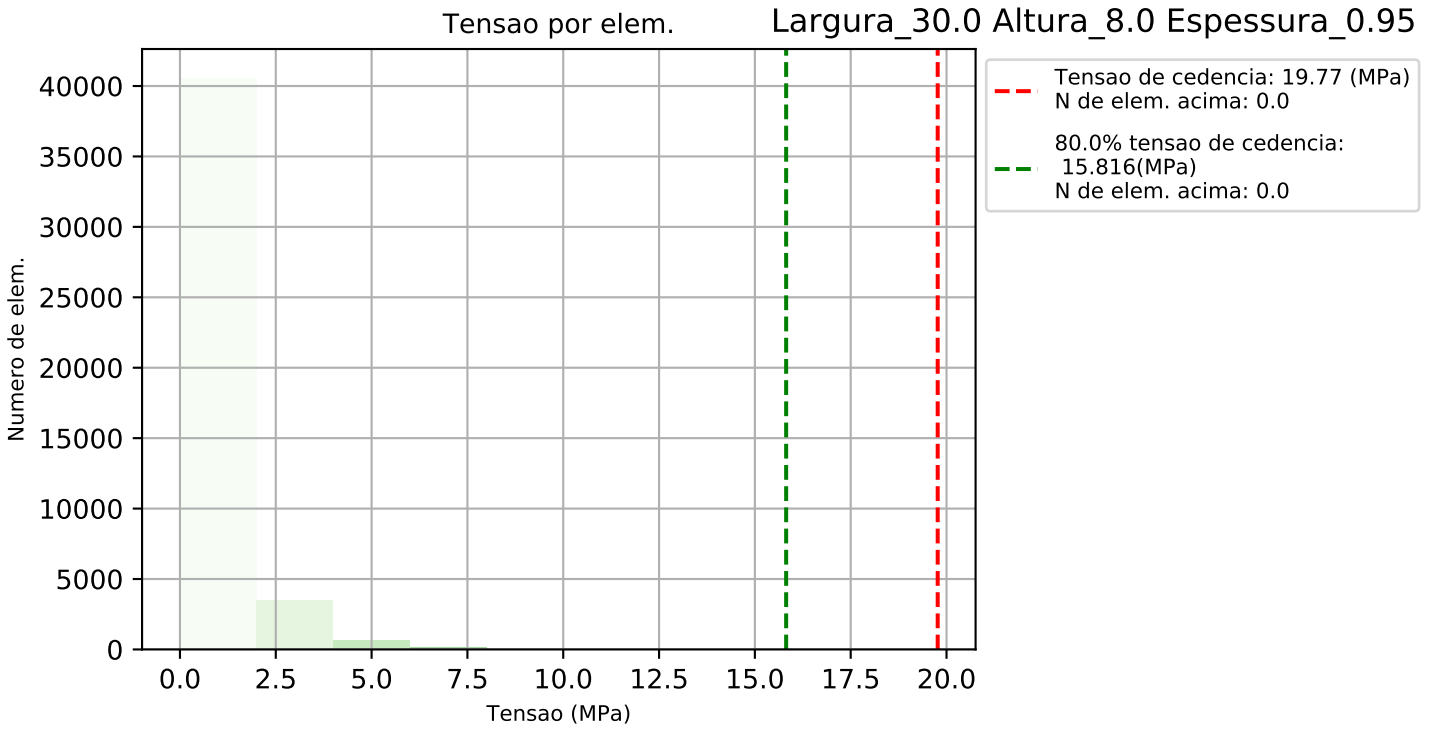

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.321e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.618e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.736e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.500e+00 (mm)
t = 2.000e+00 (s)

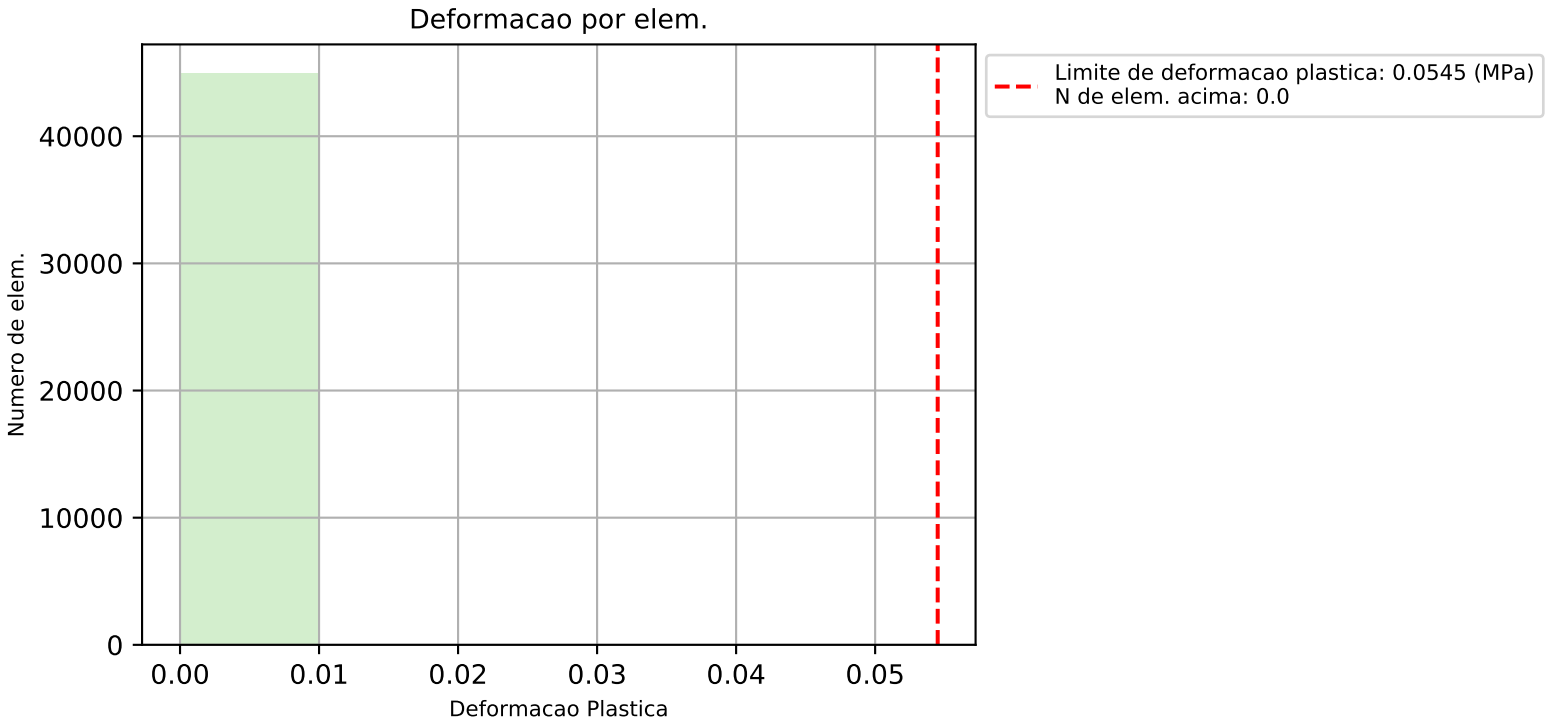

Tensao por elem.
Largura_10.0 Altura_2.0 Espessura_2.5
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

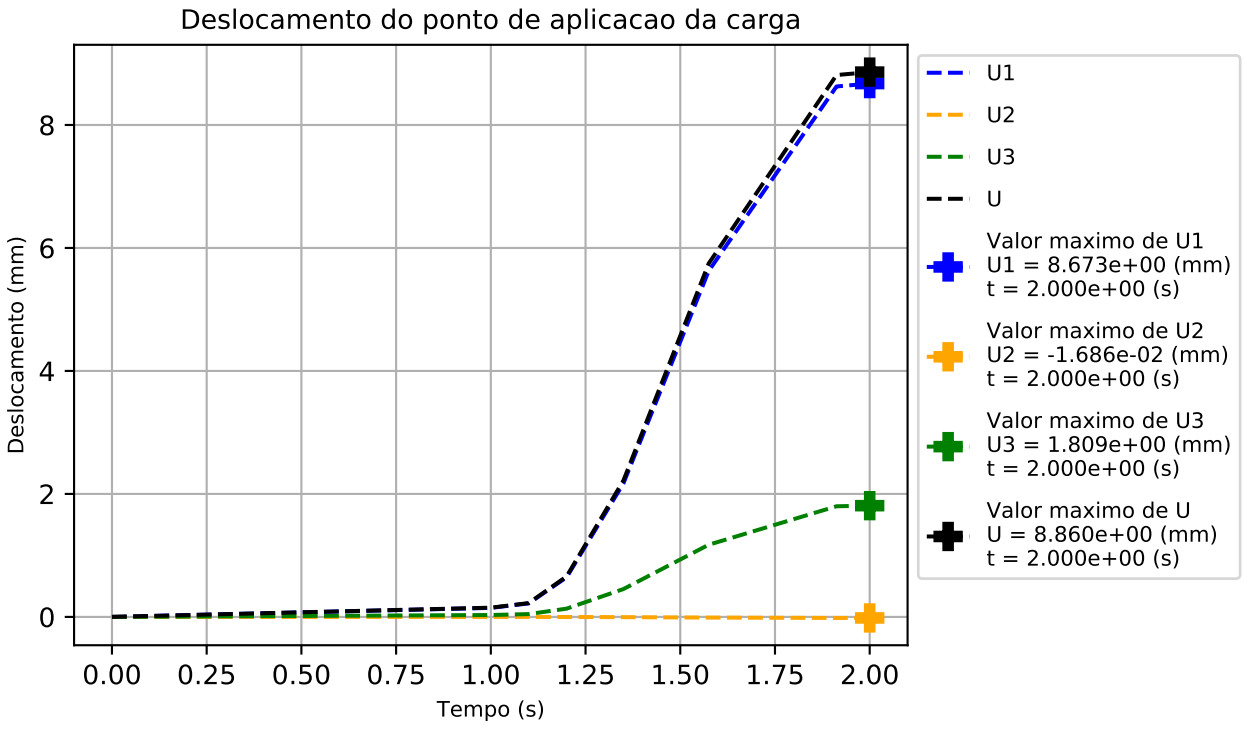

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

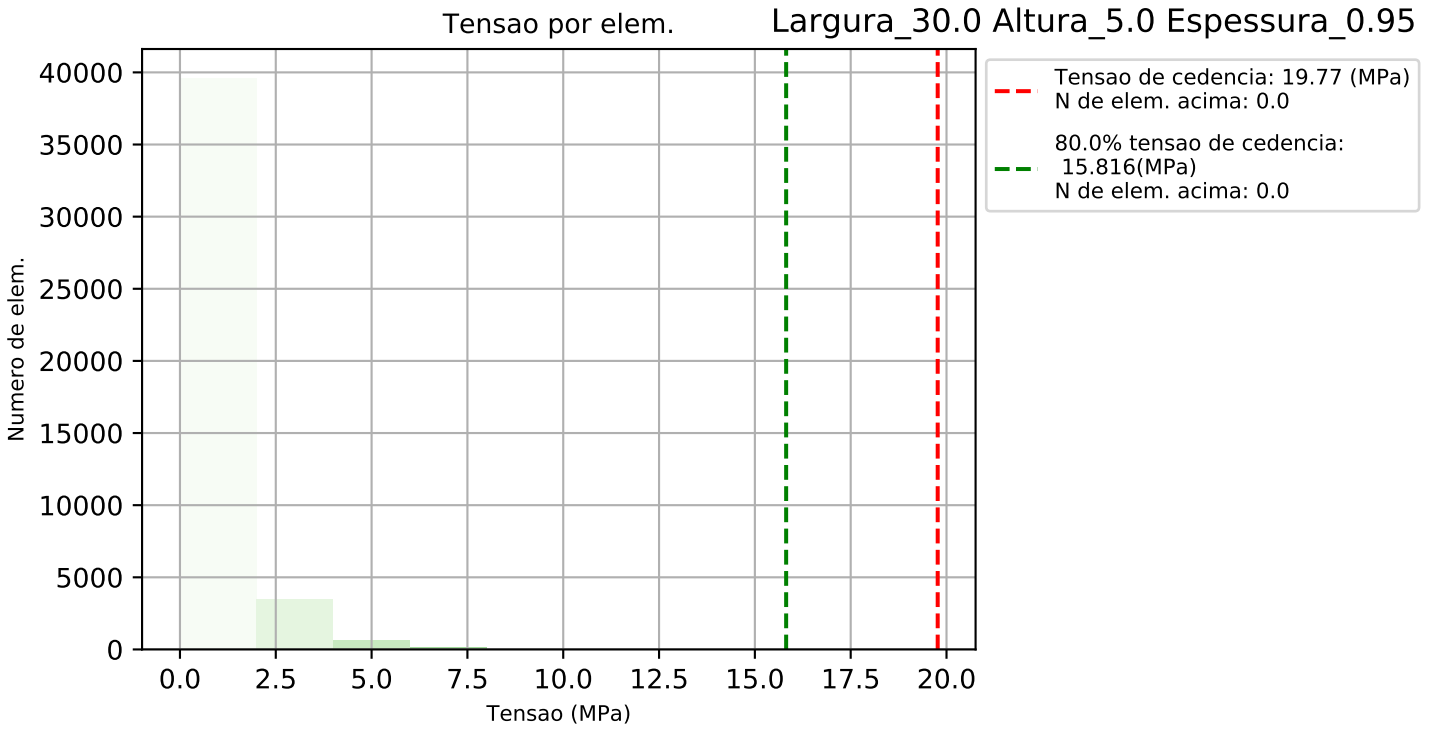

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.467e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.646e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.766e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.650e+00 (mm)
t = 2.000e+00 (s)

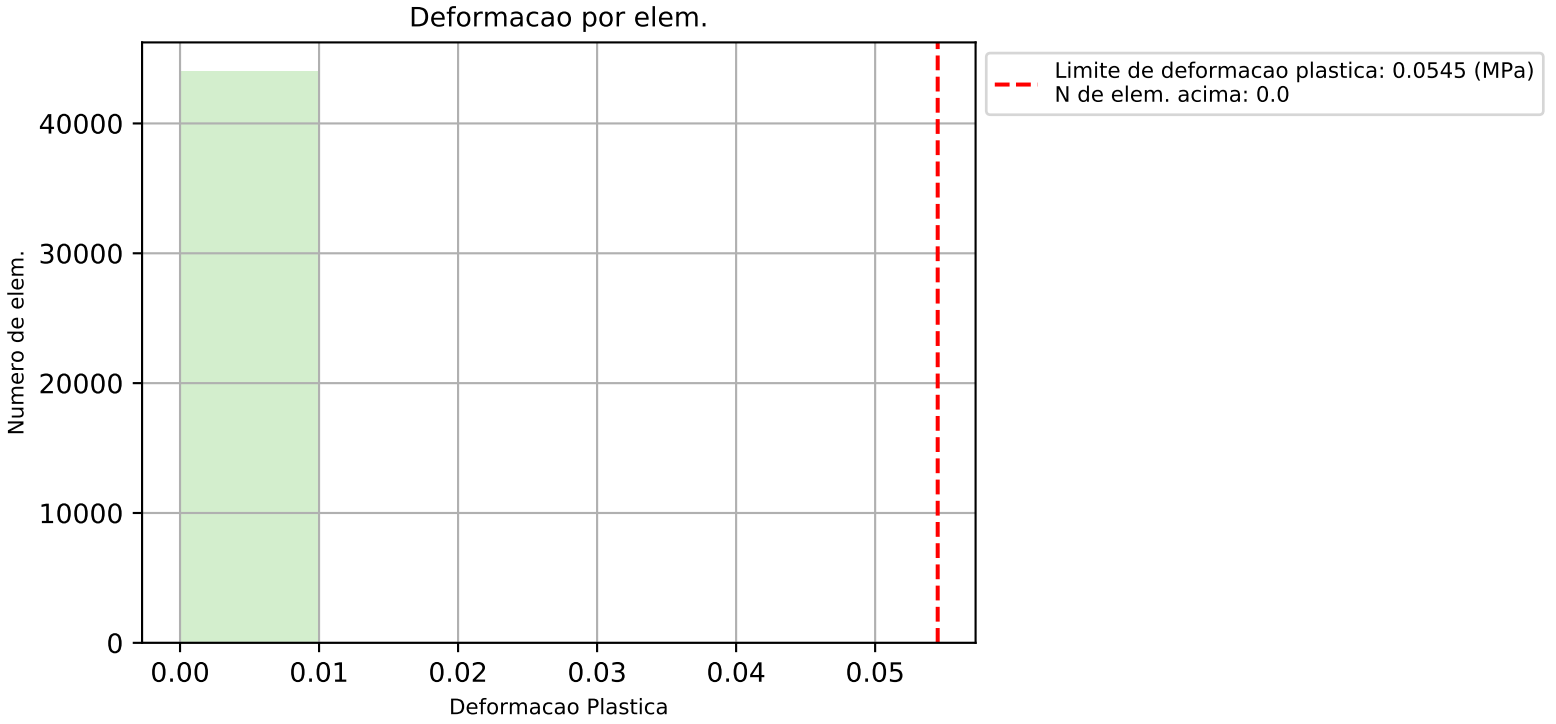
6.2 Livro de resultados do segundo estudo dos frisos

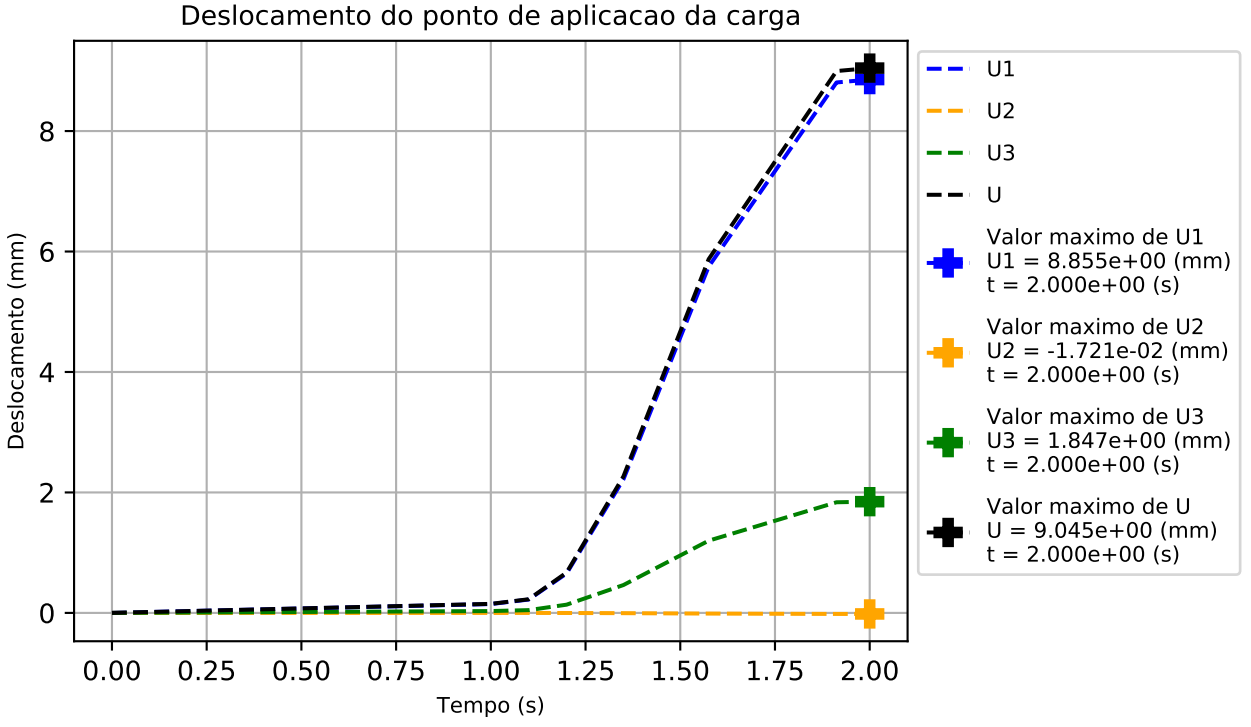

Tensao por elem.
Largura_30.0 Altura_8.0 Espessura_0.95
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

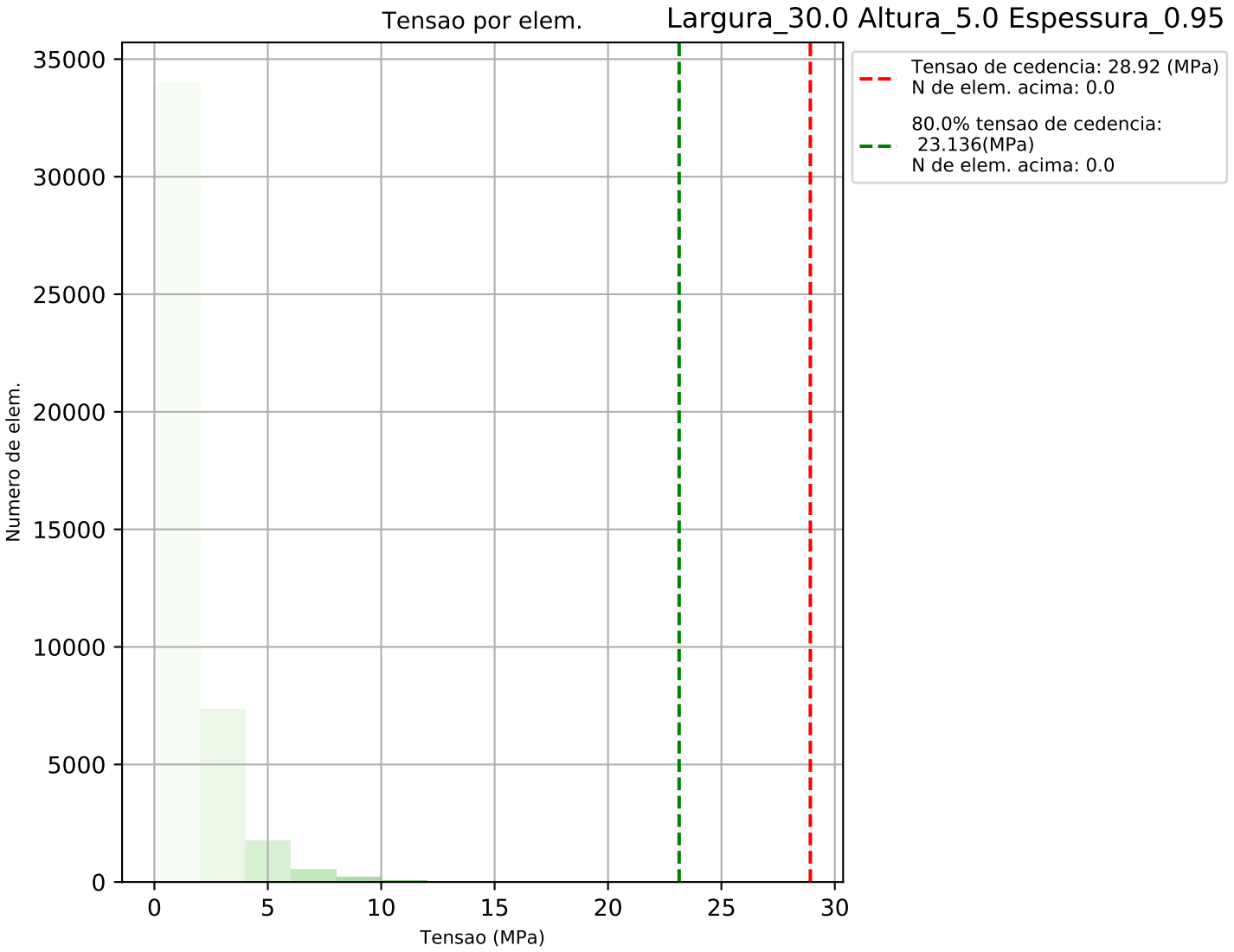

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

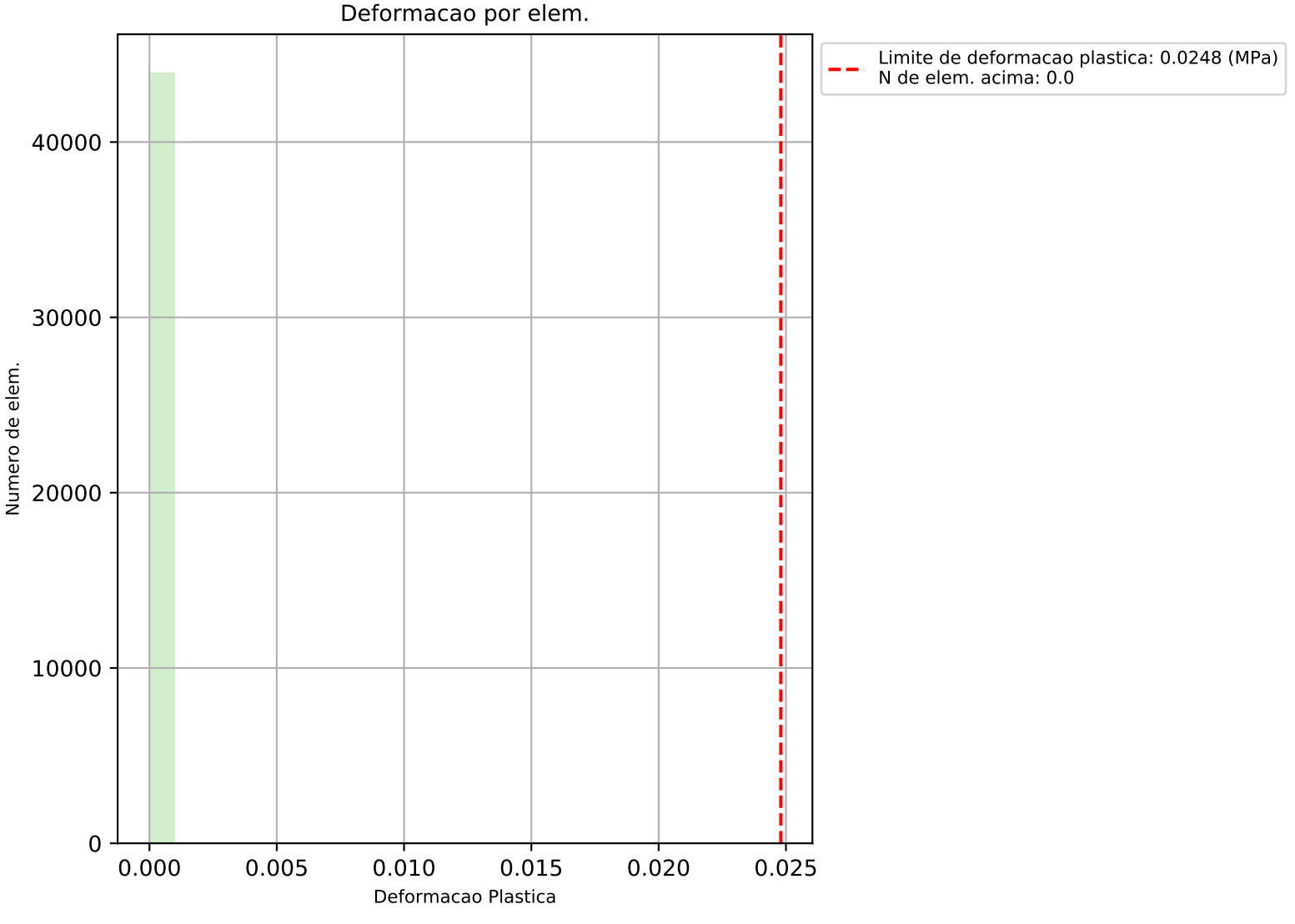

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.673e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.686e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.809e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.860e+00 (mm)
t = 2.000e+00 (s)

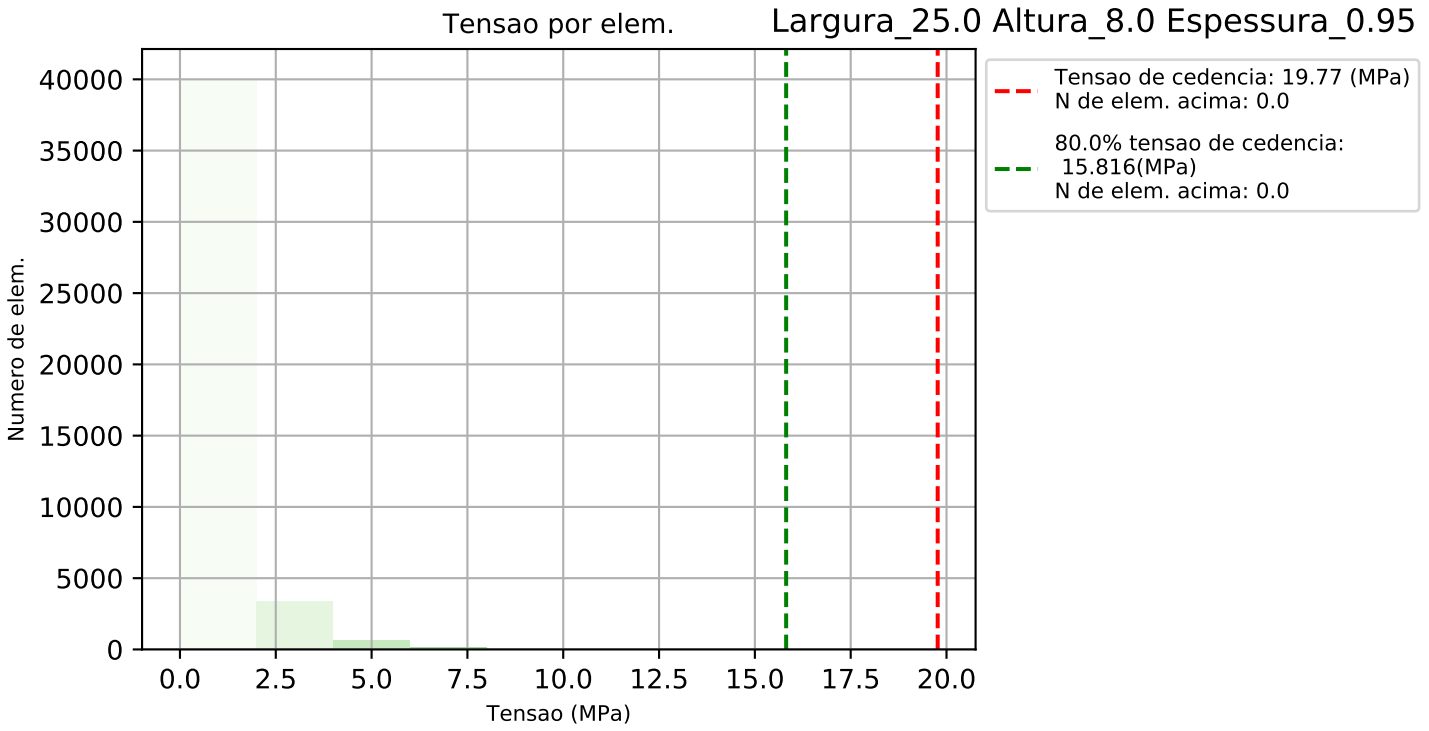

Tensao por elem.
Largura_30.0 Altura_5.0 Espessura_0.95
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

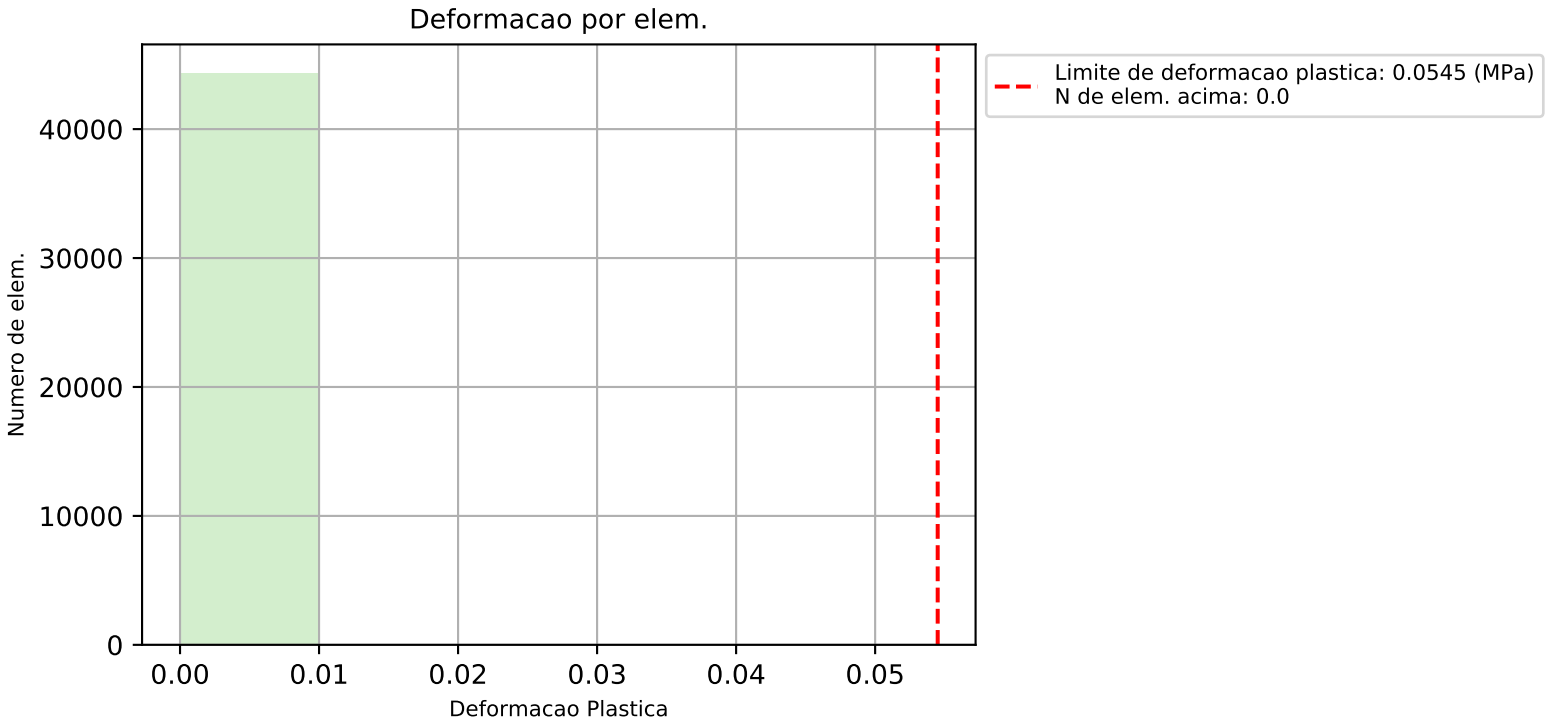

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

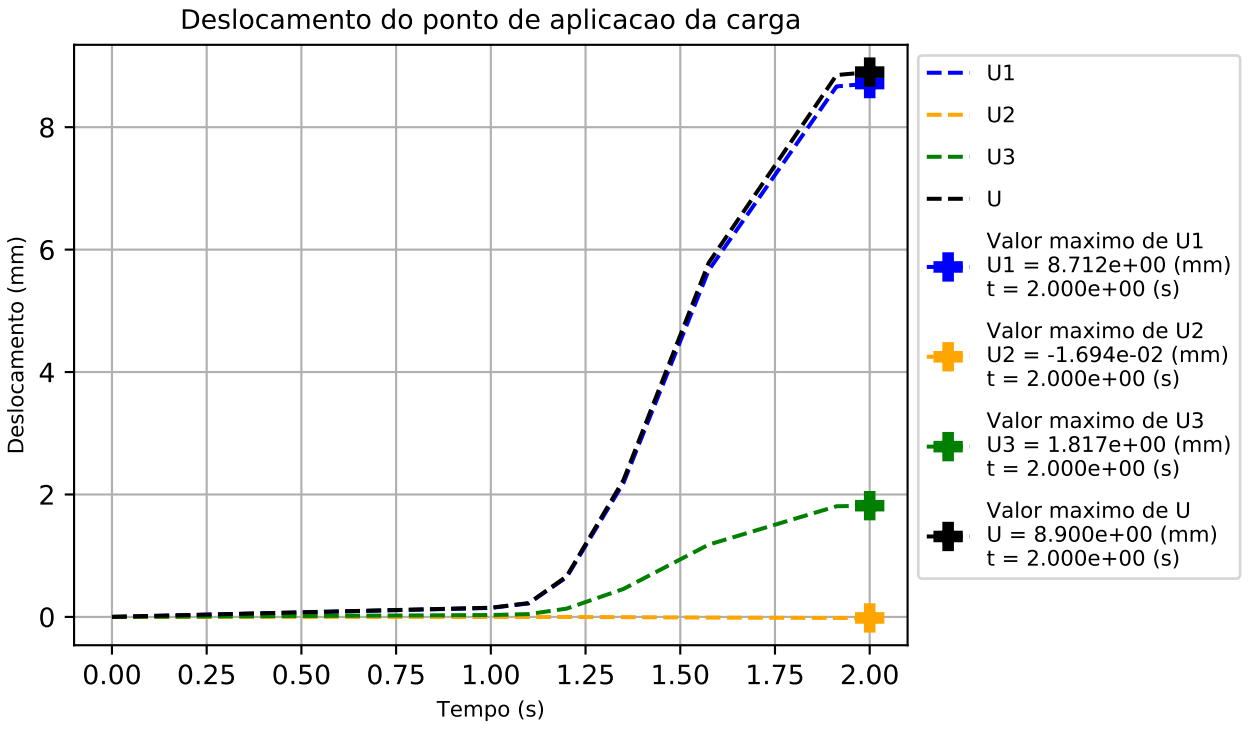

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.855e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.721e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.847e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 9.045e+00 (mm)
t = 2.000e+00 (s)

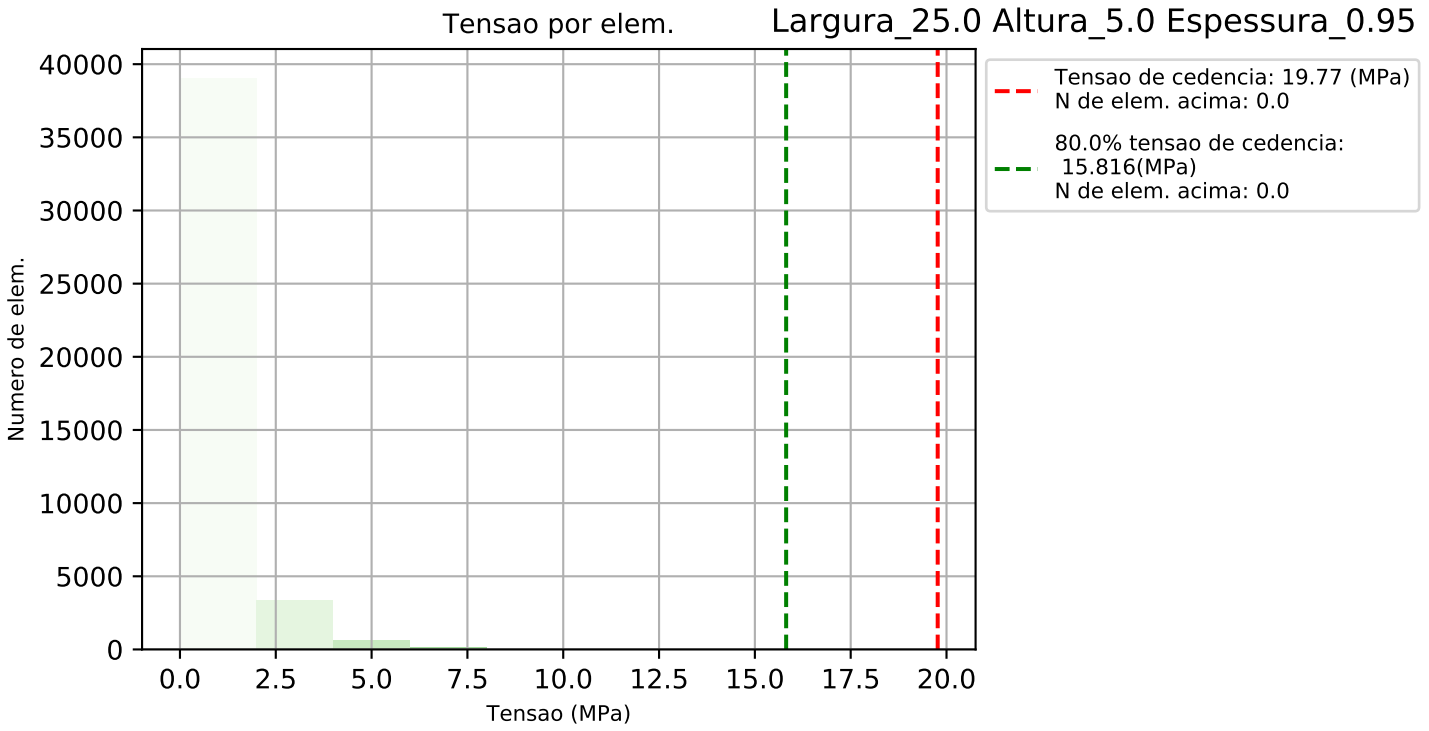

Tensao por elem.
Largura_30.0 Altura_5.0 Espessura_0.95
Tensao de cedencia: 28.92 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
23.136(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)
0
5000
10000
15000
20000
25000
30000
35000
0
5
10
15
20
25
30

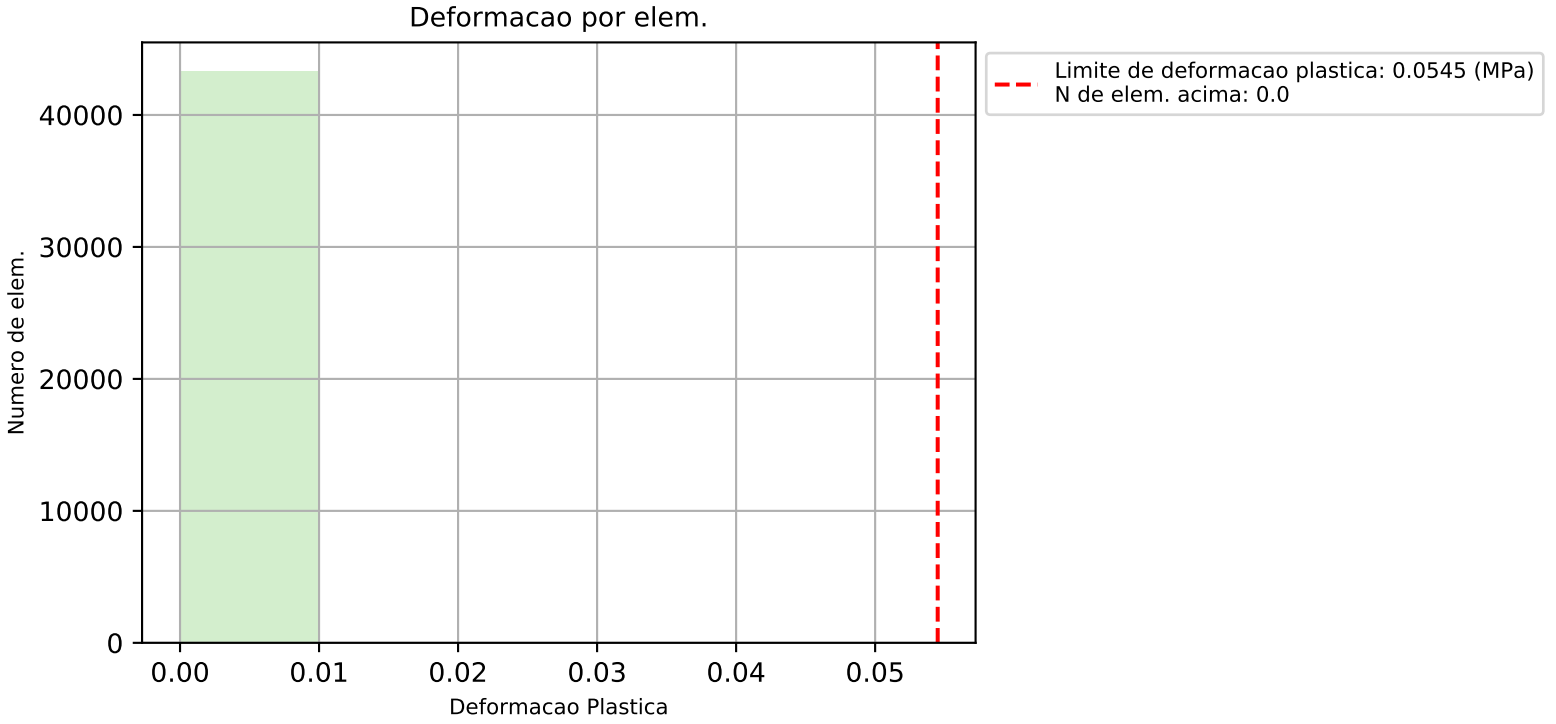

Deformacao por elem.
Limite de deformacao plastica: 0.0248 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica
0
10000
20000
30000
40000
0.000
0.005
0.010
0.015
0.020
0.025

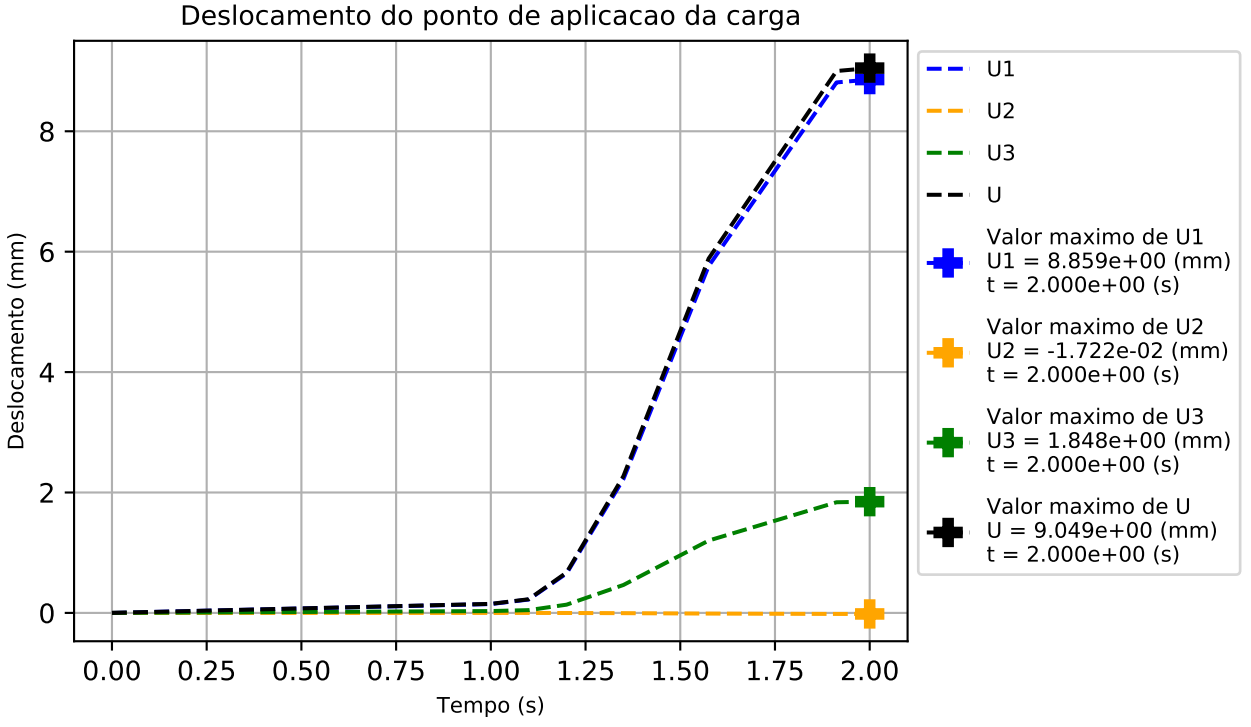

Tensao por elem.
Largura_25.0 Altura_8.0 Espessura_0.95
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

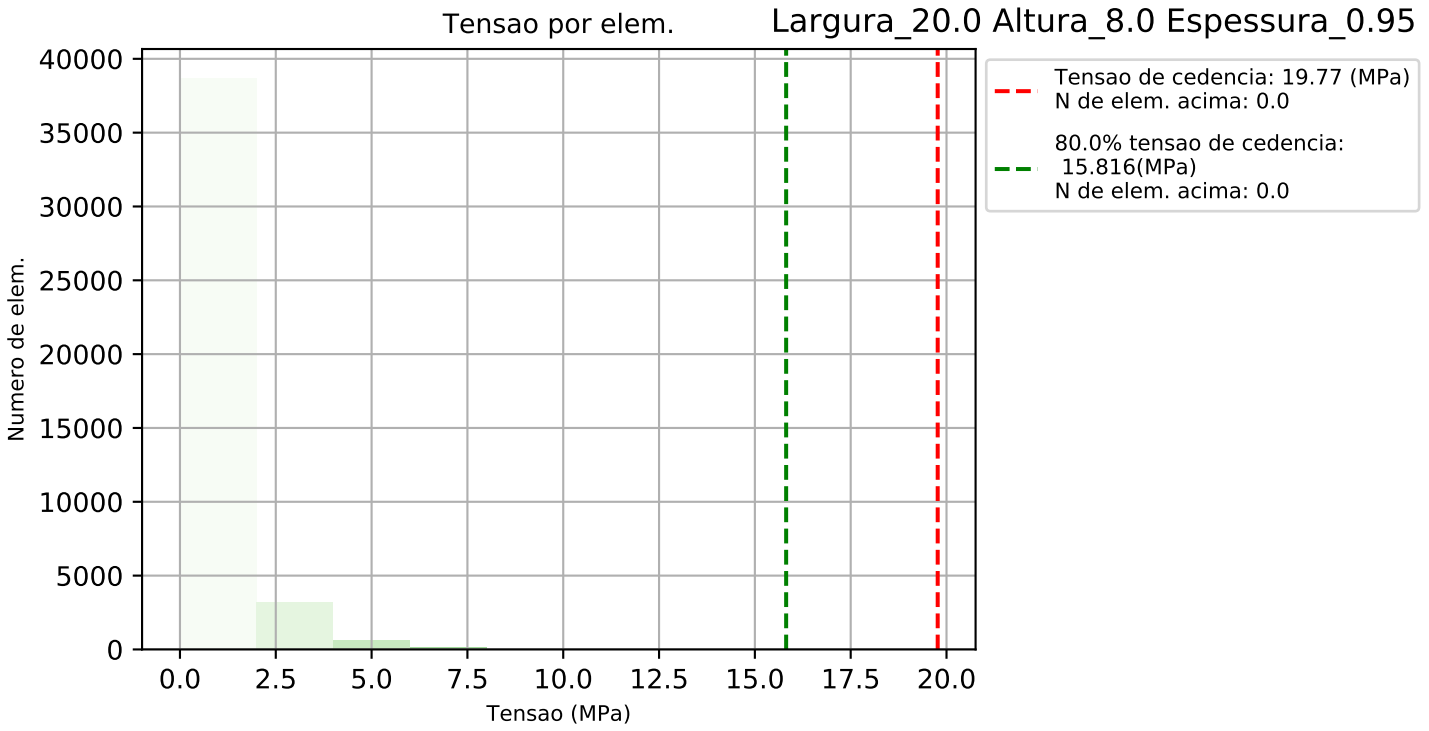

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

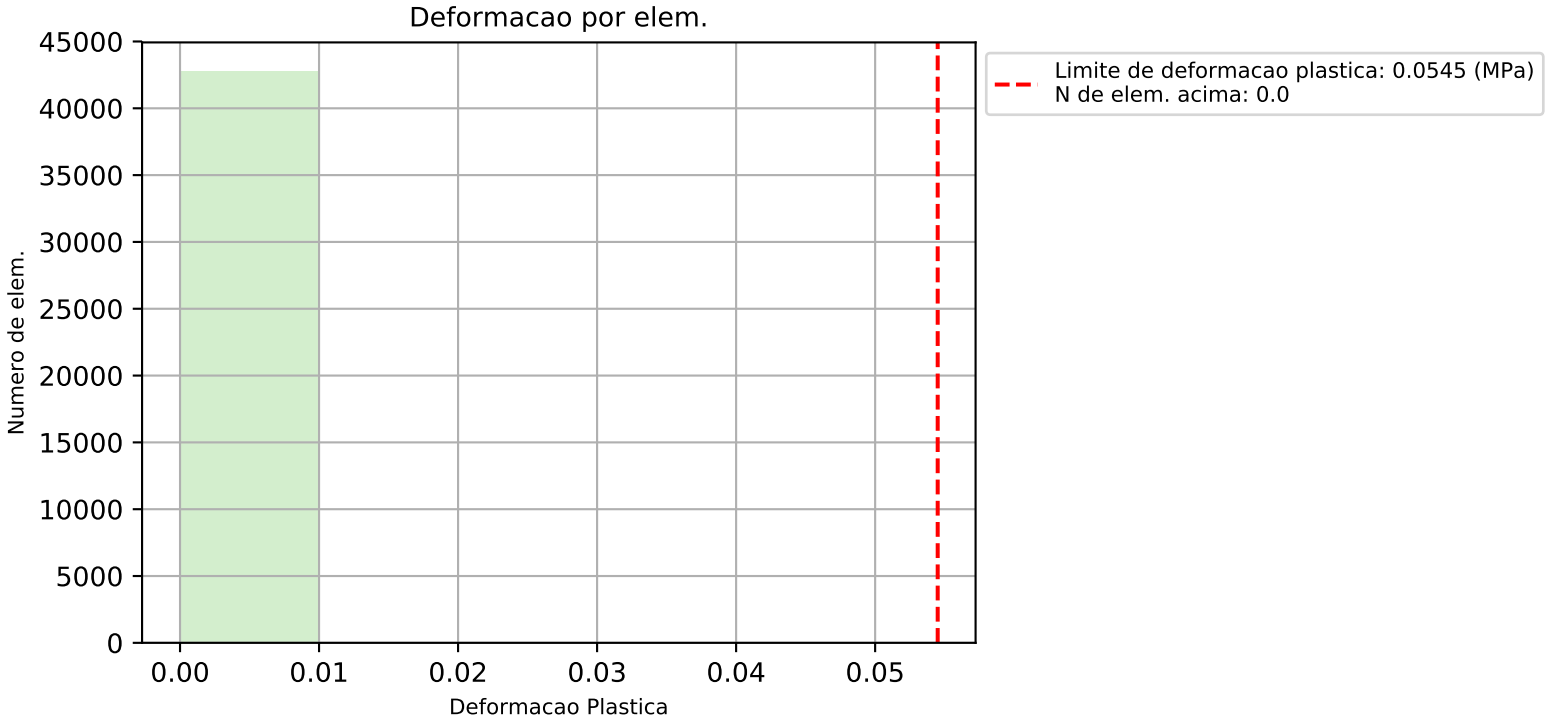

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.712e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.694e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.817e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.900e+00 (mm)
t = 2.000e+00 (s)

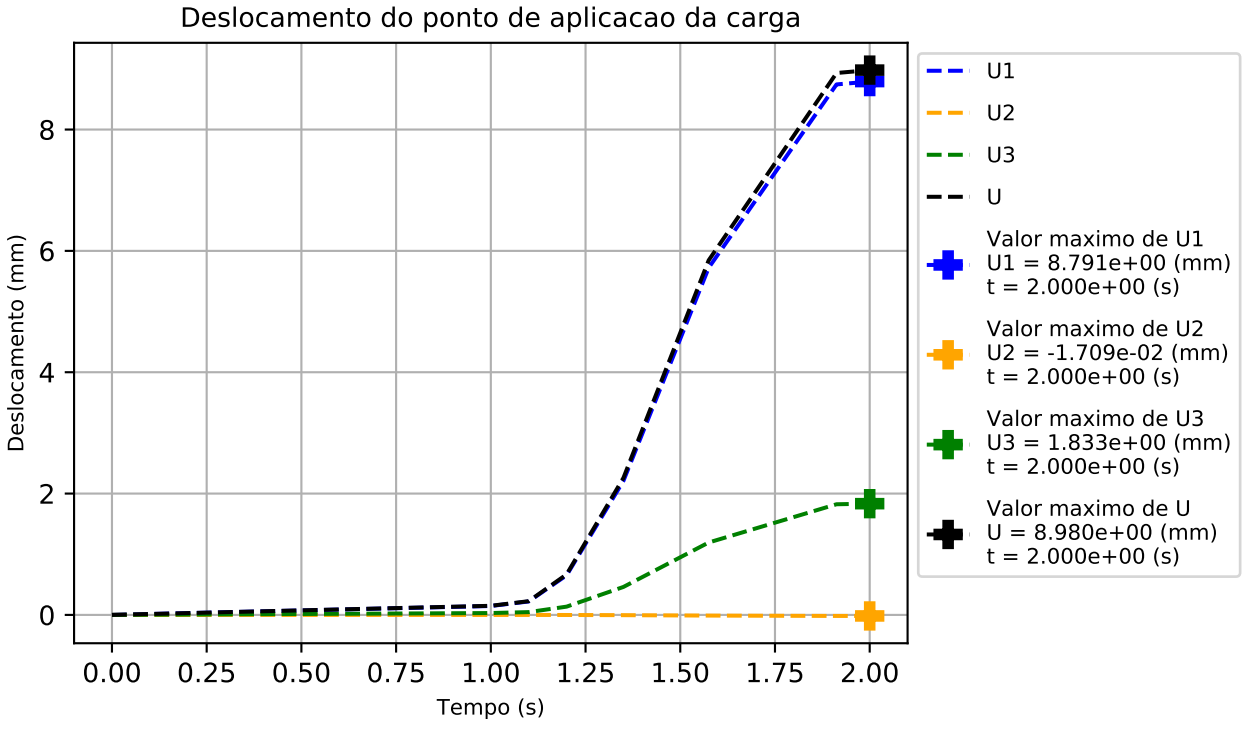

Tensao por elem.
Largura_25.0 Altura_5.0 Espessura_0.95
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

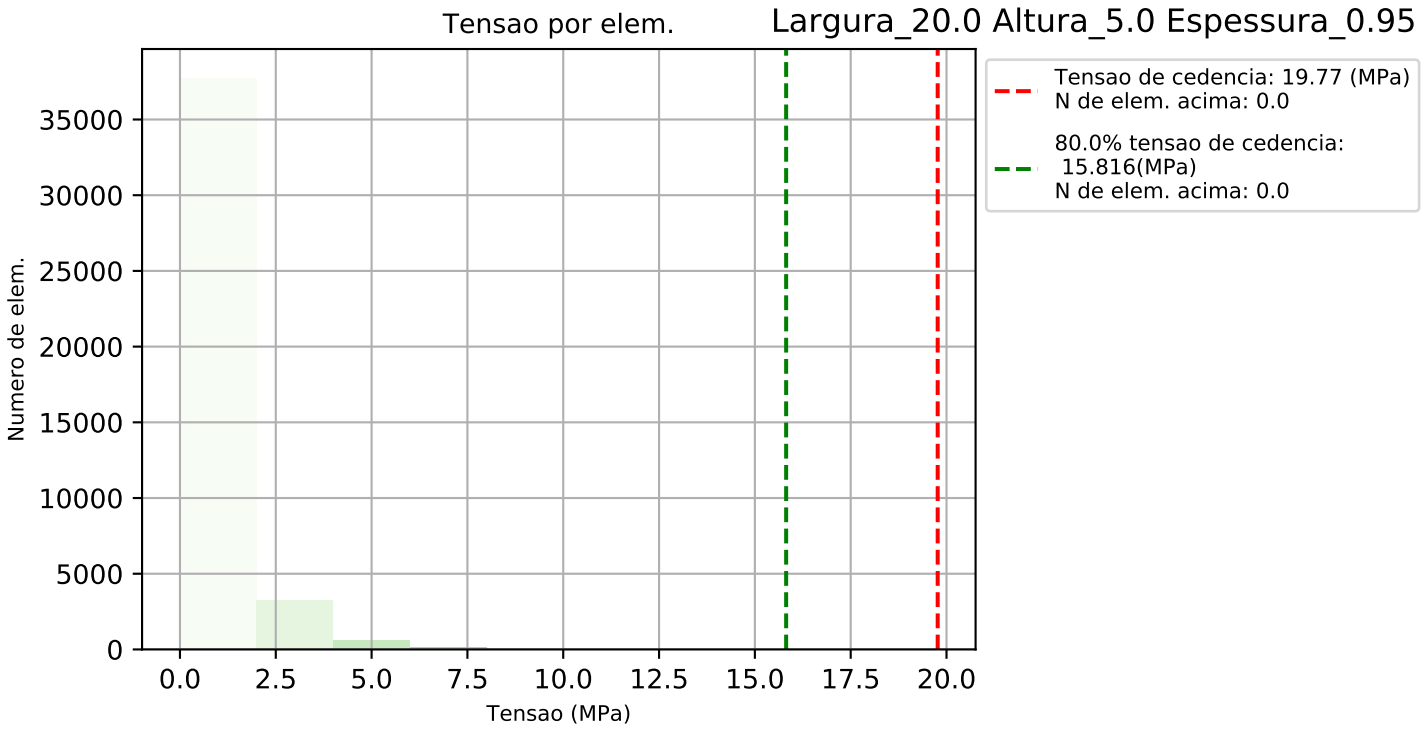

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

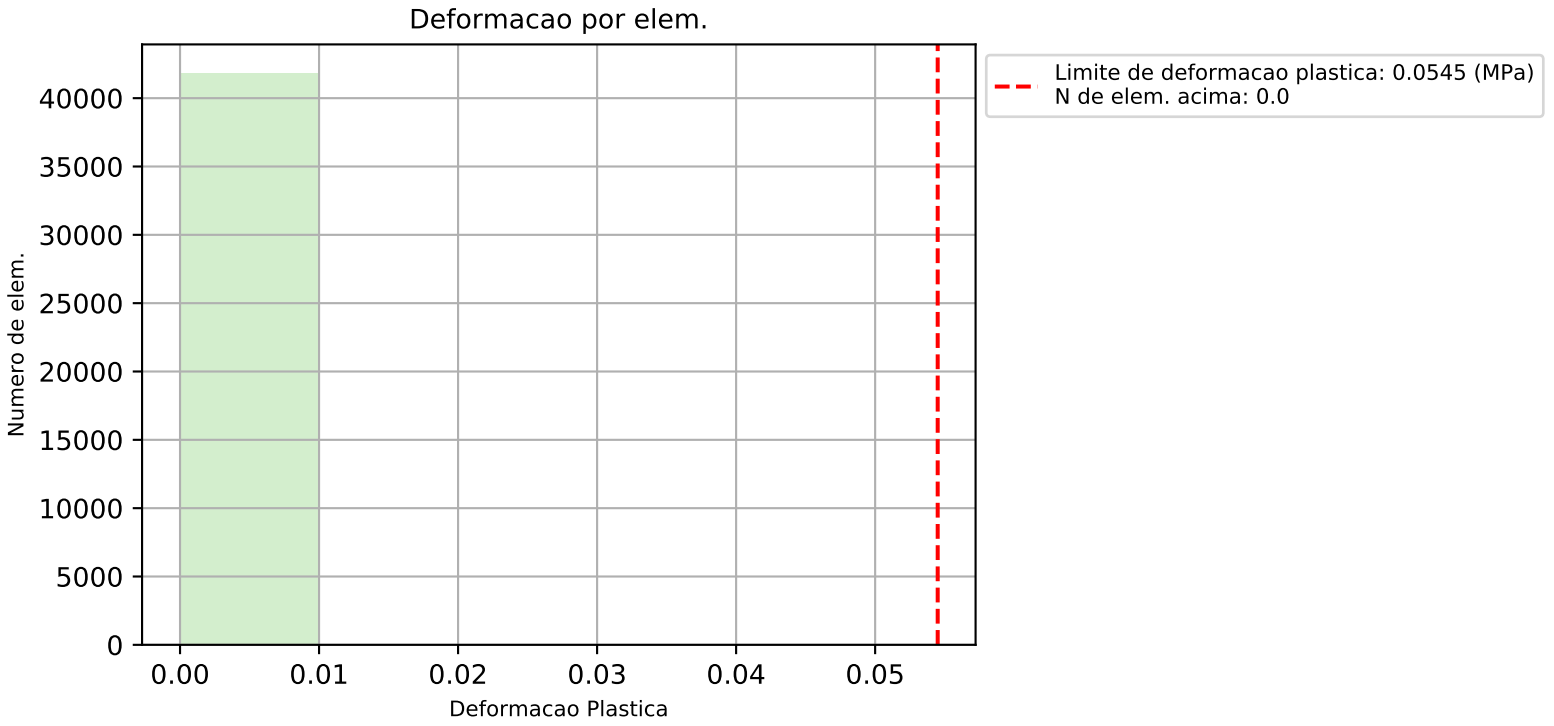

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.859e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.722e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.848e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 9.049e+00 (mm)
t = 2.000e+00 (s)

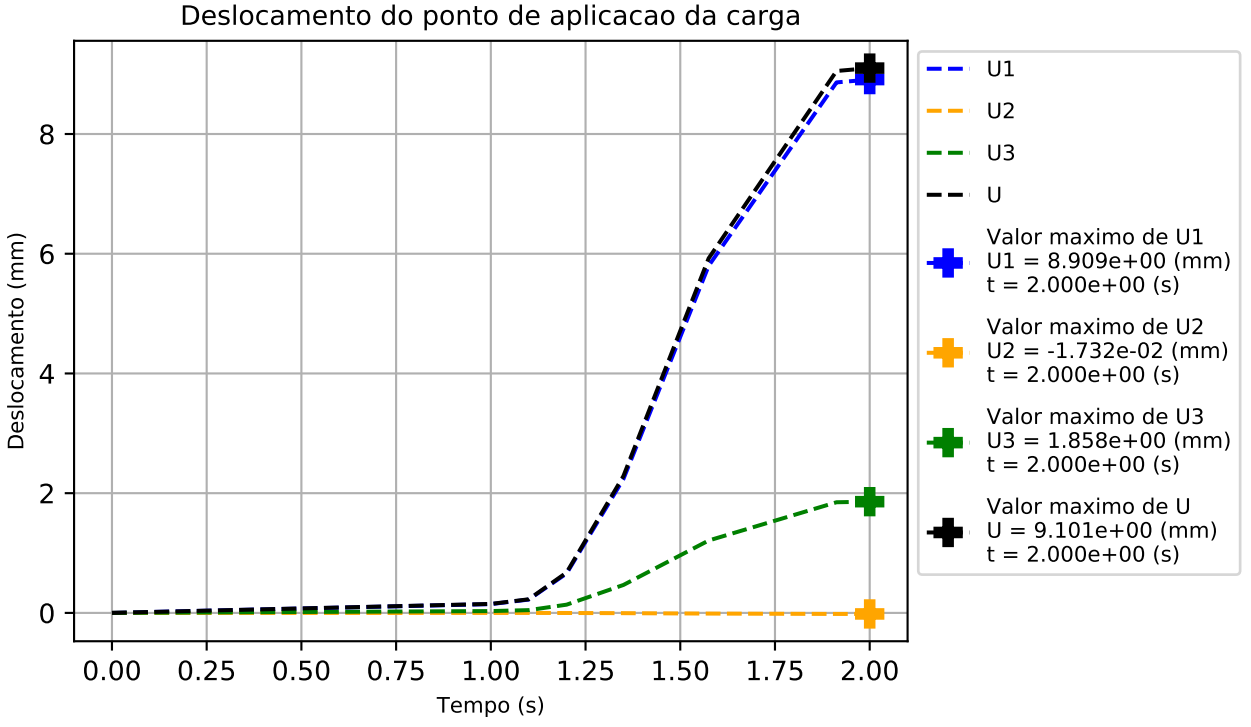

Tensao por elem.
Largura_20.0 Altura_8.0 Espessura_0.95
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

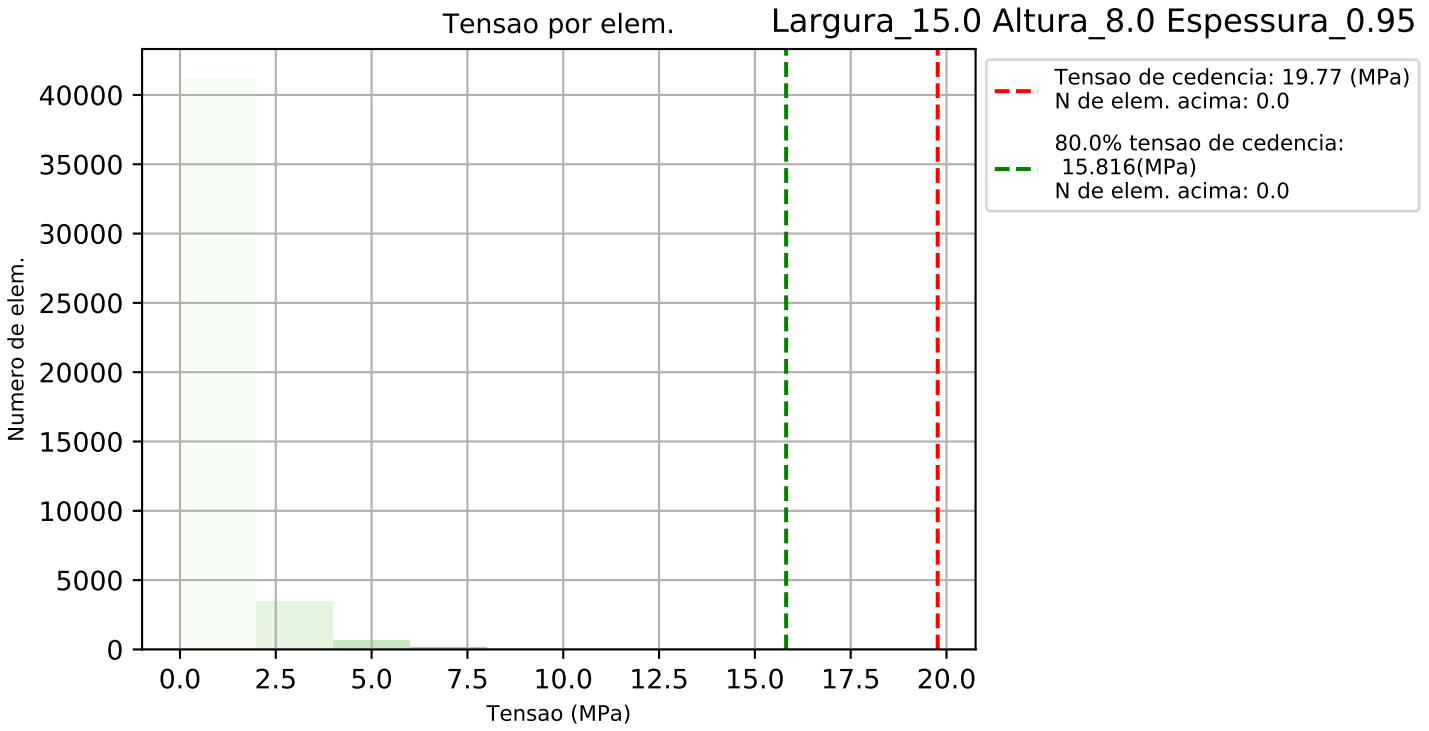

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

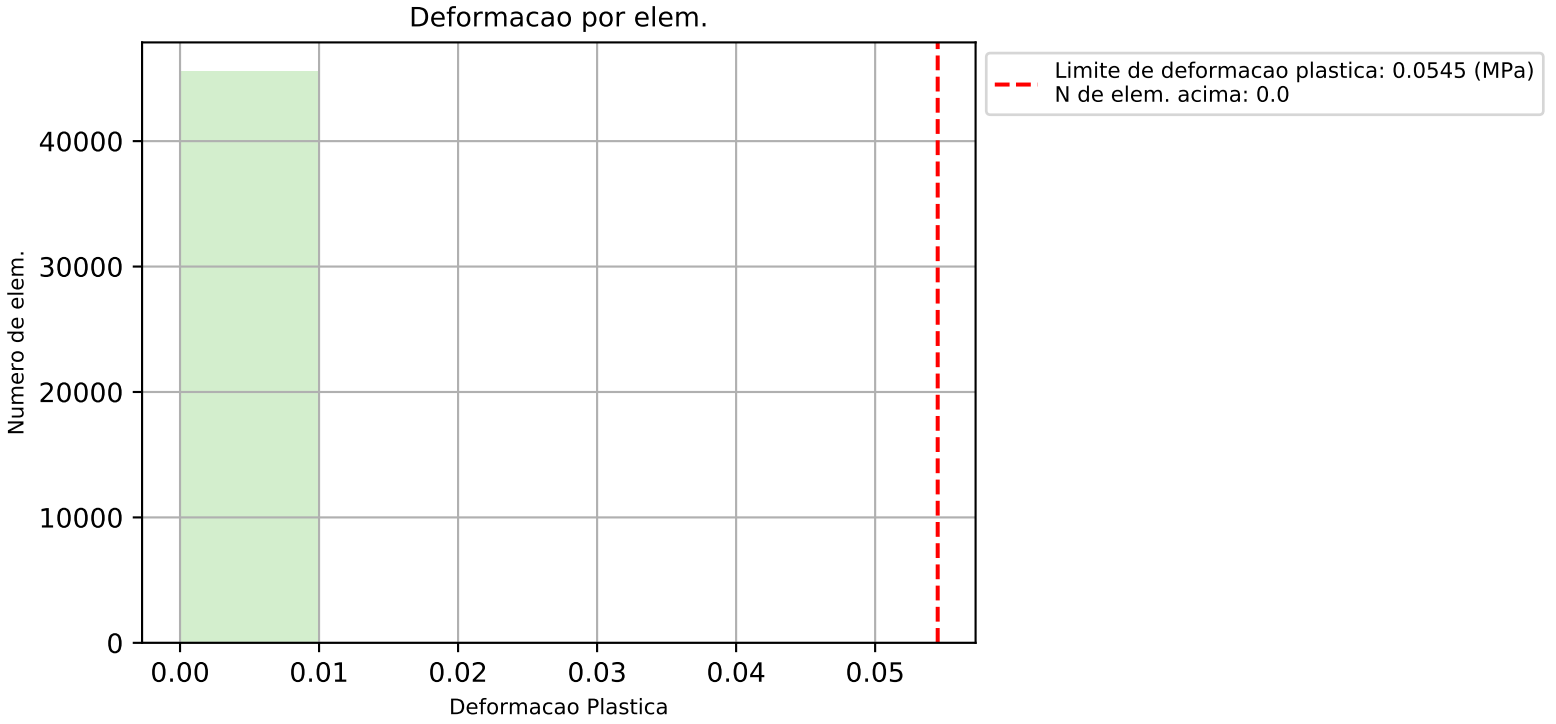

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.791e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.709e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.833e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.980e+00 (mm)
t = 2.000e+00 (s)

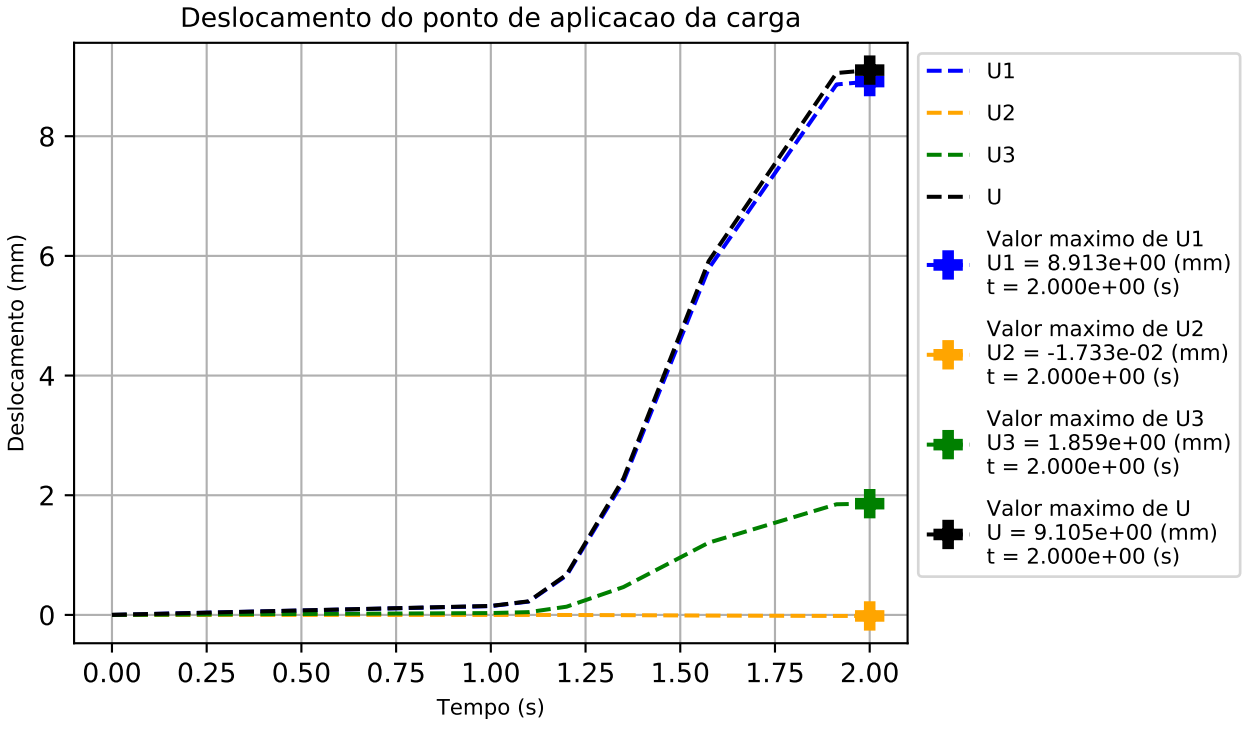

Tensao por elem.
Largura_20.0 Altura_5.0 Espessura_0.95
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)

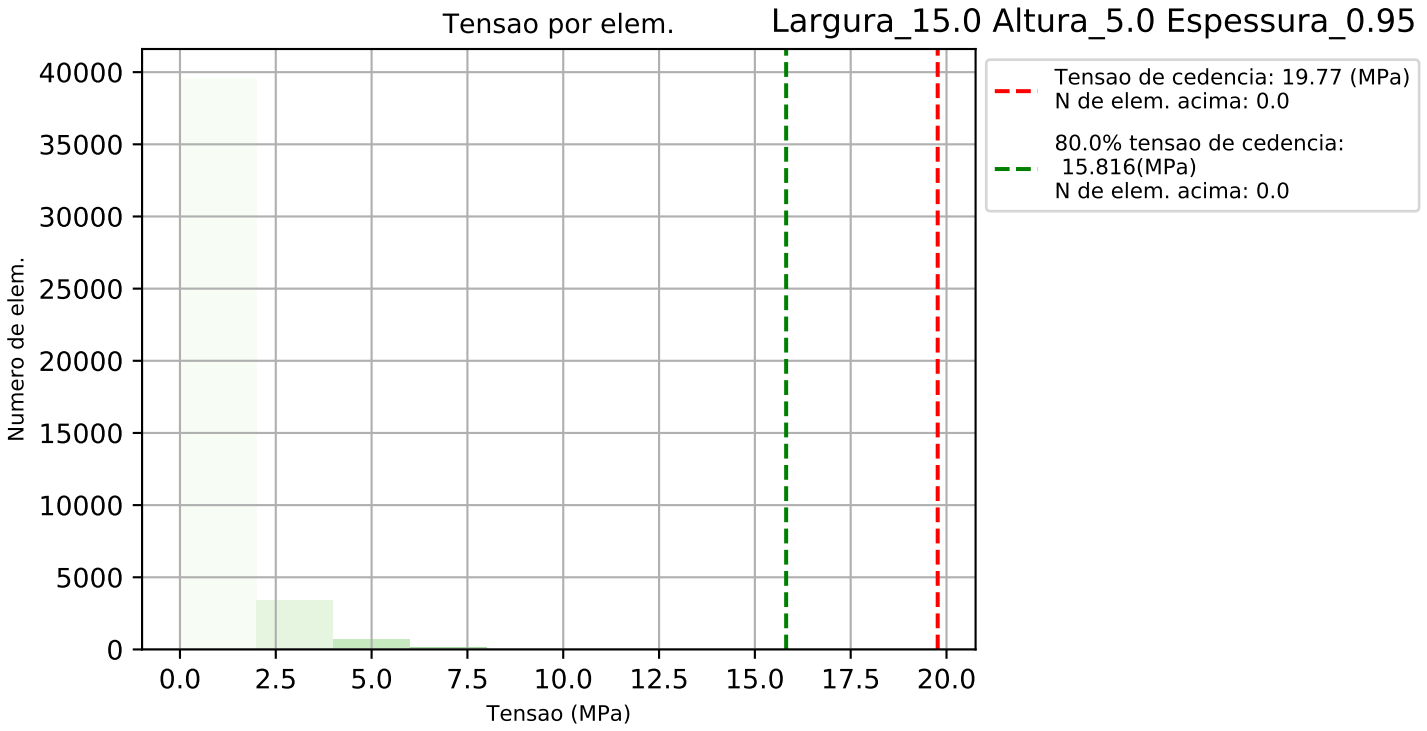

Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica

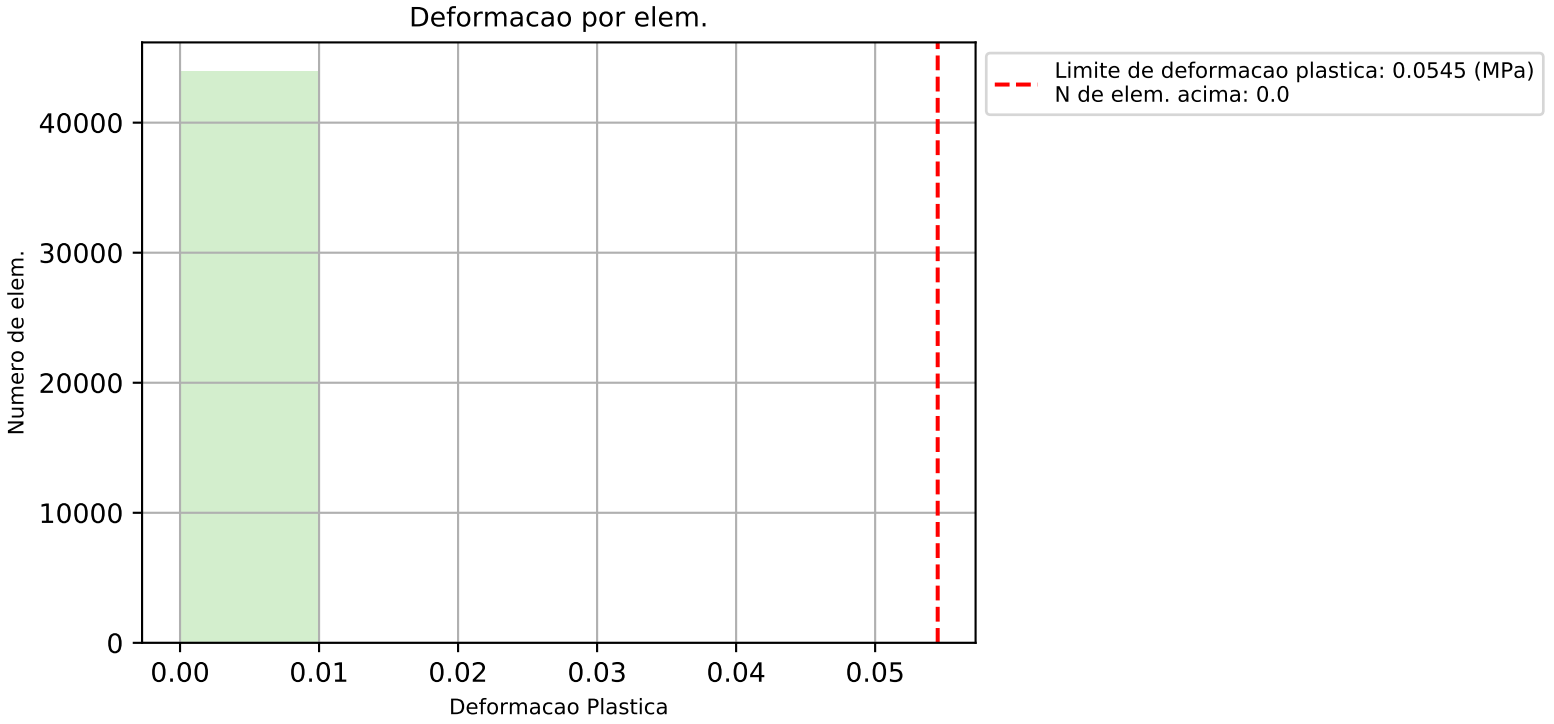

Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 8.909e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.732e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.858e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 9.101e+00 (mm)
t = 2.000e+00 (s)
Deslocamento (mm)
Tempo (s)

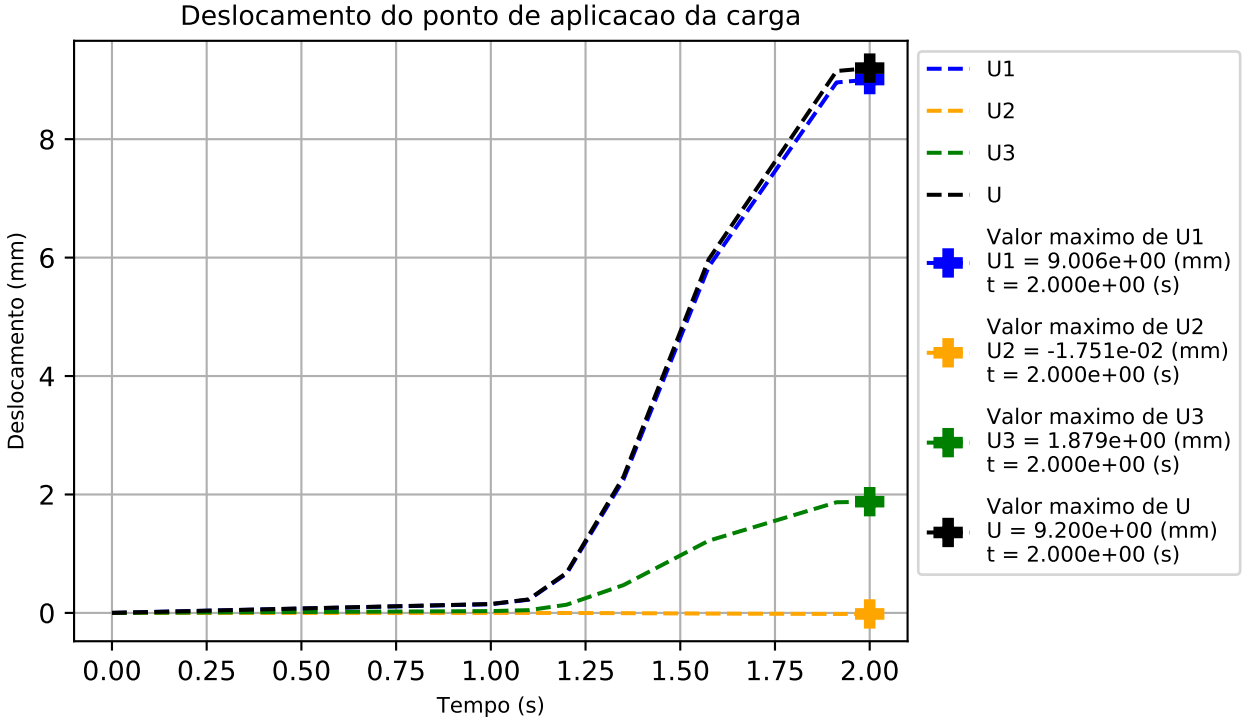

Tensao por elem.
Largura_15.0 Altura_8.0 Espessura_0.95
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

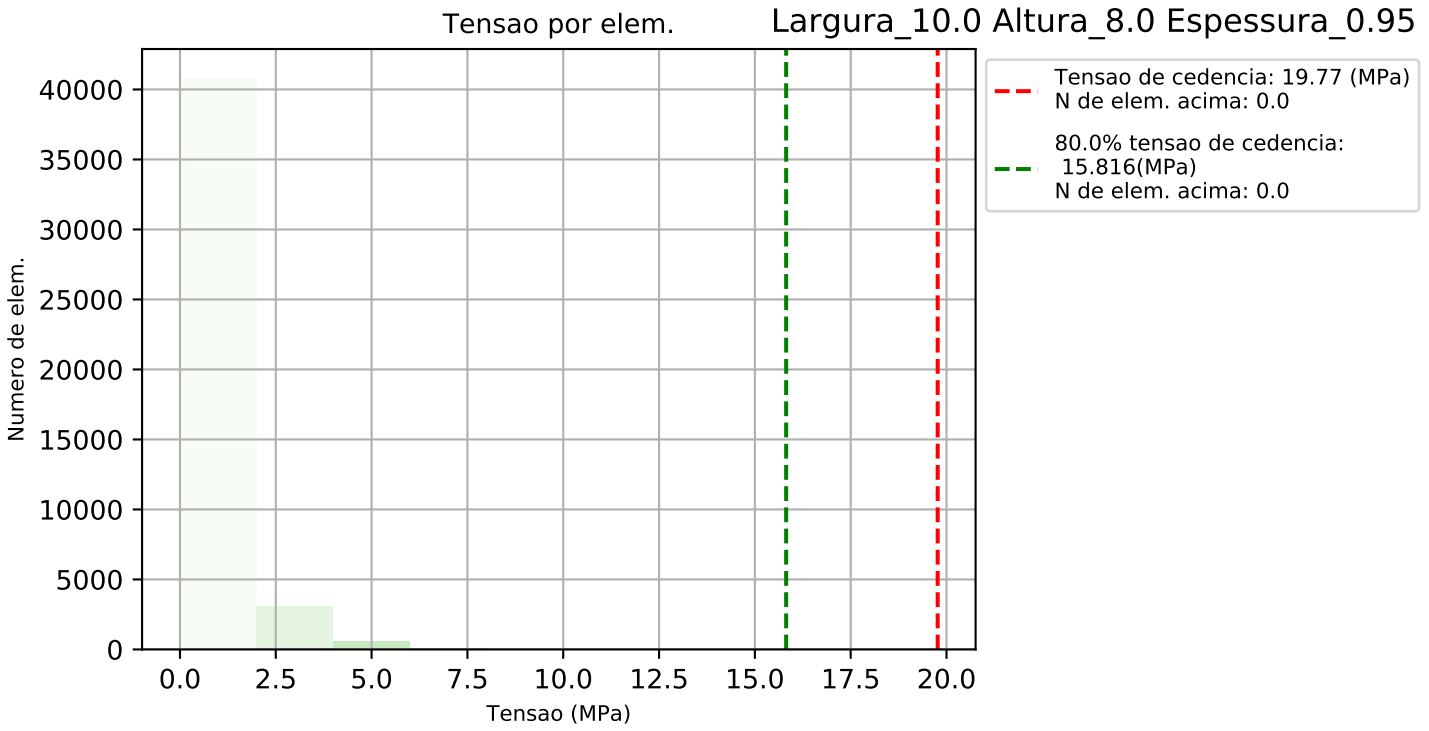

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

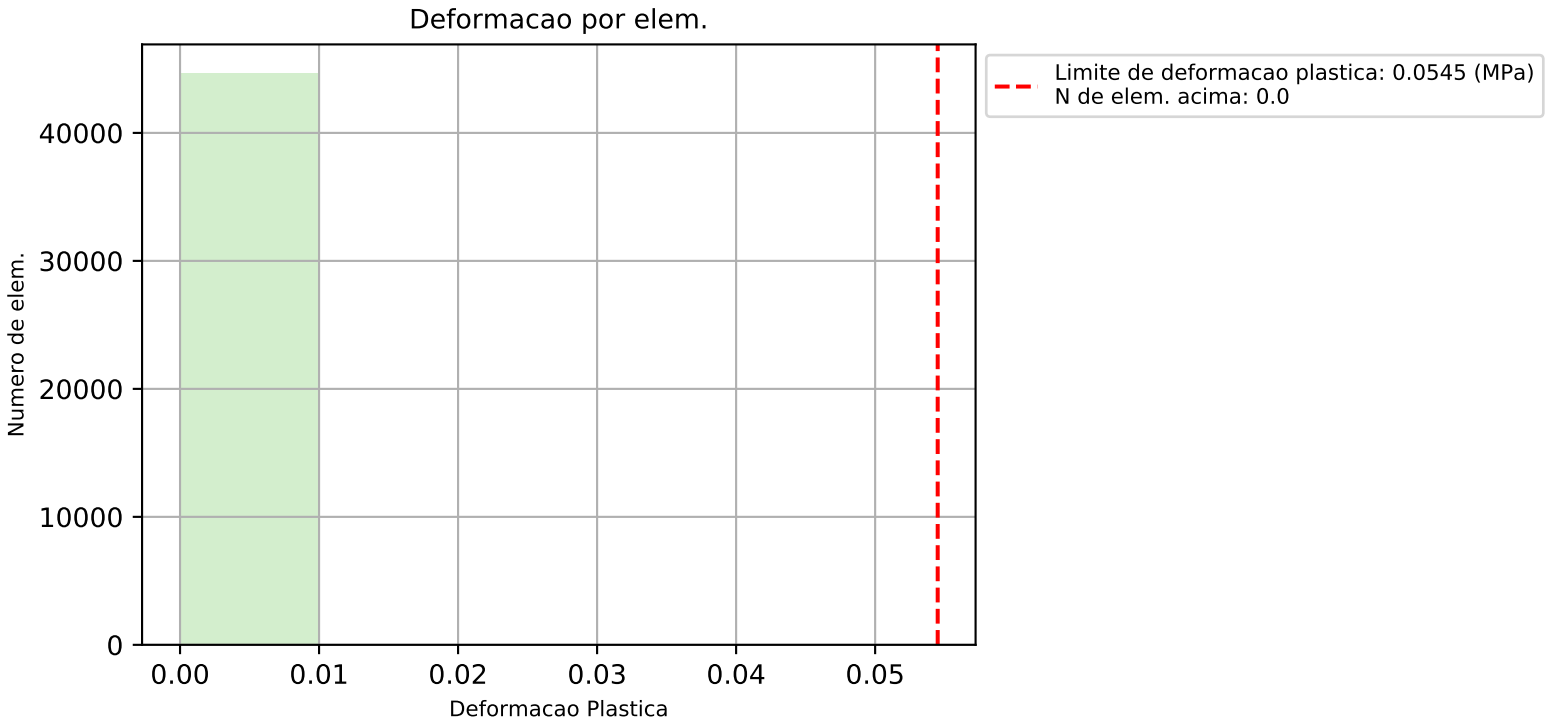

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.913e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.733e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.859e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 9.105e+00 (mm)
t = 2.000e+00 (s)

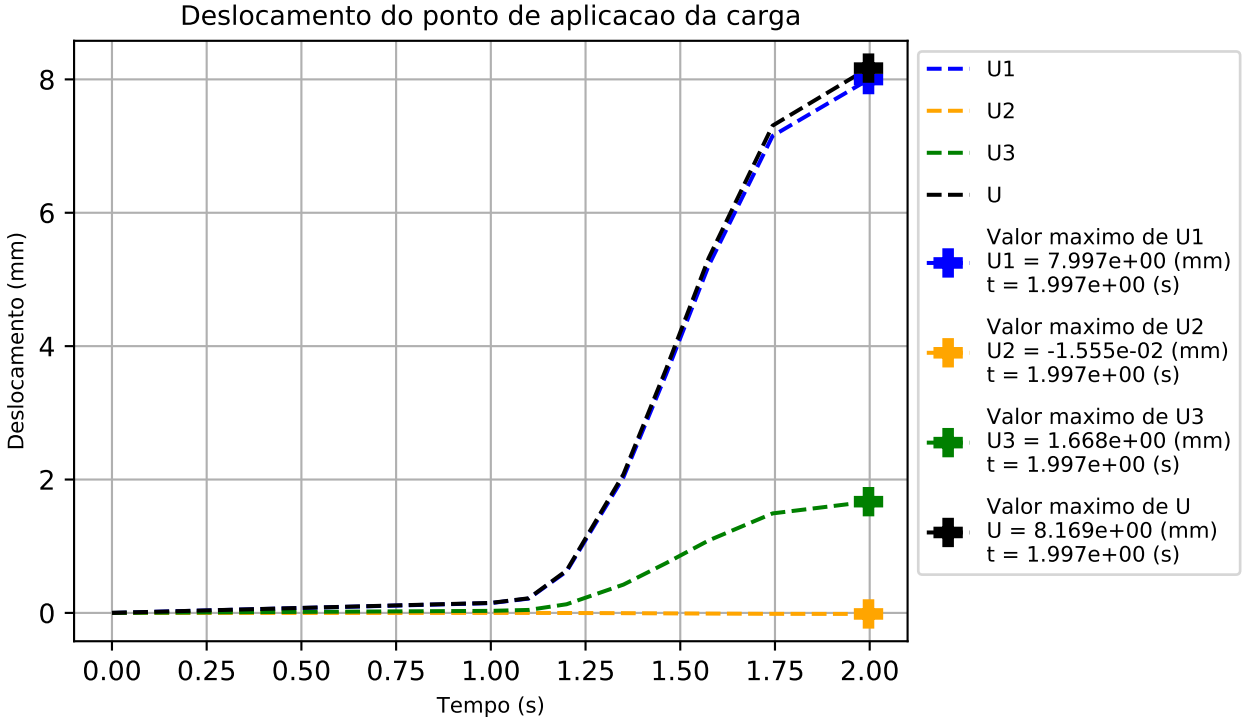

Tensao por elem.
Largura_15.0 Altura_5.0 Espessura_0.95
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0

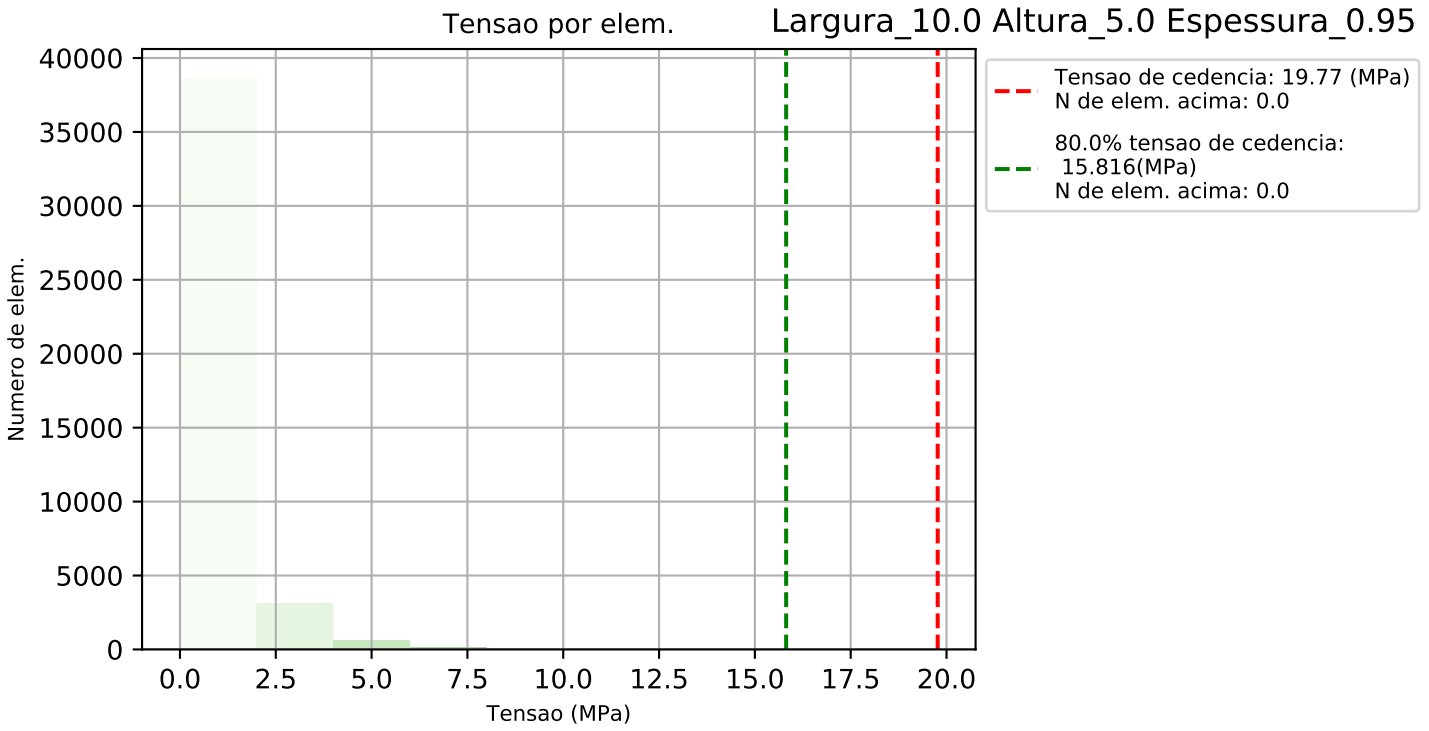

Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0

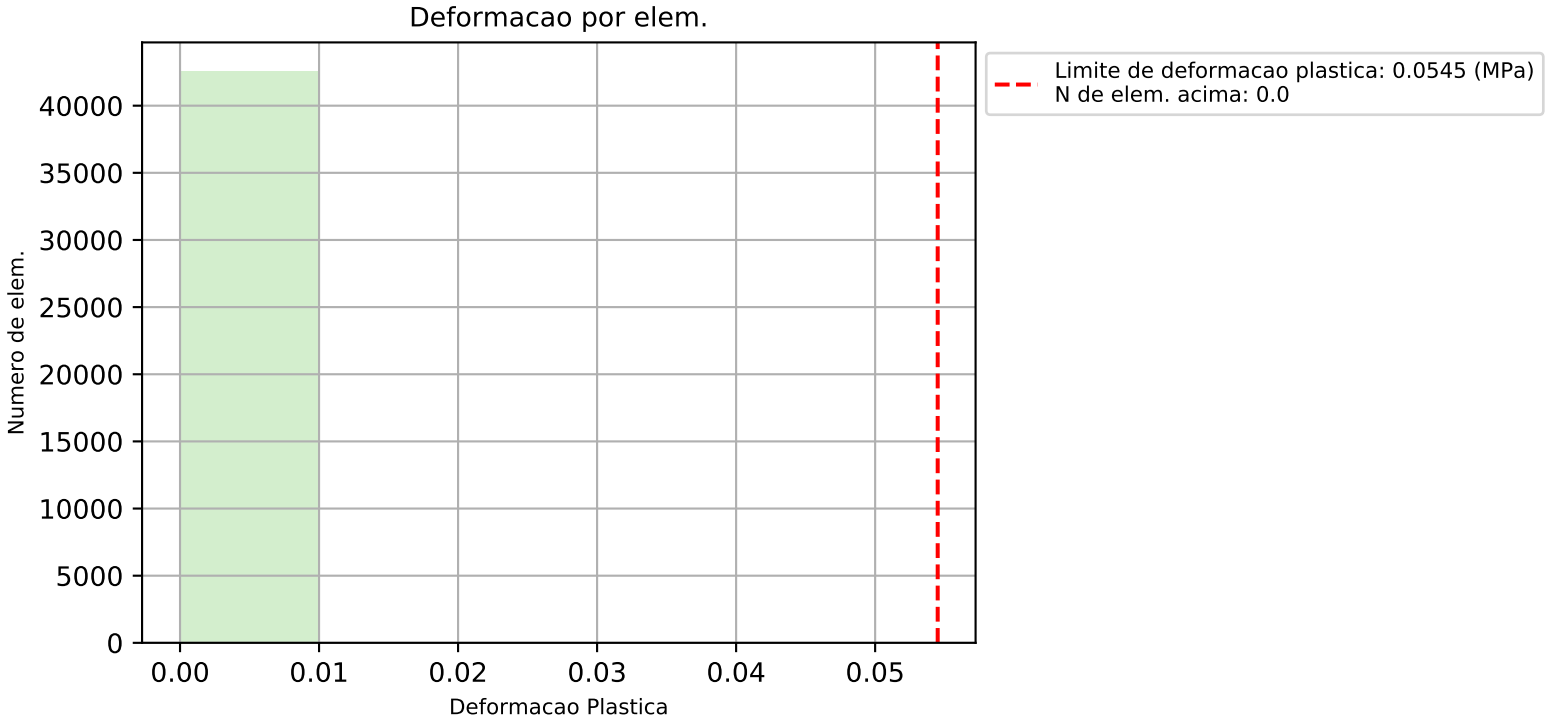

Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 9.006e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.751e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.879e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 9.200e+00 (mm)
t = 2.000e+00 (s)

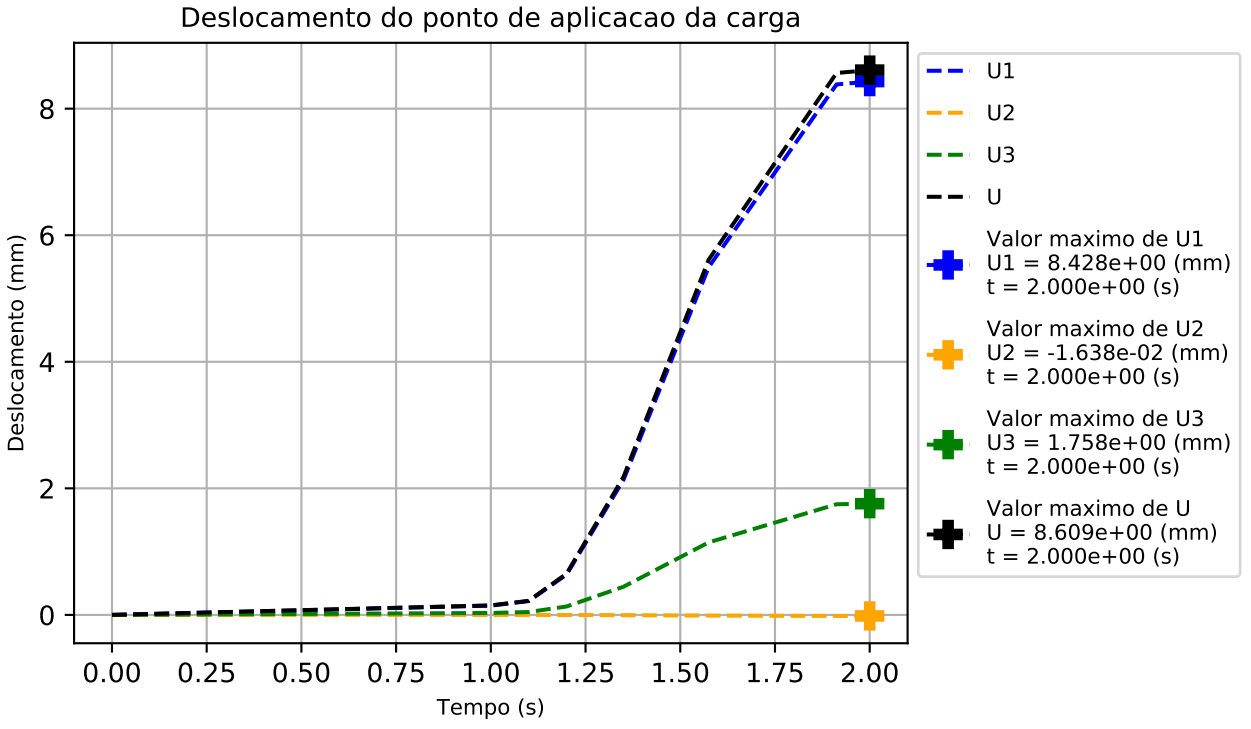

Tensao por elem.
Largura_10.0 Altura_8.0 Espessura_0.95
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0
Numero de elem.
Tensao (MPa)


Deformacao por elem.
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0
Numero de elem.
Deformacao Plastica


Deslocamento do ponto de aplicacao da carga
U1
U2
U3
U
Valor maximo de U1
U1 = 7.997e+00 (mm)
t = 1.997e+00 (s)
Valor maximo de U2
U2 = -1.555e-02 (mm)
t = 1.997e+00 (s)
Valor maximo de U3
U3 = 1.668e+00 (mm)
t = 1.997e+00 (s)
Valor maximo de U
U = 8.169e+00 (mm)
t = 1.997e+00 (s)
Deslocamento (mm)
Tempo (s)


Tensao por elem.
Largura_10.0 Altura_5.0 Espessura_0.95
Numero de elem.
Tensao (MPa)
Tensao de cedencia: 19.77 (MPa)
N de elem. acima: 0.0
80.0% tensao de cedencia:
15.816(MPa)
N de elem. acima: 0.0


Deformacao por elem.
Numero de elem.
Deformacao Plastica
Limite de deformacao plastica: 0.0545 (MPa)
N de elem. acima: 0.0


Deslocamento do ponto de aplicacao da carga
Deslocamento (mm)
Tempo (s)
U1
U2
U3
U
Valor maximo de U1
U1 = 8.428e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U2
U2 = -1.638e-02 (mm)
t = 2.000e+00 (s)
Valor maximo de U3
U3 = 1.758e+00 (mm)
t = 2.000e+00 (s)
Valor maximo de U
U = 8.609e+00 (mm)
t = 2.000e+00 (s)

6.3 Source code do software de pré-processamento

```
#from tkinter import ON
from abaqus import *
from abaqusConstants import *
from caeModules import *
import numpy as np
from abaqus import getInputs
import regionToolset
import smtplib
from email.mime.text import MIMEText
import math
import shutil
from collections import OrderedDict
from operator import getitem
import itertools as it
from itertools import product
import material
from jobMessage import*
import sys
import os
import signal
import win32api
import win32process
import win32con
import visualization
import animation
import csv

# Graphical User Interface
def GUI_Frisos():
    while True:
        # Opens the Graphical User Interface (GUI) window
        fields = [('Nome do Job:    ', 'Frisos_'), #0
                ('Largura minima dos Frisos (mm):', '15'),#1
                ('Largura maxima dos Frisos (mm):', '25'),#2
                ('N de valores da largura:', '1'),#3
                ('Espessura minima dos Frisos (mm): ', '0.5'),#4
                ('Espessura maxima dos Frisos (mm): ', '1.5'),#5
                ('N de valores da espessura:', '1'),#6
                ('Altura minima dos Frisos (mm):', '1'),#7
                ('Altura maxima dos Frisos (mm):', '10'),#8
                ('N de valores da altura:', '1'),#9
                ('Espessura da base (mm):    ', '3'),#10
                ('Tamanho de malha (mm):    ', '5'),#11
                ('Lancar a simulacao:\n 1 - Lancar a simulacao \n \
                    2 - Apenas produz o modelo    ', '1'),]#12
        user_inputs = getInputs(fields,
            label='Porfavor introduza a informacao necessaria',
            dialogTitle ='Analise iterativa de frisos')

        # Saves the GUI values into a dictionary
        user_values = {'job_name_GUI' : user_inputs[0],
        'Lenght_max_GUI' : float(user_inputs[2]),
        'Lenght_min_GUI' : float(user_inputs[1]),
        'Lenght_n_val_GUI' : int(user_inputs[3]),
```

```
        'Thickness_max_GUI' : float(user_inputs[5]),
        'Thickness_min_GUI' : float(user_inputs[4]),
        'Thickness_n_val_GUI' : int(user_inputs[6]),
        'Height_max_GUI' : float(user_inputs[8]),
        'Height_min_GUI' : float(user_inputs[7]),
        'Height_n_val_GUI' : int(user_inputs[9]),
        'base_thickness_GUI' : float(user_inputs[10]),
        'mesh_GUI' : float(user_inputs[11]),
        'simulation_GUI' : int(user_inputs[12]),}

        # Validates the GUI parameters, in order to check if any of them
        # are out of the ordinary
        if (user_values['Lenght_max_GUI']<=0) or \
            (user_values['Lenght_min_GUI']<=0)  or \
            (user_values['Lenght_n_val_GUI']<=0) or \
            (user_values['Thickness_max_GUI']<=0) or \
            (user_values['Thickness_min_GUI']<=0) or \
            (user_values['Thickness_n_val_GUI']<=0) or \
            (user_values['Height_max_GUI']<=0) or \
            (user_values['Height_min_GUI']<=0) or  \
            (user_values['Height_n_val_GUI']<=0) or \
            (user_values['base_thickness_GUI']<=0) or \
            (user_values['mesh_GUI']<=0) or \
            ((user_values['simulation_GUI'] != 1 and \
                user_values['simulation_GUI'] != 2)):
            print('Parametros Invalidos')
            answer = getWarningReply('Parametros Invalidos\n\
                Introduza novos parametros',(YES, CANCEL))
            if answer == CANCEL:
                raise Exception('A analise foi cancelada')
        else:
            return user_values



# Create a list with all the parameters of a given variable
def Create_Geometry_Variable_List(var_max_value,
    var_min_value, var_n_val):

    return np.linspace(var_min_value,
        var_max_value, var_n_val)



# Create a dictionary with all possible
# combinations of the friezes geometry
def Geometry_Test_Combinations(lenght_list,
    thickness_list, height_list):

    # Create a dictionary with all the geometry permutations
    test_permutations = [dict(zip(('Lenght','Thickness','Height'),
        (i,j,z))) for i,j,z in product(lenght_list,
```

```
109         thickness_list, height_list)]
110 
111     return test_permutations
112 
113 
114 
115 
116 # Create the folder "Estudo dos frisos_XX"
117 def Create_Folder_Structure():
118 
119     # Get current working directory
120     path = os.getcwd()
121 
122     # Create folder directory
123     newpath = path + '\Estudo dos Frisos'
124 
125     # Check if a folder with the same name already exists
126     if not (os.path.exists(newpath)):
127         os.mkdir(newpath)
128     else:
129         i = 2
130         while True:
131             newpath_2 = newpath + '_' + str(i) + ''
132             if not (os.path.exists(newpath_2)):
133                 os.mkdir(newpath_2)
134                 return newpath_2
135             else:
136                 i += 1
137 
138     return newpath
139 
140 
141 
142 
143 # Create the folder "Diretoria de trabalho"
144 def Create_Working_Directory_Folder(path):
145 
146     # Create folder directory
147     newpath = path + '\Diretoria de Trabalho'
148 
149     # Create folder
150     os.mkdir(newpath)
151 
152     return newpath
153 
154 
155 
156 
157 # Create the folder for the respective tests
158 def Create_Test_Folder(hex_size, hex_height, hex_thickness, dir_path):
159 
160     # Create folder directory
161     folder_name = ('Largura_' + str(hex_size)
162         + ' Altura_' + str(hex_height)
```

```
163         + ' Espessura_' + str(hex_thickness) + '')
164 
165     path_static = (str(dir_path) + '\\'
166         + str(folder_name) + '\Teste Estatico')
167 
168     path_ball_drop = (str(dir_path) +'\\'
169         + str(folder_name) + '\Ball Drop')
170 
171     # Create folders
172     os.makedirs(path_static)
173     os.makedirs(path_ball_drop)
174 
175     return path_static, path_ball_drop
176 
177 
178 
179 
180 def switch_working_directory(path):
181     os.chdir(path)
182 
183 
184 
185 
186 def Open_base_model(path, test):
187     newpath = path + '\Modelo_Base.cae'
188     if test == 'Estatico':
189         shutil.copyfile('C:\Users\Gabriel Ramos\Desktop\ \
190             Teste completo\Modelos\Estatico\ \
191             Modelo_prototipo_base.cae', newpath)
192     elif test == 'Ball Drop':
193         shutil.copyfile('C:\Users\Gabriel Ramos\Desktop\ \
194             Teste completo\Modelos\Ball Drop\ \
195             Modelo_prototipo_base.cae', newpath)
196 
197     return newpath
198 
199 
200 
201 
202 # Draw the hexagon profile
203 def Hex(hex_lenght,hex_height):
204 
205     # Sketch Name
206     sketch_name = 'Hex'
207 
208     # Create Hex Sketch
209     s = mymodel.ConstrainedSketch(name=sketch_name,sheetSize=200.0)
210     g, v = s.geometry, s.vertices
211     s.CircleByCenterPerimeter(center=(0.0, 0.0), point1=(11.25, 21.25))
212     s.Line(point1=(-18.75, 15.0519932234904),
213         point2=(-18.75, -15.0519932236057))
214     s.VerticalConstraint(entity=g[3], addUndoState=False)
215     s.CoincidentConstraint(entity1=v[2], entity2=g[2],
216         addUndoState=False)
```

```
    s.CoincidentConstraint(entity1=v[3], entity2=g[2],
        addUndoState=False)
    s.Line(point1=(-18.75, -15.0519932236057),
        point2=(0.0, -24.0442300770892))
    s.CoincidentConstraint(entity1=v[4],
        entity2=g[2],addUndoState=False)
    s.Line(point1=(0.0, -24.0442300770892),
        point2=(17.7218085984473, -16.25))
    s.CoincidentConstraint(entity1=v[5],
        entity2=g[2],addUndoState=False)
    s.Line(point1=(17.7218085984473, -16.25),
        point2=(17.7218085984473, 16.25))
    s.VerticalConstraint(entity=g[6], addUndoState=False)
    s.CoincidentConstraint(entity1=v[6],
        entity2=g[2],addUndoState=False)
    s.Line(point1=(17.7218085984473, 16.25),
        point2=(-1.25, 24.0117158903731))
    s.CoincidentConstraint(entity1=v[7],
        entity2=g[2],addUndoState=False)
    s.Line(point1=(-1.25, 24.0117158903731),
        point2=(-18.75, 15.0519932234904))
    s.setAsConstruction(objectList=(g[2], ))
    s.EqualLengthConstraint(entity1=g[3], entity2=g[8])
    s.EqualLengthConstraint(entity1=g[8],
        entity2=g[7],addUndoState=False)
    s.EqualLengthConstraint(entity1=g[7],
        entity2=g[6],addUndoState=False)
    s.EqualLengthConstraint(entity1=g[6],
        entity2=g[5],addUndoState=False)
    s.EqualLengthConstraint(entity1=g[5],
        entity2=g[4],addUndoState=False)

    # Set the hexagon size
    s.DistanceDimension(entity1=g[3], entity2=g[6],
        textPoint=(0.0668716430664063,
        31.3793067932129), value=hex_lenght)

    s.Spot(point=(0.0, 0.0))
    s.FixedConstraint(entity=v[8])
    s.CoincidentConstraint(entity1=v[8], entity2=v[0])

    return sketch_name



# Create an offset to be used as a particion face
def Create_Particion_Face(hex_height, base_thickness):

    # Internal variables
    p = mymodel.parts['Base']

    # Create Particion Face
    offset_distance = (hex_height+base_thickness/2)*-1
```

```
271
272     p.OffsetFaces(faceList = p.sets['Base'].faces,
273         distance= offset_distance,
274         trimToReferenceRep=False)
275
276     # Get the name of the last feature created
277     offset_name = p.features.summary()[-1][0]
278
279     # Get faces from the feature offset_name
280     offset_faces = p.getFeatureFaces(name = offset_name)
281
282     # Create Particion Face Set
283     p.Set(faces=offset_faces, name='Particion_Base')
284
285
286
287
288 # This function initiates the sketch
289 def sketch_start(drawing_plane, align_edge, p,
290     profile_name, center_point):
291
292     # Get center Point Coordinates
293     d3 =  p.datums
294     sketch_center_point_id = p.features[center_point].id
295     sketch_center_point_coordinates = p.getCoordinates(
296         d3[sketch_center_point_id])
297     center_x_coordinate = sketch_center_point_coordinates[0]
298     center_y_coordinate = sketch_center_point_coordinates[1]
299     center_z_coordinate = sketch_center_point_coordinates[2]
300
301     # Create the transformation matrix
302     t = p.MakeSketchTransform(sketchPlane= drawing_plane,
303         sketchUpEdge= align_edge,
304         sketchPlaneSide=SIDE1, sketchOrientation=RIGHT,
305         origin=(center_x_coordinate,
306         center_y_coordinate, center_z_coordinate))
307
308     # Start Sketch
309     s = mymodel.ConstrainedSketch(name=profile_name,
310             sheetSize=2460.97, gridSpacing=61.52, transform=t)
311     g, v = s.geometry, s.vertices
312     p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)
313
314     return( s, g, v )
315
316
317
318
319 # This function receives a set of coordinates
320 # and draws a hexagon on that same point
321 def draw_hex (hex_x_lenght, hex_y_lenght, s, sketch_name, g, v):
322
323     # Retrive the hexagon sketch that was saved
324     s.retrieveSketch(sketch=mymodel.sketches[sketch_name])
```

```
325
326     # Move the hexagon to the respective coordinates
327     s.move(vector=(hex_x_lenght, hex_y_lenght),
328         objectList=(g[max(g.keys())-6],
329         g[max(g.keys())-5],
330         g[max(g.keys())-4],
331         g[max(g.keys())-3],
332         g[max(g.keys())-2],
333         g[max(g.keys())-1],
334         g[max(g.keys())],
335         v[max(v.keys())]))
336
337
338
339
340
341 # This function draws the diagonal hexagons that start at
342 # line A of the hexagone area
343 def draw_diagonal_hex_A(hex_size, hex_y_lenght_base,
344     hex_x_lenght_base, first_int, hex_diagonal_number, s,
345     sketch_name, g, v):
346
347     i = 1
348     while i <= hex_diagonal_number:
349
350     # Calculation of the horizontal and
351     # vertical coordinates of the hexagon
352         hex_x_lenght = hex_x_lenght_base + \
353             i * hex_size * np.cos(np.radians(60))
354
355         hex_y_lenght = hex_y_lenght_base + \
356             i * hex_size * np.sin(np.radians(60))
357
358     # Sketch the hexagon on the calculated coordinates
359         draw_hex (hex_x_lenght, hex_y_lenght, s, sketch_name, g, v)
360
361     # This part of the functions makes it so that hexagons
362     # that aren't necessary for the drawing are skipped
363     # resulting in a simpler computacional model
364         if (i == hex_diagonal_number-1) or (first_int == 1):
365             i += 1
366         else:
367             i += 2
368
369
370
371 # This function draws the diagonal hexagons of zone B
372 def draw_diagonal_hex_B (hex_diagonal_number, s, sketch_name,
373     horizontal_limit_3, hex_size, lim_2, lim_3, lim_4,
374     lim_2_counter, lim_3_counter, hex_x_lenght, hex_y_lenght,
375     first_int, hex_diagonal_number_1, hex_diagonal_number_2, g, v):
376
377     # Loop through the diagonal line
378     index_B = 0
```

```
379     while index_B <= (hex_diagonal_number):
380 
381         # Calculate the hexagon coordenates
382         diagonal_hex_x_lenght = hex_x_lenght + \
383             (index_B)*hex_size*np.cos(np.radians(60))
384 
385         diagonal_hex_y_lenght = hex_y_lenght + \
386             (index_B)*hex_size*np.sin(np.radians(60))
387 
388         # Draw the hexagon on the calculated coordinates
389         draw_hex(diagonal_hex_x_lenght, diagonal_hex_y_lenght,
390             s, sketch_name, g, v)
391 
392         # Check if the current diagonal line is the last one
393         last_diagonal_check = final_diagonal_check(horizontal_limit_3,
394             diagonal_hex_x_lenght, hex_size, index_B,
395             hex_diagonal_number)
396 
397         # Check if the last hexagon is forced to activate
398         forced_last_line = activate_last_line(lim_3,
399             hex_diagonal_number)
400 
401         # Logical conditions that check if a hexagon should be supressed
402         if ((first_int ==2) and
403             not(lim_2  and lim_2_counter == 1 and
404             (index_B+1)>= hex_diagonal_number_1) and
405             not(lim_3  and lim_3_counter == 1 and
406             (index_B+1)>= hex_diagonal_number_2) and
407             not(lim_4  and index_B==(hex_diagonal_number-2) or\
408             (forced_last_line and last_diagonal_check and not\
409             (lim_4)))):
410 
411             index_B +=2
412             if index_B == (hex_diagonal_number+1):
413                 index_B =  hex_diagonal_number
414         else:
415             index_B +=1
416 
417 
418 
419 
420 # Check if there is continuity between Zone A and B
421 # If there is not, than it alters the parameters
422 # In order for the continuity to happen
423 def continuity_checker(first_int, hex_vertical_number_A):
424 
425   if ((hex_vertical_number_A%2 == 0) and
426         (first_int == 1) and
427         (hex_vertical_number_A%4 == 0)) or \
428         ((hex_vertical_number_A%2 == 0) and
429         (first_int == 2) and
430         (hex_vertical_number_A%4 != 0)) or \
431         ((hex_vertical_number_A%2 != 0) and
432         (first_int == 1) and
```

```
433         ((hex_vertical_number_A-1)%4 != 0) or \
434         ((hex_vertical_number_A%2 != 0) and
435         (first_int == 2) and
436         ((hex_vertical_number_A-1)%4 == 0))):
437 
438     first_int = 1
439   else:
440     first_int = 2
441 
442   return first_int
443 
444 
445 
446 
447 # This function calculates what horizontal limit the program should use,
448 # while drawing the diagonal hexagons.
449 # It basicaly works as a boundary detection function,
450 # and does not allow the hexagon drawing to leave the hexagon area
451 def limit_checker(lim_2, lim_3, lim_4, hex_y_lenght, hex_x_lenght,
452     horizontal_limit_1, horizontal_limit_2, horizontal_limit_3,
453     vertical_limit_1, vertical_limit_2, vertical_limit_3,
454     lim_2_counter, lim_3_counter,
455     hex_diagonal_number_1, hex_diagonal_number_2,hex_size,
456     hex_thickness, hex_height, hex_thickness_height):
457 
458     # Internal Variable
459     prev_lim = False
460 
461     # Measurements of the first limit triangle
462     hex_vertical_leg_1 = vertical_limit_1 - hex_y_lenght
463     hex_horizontal_leg_1 = hex_vertical_leg_1/np.tan(np.radians(60))
464 
465     # Check if it should use line H as it's upper limit
466     if hex_horizontal_leg_1 <= (horizontal_limit_1-hex_x_lenght):
467 
468         # Calculate the number of diagonal hexagons
469         hex_diagonal_number = int(math.floor((
470             (hex_vertical_leg_1)/\
471             np.cos(np.radians(30))-\
472             (hex_height/2+hex_thickness_height)/\
473             np.cos(np.radians(30)))/hex_size))
474 
475         # Argument used for the correction 1 to know when to activate
476         hex_diagonal_number_1 = hex_diagonal_number
477 
478         # Turns of limit 2, in case if the function enters
479         # it's recursive state
480         lim_2 = False
481 
482     else:
483         # Measurements of the second limit triangle
484         hex_vertical_leg_2 = vertical_limit_2 - hex_y_lenght
485         hex_horizontal_leg_2 = hex_vertical_leg_2/np.tan(np.radians(60))
486 
```

```
487         # Check if it should use line F as it's upper limit
488         if hex_horizontal_leg_2 <= (horizontal_limit_2-hex_x_lenght):
489
490             # Calculate the number of diagonal hexagons
491             hex_diagonal_number = int(math.floor((
492                 (hex_vertical_leg_2)/\
493                 np.cos(np.radians(30))-\
494                 (hex_height/2 + hex_thickness_height)/\
495                 np.cos(np.radians(30)))/hex_size))
496
497             # Argument used for the correction 1
498             # to know when to activate
499             hex_diagonal_number_2 = hex_diagonal_number
500
501             # Turns of limit 3, in case if the function
502             # enters it's recursive state
503             lim_2 = True
504             lim_3 = False
505             lim_2_counter += 1
506
507             # Use recursion in order to check if there was
508             # a transition between limits
509             if lim_2_counter == 1:
510                 prev_lim = limit_checker(lim_2, lim_3, lim_4,
511                     hex_y_lenght, (hex_x_lenght-hex_size),
512                     horizontal_limit_1, horizontal_limit_2,
513                     horizontal_limit_3, vertical_limit_1,
514                     vertical_limit_2, vertical_limit_3,
515                     lim_2_counter, lim_3_counter,
516                     hex_diagonal_number_1, hex_diagonal_number_2,
517                     hex_size, hex_thickness,
518                     hex_height, hex_thickness_height)[0]
519
520             if prev_lim:
521                 lim_2_counter +=1
522         else:
523             # Measurements of the third limit triangle
524             hex_vertical_leg_3 = vertical_limit_3 - hex_y_lenght
525             hex_horizontal_leg_3 = hex_vertical_leg_3/\
526                 np.tan(np.radians(60))
527
528             # Check if it should use line D as it's upper limit
529             if hex_horizontal_leg_3 <= (horizontal_limit_3- \
530                 hex_x_lenght):
531
532                 # Calculate the number of diagonal hexagons
533                 hex_diagonal_number = int(math.floor((
534                     (hex_vertical_leg_3)/\
535                     np.cos(np.radians(30))-\
536                     (hex_height/2 + hex_thickness_height)/\
537                     np.cos(np.radians(30)))/hex_size))
538
539                 # Turns of limit 2, in case if the function enters
540                 # it's recursive state
```

```
541                 lim_2 = False
542                 lim_3 = True
543                 lim_3_counter += 1
544 
545                 # Use recursion in order to check if there was a
546                 # transition between limits
547                 if lim_3_counter == 1:
548                     prev_lim = limit_checker(lim_2, lim_3, lim_4,
549                     hex_y_lenght, (hex_x_lenght-hex_size),
550                         horizontal_limit_1, horizontal_limit_2,
551                         horizontal_limit_3, vertical_limit_1,
552                         vertical_limit_2, vertical_limit_3,
553                         lim_2_counter, lim_3_counter,
554                         hex_diagonal_number_1, hex_diagonal_number_2,
555                         hex_size, hex_thickness,
556                         hex_height, hex_thickness_height)[1]
557 
558                 if prev_lim:
559                     lim_3_counter +=1
560 
561             # Check if it should use line C as it's upper limit
562             else:
563                 # Measurements of the fourth limit triangle
564                 hex_horizontal_leg_4 = horizontal_limit_3 - hex_x_lenght
565                 hex_vertical_leg_4 = hex_horizontal_leg_4*\
566                     np.tan(np.radians(60))
567 
568                 # Calculate the number of diagonal hexagons
569                 hex_diagonal_number = int(math.floor((
570                     (hex_vertical_leg_4)/\
571                     np.cos(np.radians(30))-\
572                     (hex_height/2 + hex_thickness_height)/\
573                     np.cos(np.radians(30)))/hex_size))
574 
575 
576                 # Turns of limit 3, in case if the function enters
577                 # it's recursive state
578                 lim_3 = False
579                 lim_4 = True
580 
581     return (lim_2, lim_3, lim_4, hex_diagonal_number,
582         lim_2_counter, lim_3_counter,
583         hex_diagonal_number_1, hex_diagonal_number_2)
584 
585 
586 
587 
588 # This function checks if a diagonal line is the last one of the limit
589 def final_diagonal_check(horizontal_limit,
590     diagonal_hex_x_lenght, hex_size, index_B, hex_diagonal_number):
591 
592     if (horizontal_limit-diagonal_hex_x_lenght) < (2.2*hex_size) and \
593         (index_B == (hex_diagonal_number-2)):
594 
```

```
595         # The value True means that it is the last line
596         last_diagonal_check = True
597     else:
598         # The value False means that it isn't the last line
599         last_diagonal_check = False
600 
601     return last_diagonal_check
602 
603 
604 
605 
606 # This function checks if the last hexagon
607 # of the diagonal line is forced to activate
608 def activate_last_line(lim, hex_diagonal_number):
609 
610     if (lim and (hex_diagonal_number%2!=0)):
611         # The value True means that the hexagon
612         # was forced to activate
613         forced_last_line = True
614     else:
615         # The value False means that the hexagon
616         # was not forced to activate
617         forced_last_line = False
618 
619     return forced_last_line
620 
621 
622 
623 
624 def Create_Hex_Assembly(hex_size, hex_lenght, hex_height,
625     hex_thickness, sketch_name,
626     A,B,C,D,E,F,G,H):
627 
628     # Pre-requesite: Center Point Name
629     center_point = 'Sketch_Center_Point'
630 
631     # Pre-requesite: CSYS Datum that gives the drawing orientation
632     p = mymodel.parts['Base']
633     d3 =  p.datums
634     align_edge_id = p.features['Drawing_CSYS'].id
635     align_edge = d3[align_edge_id].axis2
636 
637     # Pre-requesite: Drawing plane datum
638     drawing_plane_id = p.features['Sketch_Plane'].id
639     drawing_plane = d3[drawing_plane_id]
640 
641     # Profile Name
642     profile_name = '__profile__'
643 
644     # Initiate the Sketch
645     s, g, v = sketch_start(drawing_plane, align_edge, p,
646         profile_name, center_point)
647 
648     # Bollean that checks its the first sketch step
```

```
649     sketch_step_1 = True
650
651
652     # Sketching the hexagons through line A
653     for first_int in range(1,3):
654
655         # Calculate the number of hexagons that fit in line A
656         hex_thickness_height = hex_thickness/(2*np.cos(np.radians(30)))
657         available_height = A + hex_lenght - 2*hex_thickness_height
658         geometry_height = hex_height + hex_lenght
659         hex_vertical_number_A = int(
660             math.floor(available_height/geometry_height))
661
662         # Alters the base value of the pair index
663         pair_index_A = 1
664         if (hex_vertical_number_A%2 != 0) and (first_int == 2):
665             pair_index_A = 2
666
667         # Calculate the hexagon coordinates
668         while (2*pair_index_A-1) <= hex_vertical_number_A:
669
670             # Calculate the hexagon x coordinate
671             hex_x_lenght = hex_size/2 + hex_thickness/2
672
673             # Calculations if the number of hexagons is even
674             if hex_vertical_number_A%2 == 0:
675                 # Conditions to intercalate the sketch
676                 if (first_int == 1 and pair_index_A%2 != 0) \
677                     or (first_int == 2 and (pair_index_A)%2 == 0):
678                     # Bottom hexagon y coordinate
679                     hex_y_lenght = A/2-(pair_index_A-\
680                        0.5)*geometry_height
681                 else:
682                     # Top hexagon y coordinate
683                     hex_y_lenght = A/2+(pair_index_A-\
684                        0.5)*geometry_height
685
686                 # Sketch the hexagon that's along the vertical line
687                 draw_hex(hex_x_lenght, hex_y_lenght, s,
688                        sketch_name, g, v)
689
690                 # Calculate the number of hexagons that fit diagonaly
691                 hex_diagonal_number = int(math.floor((\
692                     (A-hex_y_lenght)/np.cos(np.radians(30)) \
693                     -(hex_height/2 + hex_thickness_height)/ \
694                     np.cos(np.radians(30)))/hex_size))
695
696                 # Sketch the hexagons that fit diagonaly
697                 draw_diagonal_hex_A(hex_size, hex_y_lenght,
698                     hex_x_lenght,
699                     first_int,hex_diagonal_number, s,
700                     sketch_name, g, v)
701
702             else:
```

```
703                 # Calculations if the number of hexagones is odd
704                 # Calculate the hexagons y coordinates
705                 const = pair_index_A-1
706                 hex_y_lenght_1 = A/2-(const)*geometry_height
707                 hex_y_lenght_2 = A/2+(const)*geometry_height
708 
709                 # Sketch the hexagon that's along the vertical line
710                 draw_hex(hex_x_lenght, hex_y_lenght_1, s,
711                     sketch_name, g, v)
712 
713                 # Calculate the number of hexagons that fit diagonaly
714                 hex_diagonal_number = int(
715                     math.floor(((A-hex_y_lenght_1)/ \
716                     np.cos(np.radians(30))-(hex_height/2 + \
717                     hex_thickness_height)/ \
718                     np.cos(np.radians(30)))/hex_size))
719 
720                 # Sketch the hexagons that fit diagonaly
721                 draw_diagonal_hex_A(hex_size, hex_y_lenght_1,
722                     hex_x_lenght,
723                     first_int, hex_diagonal_number, s,
724                     sketch_name, g, v)
725 
726                 # Sketch the second hexagon
727                 if hex_y_lenght_1 != hex_y_lenght_2:
728 
729                     # Sketch the hexagon that's along the vertical line
730                     draw_hex(hex_x_lenght, hex_y_lenght_2, s,
731                         sketch_name, g, v)
732 
733                     # Calculate the number of diagonal hexagons
734                     hex_diagonal_number = int(
735                         math.floor(((A-hex_y_lenght_2)/ \
736                         np.cos(np.radians(30)) -(hex_height/2 + \
737                         hex_thickness_height)/ \
738                         np.cos(np.radians(30)))/hex_size))
739 
740                     # Sketch the hexagons that fit diagonaly
741                     draw_diagonal_hex_A(hex_size, hex_y_lenght_2,
742                         hex_x_lenght,
743                         first_int, hex_diagonal_number, s,
744                         sketch_name, g, v)
745 
746                 pair_index_A += 1
747             pair_index_A += 1
748 
749 
750         # calculate the boundary limits
751         horizontal_limit_1 = H + (hex_lenght + hex_thickness)/2
752         horizontal_limit_2 = H + F + (hex_lenght + hex_thickness)/2
753         horizontal_limit_3 = B - hex_size/2 - hex_thickness/2
754         vertical_limit_1 = A
755         vertical_limit_2 = A + G
756         vertical_limit_3 = C
```

```
757
758         # Limit Switches, that check in what area the hexagon is drawn
759         lim_2 = False
760         lim_2_counter = 0
761         lim_3 = False
762         lim_3_counter = 0
763         lim_4 = False
764         hex_diagonal_number_1 = 0
765         hex_diagonal_number_2 = 0
766
767         # Calls the function that makes sure that there is continuity
768         # between Line A and B
769         first_int_B = continuity_checker(first_int,
770             hex_vertical_number_A)
771
772         # Calculate the hexagon vertical coordinate
773         hex_y_lenght = A/2-(hex_vertical_number_A-1)*((hex_lenght +\
774             hex_height)/2)
775
776         # Calculate the number of hexagons along line B
777         horizontal_hex_number_B = int(
778             math.floor((B-hex_thickness)/hex_size))
779
780         for index_B in range (first_int_B, horizontal_hex_number_B, 2):
781
782             # Calculate the hexagon horizontal coordinate
783             hex_x_lenght = hex_size/2 + hex_thickness/2 + \
784                index_B*hex_size
785
786             # This checks what boundaries of the hexagon area uses
787             lim_2, lim_3, lim_4, hex_diagonal_number,lim_2_counter, \
788                 lim_3_counter,hex_diagonal_number_1, \
789                 hex_diagonal_number_2 = limit_checker(
790                 lim_2, lim_3, lim_4, hex_y_lenght,
791                 hex_x_lenght, horizontal_limit_1,
792                 horizontal_limit_2,
793                 horizontal_limit_3, vertical_limit_1, vertical_limit_2,
794                 vertical_limit_3, lim_2_counter,
795                 lim_3_counter, hex_diagonal_number_1,
796                 hex_diagonal_number_2,
797                 hex_size, hex_thickness,
798                 hex_height, hex_thickness_height)
799
800             # Draw the base hexagon and it's diagonal line
801             draw_diagonal_hex_B(hex_diagonal_number, s, sketch_name,
802                 horizontal_limit_3, hex_size,
803                 lim_2, lim_3, lim_4, lim_2_counter,
804                 lim_3_counter, hex_x_lenght, hex_y_lenght,
805                 first_int, hex_diagonal_number_1,
806                 hex_diagonal_number_2, g, v)
807
808
809         # If it's the first sketch, it needs to be saved
810         # in memory, to be extruded later
```

```
811         if sketch_step_1:
812 
813             # Save current sketch in memory
814             mymodel.ConstrainedSketch(name='Sketch_Step_1',
815                 objectToCopy=s)
816 
817             # Get all the geometrys from the current sketch
818             base_hedge_keys = g.keys()
819             list_of_hedges_to_delete = []
820 
821             for del_index in base_hedge_keys:
822                 list_of_hedges_to_delete.append(g[del_index])
823 
824             # Delete Current sketch from the sketch window
825             s.delete(objectList =(list_of_hedges_to_delete))
826 
827         # This boolean makes the program know that the first step
828         # has been completed
829         sketch_step_1 = False
830 
831 
832     # Extrude Sketch Step 2
833     Extrude_sketch(drawing_plane, align_edge, s, profile_name)
834 
835     # Start Sketch Step 1
836     s, g, v = sketch_start(drawing_plane, align_edge,
837         p, profile_name, center_point)
838 
839     # Get Sketch_Step_1 saved sketch
840     s.retrieveSketch(sketch=mymodel.sketches['Sketch_Step_1'])
841 
842     # Extrude Sketch Step 1
843     Extrude_sketch(drawing_plane, align_edge, s, profile_name)
844 
845 
846 
847 
848 # Extrude a given sketch
849 def Extrude_sketch(drawing_plane, align_edge,
850     sketch_instance, profile_name):
851 
852     # Internal variable
853     p = mymodel.parts['Base']
854 
855     # Get the face to extrude to
856     bottom_extrude_face = p.sets['Sketch_Plane_Bottom'].faces[0]
857 
858     # Extrude the sketch
859     p.ShellExtrude(sketchPlane = drawing_plane,
860         sketchUpEdge = align_edge, upToFace = bottom_extrude_face,
861         sketchPlaneSide=SIDE1, sketchOrientation=RIGHT,
862         sketch=sketch_instance, flipExtrudeDirection=ON)
863 
864     # Delete the sketch
```

```
865     del mymodel.sketches[profile_name]
866
867
868
869
870 # Function to trim the Frizes, so that they are at the appropriate height
871 def Trim_Hex_Assembly(bounding_box_point_1_datum,
872     bounding_box_point_2_datum):
873
874     # Internal variables
875     p = mymodel.parts['Base']
876     f = p.faces
877     s = p.edges
878     d = p.datums
879
880     # Boundary box Coordinates
881     bounding_box_point_1_id = p.features[bounding_box_point_1_datum].id
882     bounding_box_point_2_id = p.features[bounding_box_point_2_datum].id
883     bounding_box_point_1 = p.getCoordinates(d[bounding_box_point_1_id])
884     bounding_box_point_2 = p.getCoordinates(d[bounding_box_point_2_id])
885
886     # Reordering of the boundary box coordinates
887     min_x_top = min(bounding_box_point_1[0],bounding_box_point_2[0])
888     min_y_top = min(bounding_box_point_1[1],bounding_box_point_2[1])
889     min_z_top = min(bounding_box_point_1[2],bounding_box_point_2[2])
890
891     max_x_top = max(bounding_box_point_1[0],bounding_box_point_2[0])
892     max_y_top = max(bounding_box_point_1[1],bounding_box_point_2[1])
893     max_z_top = max(bounding_box_point_1[2],bounding_box_point_2[2])
894
895     # Get boundary edges from the frizes trim area
896     hex_boundary_edges_region = s.getByBoundingBox(min_x_top,min_y_top,
897         min_z_top,max_x_top,max_y_top,max_z_top)
898
899     # Get the faces associated with each boundary edge and delete them
900     del_face_region = []
901     del_face_list = []
902     for edge_index in range(len(hex_boundary_edges_region)):
903         face_from_edge_tuple = hex_boundary_edges_region[edge_index] \
904             .getFaces()
905         face_from_edge = face_from_edge_tuple[0]
906         del_face_list.append(face_from_edge)
907         del_face_region.append(f[face_from_edge:(face_from_edge+1)])
908     p.Set(faces=del_face_region, name='Set-remove')
909
910     # Create a set with the frizes faces,
911     # if they are from the top bounding box
912     if (bounding_box_point_1_datum == 'Top_Bounding_Box_P1') or \
913         (bounding_box_point_2_datum == 'Top_Bounding_Box_P2'):
914         Frizes_Set(del_face_list)
915
916     # Delete the frizes that are in excession
917     Delete_Set('Set-remove')
918
```

```
919
920
921
922 # Delete the faces inside a given set
923 def Delete_Set(set_for_deletion):
924
925     # Internal variables
926     p = mymodel.parts['Base']
927
928     # Remove the faces inside the set that was
929     # received as an argument
930     set_deletion_region = p.sets[set_for_deletion].faces
931     p.RemoveFaces(faceList = set_deletion_region, deleteCells=False)
932
933
934
935
936 # Create a Set with the Frizes in it
937 def Frizes_Set(del_face_region):
938
939     # Internal Variables
940     p = mymodel.parts['Base']
941     f = p.faces
942     friezes_region = []
943     friezes_set_region_id = []
944     friezes_set_region = []
945
946     # Get adjacent faces from the ones that are in excess
947     for i in range(len(del_face_region)):
948         friezes_region.extend(f[del_face_region[i]].getAdjacentFaces())
949
950     # Get the adjacent faces index number
951     for i in range(len(friezes_region)):
952         friezes_set_region_id.append(friezes_region[i].index)
953
954     # Get the adjacent faces in a list, in a form to create a set
955     for i in range(len(friezes_set_region_id)):
956         friezes_set_region.append( \
957             f[friezes_set_region_id[i]:(friezes_set_region_id[i]+1)])
958
959     # Create the Set
960     p.Set(faces=friezes_set_region, name='Frizes')
961
962
963
964 # Read the data base of materials
965 def Read_True_Stress_Strain_Values(test_speed):
966
967     if test_speed == 16.6:
968
969         material_file_name = (r'D:\Gabriel Ramos\Tese\Teste do software\
970             Base de Dados Material\EE189HP\EE189HP_16.6.txt')
971     elif test_speed == 0.01:
972         material_file_name = (r'D:\Gabriel Ramos\Tese\Teste do software\
```

```
973             Base de Dados Material\EE189HP\EE189HP_0.01.txt')
974
975
976     # Open the materaial file
977     f = open(material_file_name)
978
979     # Read the file
980     lines=f.readlines()
981
982     # Get the yield stress
983     yield_stress = (float(lines[0].split('    ')[5]))
984
985     # Get the Elastic modulus
986     modulus = (float(lines[0].split(' ')[2]))
987
988     # Get the poisson modulus
989     poisson = (float(lines[0].split(' ')[3]))
990
991     # Get the density
992     density = (float(lines[0].split(' ')[4]))
993
994     # Get elastic energy coefficient
995     limit_strain = (float(lines[0].split('    ')[6]))
996
997     # Get the platicity curve values
998     plasticity_curve_list = []
999     for x in lines:
1000         stress_strain_line = x
1001         stress_strain_list = stress_strain_line.split('      ')
1002         plasticity_curve_list.append((float(stress_strain_list[1]),
1003         float(stress_strain_list[0])))
1004     f.close()
1005
1006     # Convert the plasticity curve values into a tuple
1007     plasticity_curve_tuple = tuple(plasticity_curve_list)
1008
1009     return plasticity_curve_tuple, yield_stress, modulus, \
1010         poisson, density, limit_strain
1011
1012
1013
1014
1015 def Atribute_thickness(density, modulus, poisson,
1016     hex_thickness, plasticity_curve_tuple):
1017
1018     # Create Material Properties
1019     p = mymodel.parts['Base']
1020     mymaterial = mymodel.Material(name='Script Material')
1021     mymaterial.Density(table=((density, ), ))
1022     mymaterial.Elastic(table=((modulus, poisson), ))
1023     mymaterial.Plastic(table = plasticity_curve_tuple)
1024
1025     # Create Section - Frizes Section
1026     frizes_section = mymodel.HomogeneousShellSection(
```

```
1027         name='Section-Frizes',
1028         preIntegrate=OFF, material='Script Material',
1029         thicknessType=UNIFORM,
1030         thickness=hex_thickness, thicknessField='',
1031         nodalThicknessField='',
1032         idealization=NO_IDEALIZATION, poissonDefinition=DEFAULT,
1033         thicknessModulus=None, temperature=GRADIENT, useDensity=ON,
1034         integrationRule=SIMPSON, numIntPts=5)
1035 
1036     # Create Section - Base Section
1037     base_section = mymodel.HomogeneousShellSection(name='Section-Base',
1038         preIntegrate=OFF, material='Script Material',
1039         thicknessType=UNIFORM,
1040         thickness=3.0, thicknessField='', nodalThicknessField='',
1041         idealization=NO_IDEALIZATION, poissonDefinition=DEFAULT,
1042         thicknessModulus=None, temperature=GRADIENT, useDensity=ON,
1043         integrationRule=SIMPSON, numIntPts=5)
1044 
1045     # Assign Section - Friezes Section
1046     p.SectionAssignment(region=p.sets['Frizes'],
1047         sectionName=frizes_section.name, offset=0.0,
1048         offsetType=MIDDLE_SURFACE, offsetField='',
1049         thicknessAssignment=FROM_SECTION)
1050 
1051     # Assign Section - Base Section
1052     p.SectionAssignment(region=p.sets['Base'],
1053         sectionName=base_section.name, offset=0.0,
1054         offsetType=MIDDLE_SURFACE, offsetField='',
1055         thicknessAssignment=FROM_SECTION)
1056 
1057 
1058 
1059 # Mesh the base
1060 def Mesh_Part(mesh_size):
1061 
1062     # Internal variables
1063     p = mymodel.parts['Base']
1064 
1065     # Create Mesh
1066     myview.setValues(displayedObject=p)
1067     p.seedPart(size=mesh_size, deviationFactor=0.1,
1068     minSizeFactor=0.1)
1069     p.generateMesh()
1070 
1071 
1072 
1073 
1074 # Create the interaction an interaction properties
1075 # between the different components in the static test
1076 def Interaction_Static_Test():
1077 
1078     # Internal Variable
1079     a = mymodel.rootAssembly
1080 
```

```
    # Create interaction Properties: Stamp contact
    mymodel.ContactProperty('IntProp-Stamp')
    mymodel.interactionProperties['IntProp-Stamp'].TangentialBehavior(
        formulation=PENALTY, directionality=ISOTROPIC,
        slipRateDependency=OFF,
        pressureDependency=OFF, temperatureDependency=OFF,
        dependencies=0,
        table=((0.22, ), ), shearStressLimit=None,
        maximumElasticSlip=FRACTION,
        fraction=0.005, elasticSlipStiffness=None)

    mymodel.interactionProperties['IntProp-Stamp'].NormalBehavior(
        pressureOverclosure=HARD, allowSeparation=ON,
        contactStiffness=DEFAULT,
        contactStiffnessScaleFactor=1.0,
        clearanceAtZeroContactPressure=0.0,
        constraintEnforcementMethod=AUGMENTED_LAGRANGE)

    # Create face to face interaction
    region1 = a.instances['Base-1'].surfaces['Base_Surf']
    region2 = a.instances['Stamp-1'].surfaces['Stamp_Surf']

    mymodel.SurfaceToSurfaceContactStd(name='Stamp_Contact',
        createStepName='Initial', master=region2, slave=region1,
        sliding=FINITE, thickness=ON,
        interactionProperty='IntProp-Stamp',adjustMethod=NONE,
        initialClearance=OMIT, datumAxis=None, clearanceRegion=None)



def Inertia():

    a = mymodel.rootAssembly

    # Calculate Inertia
    ball_mass_ton = 0.0005/4
    inertia = (2/5*ball_mass_ton*25**2)/2

    # Atribute Inertia
    region=a.sets['Ball_Drop_RP']
    mymodel.rootAssembly.engineeringFeatures.PointMassInertia(
        name='Inertia-2', region=region, mass=ball_mass_ton,
        i11=inertia, i22=inertia,
        i33=inertia, alpha=0.0, composite=0.0)



def Interaction_Ball_Drop_Test():

    a = mymodel.rootAssembly
    Inertia()

```

```
1135     # Create interaction: Ball contact
1136     mymodel.ContactProperty('IntProp-Ball')
1137     mymodel.interactionProperties['IntProp-Ball'].TangentialBehavior(
1138         formulation=PENALTY, directionality=ISOTROPIC,
1139         slipRateDependency=OFF,
1140         pressureDependency=OFF, temperatureDependency=OFF,
1141         dependencies=0,
1142         table=((0.22, ), ), shearStressLimit=None,
1143         maximumElasticSlip=FRACTION,
1144         fraction=0.005, elasticSlipStiffness=None)
1145
1146     mymodel.interactionProperties['IntProp-Ball'].NormalBehavior(
1147         pressureOverclosure=HARD, allowSeparation=ON,
1148         constraintEnforcementMethod=DEFAULT)
1149
1150     region1=a.instances['Ball-1'].surfaces['SURF']
1151     region2=a.instances['Base-1'].surfaces['SURF']
1152
1153     mymodel.SurfaceToSurfaceContactExp(name ='Int-Ball',
1154         createStepName='Initial', master = region1,
1155         slave = region2,
1156         mechanicalConstraint=KINEMATIC, sliding=FINITE,
1157         interactionProperty='IntProp-Ball',
1158         initialClearance=OMIT, datumAxis=None,
1159         clearanceRegion=None)
1160
1161
1162
1163
1164 # Create the static test parameters
1165 def Static_test(solve_model, job_name, my_path_1, my_path_2,
1166     final_report, report_counter, hex_size, hex_thickness,
1167     hex_height, yield_stress, limit_strain):
1168
1169     # Internal ariables
1170     a = mymodel.rootAssembly
1171     da = a.datums
1172     Stamp_Load_Point_CSYS_id = a.features['Stamp_Load_Point_CSYS'].id
1173
1174     # Create Displacement Step
1175     mymodel.StaticStep(name='Step-Static-Test-Displacement',
1176         previous='Initial',
1177         timePeriod=1.0, initialInc=0.1, minInc=2e-05, maxInc=1.0,
1178         nlgeom=ON)
1179
1180     # Create Load Step
1181     mymodel.StaticStep(name='Step-Static-Test-Force',
1182         previous='Step-Static-Test-Displacement',
1183         timePeriod=1.0, initialInc=0.1, minInc=2e-05, maxInc=1.0,
1184         nlgeom=ON)
1185
1186     # Create Displacement
1187     region = a.sets['Stamp-1.Stamp_RP']
1188     mymodel.DisplacementBC(name='BC-Stamp_Displacement',
```

```
1189        createStepName='Initial', region=region, u1=SET, u2=SET,
1190        u3=UNSET, ur1=SET, ur2=SET, ur3=SET, amplitude=UNSET,
1191        distributionType=UNIFORM, fieldName='',
1192        localCsys=da[Stamp_Load_Point_CSYS_id])
1193
1194    mymodel.boundaryConditions['BC-Stamp_Displacement']. \
1195        setValuesInStep(stepName='Step-Static-Test-Displacement',
1196        u3=0.15)
1197
1198    mymodel.boundaryConditions['BC-Stamp_Displacement']. \
1199        setValuesInStep(stepName='Step-Static-Test-Force',
1200        u3=FREED)
1201
1202    # Static load amplitude
1203    myamplitude = mymodel.SmoothStepAmplitude(
1204        name='Static_Load_Amplitude', timeSpan=STEP,
1205        data=((0.0, 0.0), (1.0, 1.0)))
1206
1207    # Create Load
1208    region = a.sets['Stamp-1.Stamp_RP']
1209    mymodel.ConcentratedForce(name='Load-Static-Test-Force',
1210        createStepName='Step-Static-Test-Force', region= region,
1211        cf3=200.0, distributionType=UNIFORM, field='',
1212        amplitude=myamplitude.name,
1213        localCsys=da[Stamp_Load_Point_CSYS_id])
1214
1215    # Create Field Output
1216    mymodel.FieldOutputRequest(name='F-Output-Static_Test',
1217        createStepName='Step-Static-Test-Force', variables=(
1218            'MISES', 'U', 'UR','PE'),frequency=100)
1219
1220    # Create History Output
1221    mymodel.historyOutputRequests['H-Output-1'].setValues(
1222        variables=('ALLFD', 'ALLKE', 'ALLSE'))
1223
1224    region = a.allInstances['Stamp-1'].sets['Stamp_RP']
1225    mymodel.HistoryOutputRequest(
1226        name='H-Output-Displacement_Study_Point',
1227        createStepName='Step-Static-Test-Force',
1228        variables=('U1', 'U2', 'U3', 'UR1', 'UR2', 'UR3'),
1229        region=region, sectionPoints=DEFAULT, rebar=EXCLUDE)
1230
1231    # Create Job
1232    myjob = mdb.Job(name= job_name, model='Model-1', description='',
1233        type=ANALYSIS, atTime=None, waitMinutes=0, waitHours=0,
1234        queue=None, memory=90, memoryUnits=PERCENTAGE,
1235        getMemoryFromAnalysis=True, explicitPrecision=SINGLE,
1236        nodalOutputPrecision=SINGLE, echoPrint=OFF,
1237        modelPrint=OFF, contactPrint=OFF, historyPrint=OFF,
1238        userSubroutine='', scratch='', resultsFormat=ODB ,numCpus=2,
1239        numDomains=2, numGPUs=0)
1240
1241    # The program only submits the job, if the user as given
1242    # that information at the start
```

```
1243    if solve_model:
1244
1245        # Monitor the job
1246        monitorManager.addMessageCallback(jobName=job_name,
1247            messageType=ANY_MESSAGE_TYPE, callback=jobMonitorCallback,
1248            userData=None)
1249
1250        # Submit the job and wait for it's completion
1251        myjob.submit()
1252        myjob.waitForCompletion()
1253
1254        # If the job was completed with success, then
1255        # the program will export all of the results
1256        if jobMonitorCallback.job_completed == True:
1257
1258            # Create an animation of the test
1259            create_animations(job_name, path = my_path_1)
1260
1261            # Exclude the fixtures values
1262            exclusion_element_list=[]
1263            exclusion_element_list = Exclusion_Area(job_name)
1264
1265            # Take a screenshot of the last frame of the test
1266            screenshot(job_name,
1267                step_name='Step-Static-Test-Force',
1268                path = my_path_1)
1269
1270            # Export stress results
1271            max_stress = Export_Stress_Results(job_name,
1272                exclusion_element_list,
1273                step_name= 'Step-Static-Test-Force', study_frame = -1,
1274                mat_yield_stress = yield_stress, my_path = my_path_1)
1275
1276            # Export plastic strain
1277            max_plastic_strain = Export_Plastic_Strain_Results(job_name,
1278                exclusion_element_list, study_frame = -1,
1279                step_name='Step-Static-Test-Force',
1280                mat_limit_strain=limit_strain, my_path=my_path_1)
1281
1282            # Export displacement results
1283            max_displacement = Export_nodal_displacement(job_name,
1284                my_path = my_path_1)
1285
1286            # Get the mass of the component
1287            p = mymodel.parts['Base']
1288            myview.setValues(displayedObject=p)
1289            part_mass = p.getMassProperties()['mass']
1290
1291            # Update report card
1292            final_report[report_counter] = {'size':hex_size,
1293                'height':hex_height , 'thickness':hex_thickness,
1294                'test':'Estatico', 'stress':max_stress,
1295                'displacement':max_displacement,
1296                'plastic_strain':max_plastic_strain,
```

```
                'weight':part_mass,
                'path':my_path_2}

            report_counter+=1

    return final_report, report_counter



def Ball_Drop_Test(solve_model, job_name, my_path_1,
    final_report_ball_drop, report_counter,
    hex_size, hex_height, hex_thickness, yield_stress, limit_strain):

    # Internal Variables
    a = mymodel.rootAssembly
    da = a.datums
    converted_report = []
    best_result = "False"

    Ball_Load_Point_CSYS_id = a.features['Ball_Drop_CSYS'].id

    # Initial velocity
    v0 = 3130 #mm/s

    # Step
    mymodel.ExplicitDynamicsStep(name='Step-Ball_Drop',
        previous='Initial',
        timePeriod=0.01, improvedDtMethod=ON)

    # BC: Displacement
    region = a.sets['Ball_Drop_RP']
    local_CSYS = da[Ball_Load_Point_CSYS_id]

    mymodel.DisplacementBC(name='BC-PROJECTILE',
        createStepName='Initial', region=region,
        u1=UNSET, u2=SET, u3=SET,
        ur1=SET, ur2=SET, ur3=SET,
        amplitude=UNSET, distributionType=UNIFORM,
        fieldName='',localCsys= local_CSYS)

    # Initial condition: velocity
    mymodel.Velocity(name='PROJECTILE_VELOCITY', region=region,
        field='', distributionType=MAGNITUDE, velocity1=v0,
        velocity2=0.0,
        velocity3=0.0, omega=0.0)

    mymodel.FieldOutputRequest(name='F-Output-Ball_Drop_Test',
        createStepName='Step-Ball_Drop', variables=('S', 'PE'),
        frequency=100)


    # Job
    myjob = mdb.Job(name= job_name, model=mymodel.name, description='',
```

```
1351        type=ANALYSIS,
1352        atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90,
1353        memoryUnits=PERCENTAGE, explicitPrecision=DOUBLE_PLUS_PACK,
1354        nodalOutputPrecision=SINGLE, echoPrint=OFF, modelPrint=OFF,
1355        contactPrint=OFF, historyPrint=OFF, userSubroutine='',
1356        scratch='',
1357        resultsFormat=ODB, numCpus=2,
1358        numDomains=2, numGPUs=0)
1359
1360    # Submit the job
1361    if solve_model:
1362
1363        # Monitor the job
1364        monitorManager.addMessageCallback(jobName=job_name,
1365        messageType=ANY_MESSAGE_TYPE,
1366        callback=jobMonitorCallback, userData=None)
1367
1368        # Submit the job and wait for it's completion
1369        myjob.submit()
1370        myjob.waitForCompletion()
1371
1372        # Extract the data, if the job was completed successfully
1373        if jobMonitorCallback.job_completed == True:
1374
1375            step_name = 'Step-Ball_Drop'
1376
1377            # Find the critical element
1378            study_element_label = critical_elements()
1379
1380            # Find the critical frame
1381            study_frame =Critical_Frame(job_name, step_name,
1382            study_element_label)
1383
1384            # Create animations of the results
1385            create_animations(job_name, path = my_path_1)
1386
1387            # Create screenshots of the results
1388            screenshot(job_name, step_name ,path = my_path_1)
1389
1390            # Exclude the fixtures values
1391            exclusion_element_list=[]
1392            exclusion_element_list = Exclusion_Area(job_name)
1393
1394            # Extract the data from the odb
1395            max_stress = Export_Stress_Results(job_name,
1396                exclusion_element_list,
1397                step_name, study_frame,
1398                mat_yield_stress = yield_stress,
1399                my_path = my_path_1)
1400
1401            max_plastic_strain = Export_Plastic_Strain_Results(
1402                job_name,
1403                exclusion_element_list,
1404                step_name, study_frame,
```

```
1405                mat_limit_strain=limit_strain,
1406                my_path=my_path_1)
1407
1408            # Get the total mass of the component
1409            p = mymodel.parts['Base']
1410            myview.setValues(displayedObject=p)
1411            part_mass = \
1412               mymodel.parts['Base'].getMassProperties()['mass']
1413
1414            # Update Report Card
1415            final_report_ball_drop[report_counter] = {'size':hex_size,
1416                'height':hex_height , 'thickness':hex_thickness,
1417                'test':'Ball Drop', 'stress':max_stress,
1418                'plastic_strain':max_plastic_strain,
1419                'weight':part_mass, 'status':best_result}
1420
1421            converted_report.append(
1422            final_report_ball_drop[report_counter])
1423            result_confirmation = Result_check(converted_report,
1424                yield_stress, limit_strain)
1425            if result_confirmation:
1426                best_result = "best"
1427            final_report_ball_drop[report_counter]['status'] = \
1428                best_result
1429
1430
1431            report_counter+=1
1432
1433
1434
1435    return final_report_ball_drop, report_counter, best_result
1436
1437
1438
1439
1440 def critical_elements():
1441
1442    # Abaqus variables
1443    a = mymodel.rootAssembly
1444    n = a.nodes
1445    p = mymodel.parts['Base']
1446    study_elements_labels=[]
1447
1448    # Get impactor coordinates
1449    impactor_point = a.sets['Ball_Drop_RP'].referencePoints[0]
1450    impactor_point_RP = a.getCoordinates(impactor_point)
1451
1452    # Get center point x, y and z values
1453    center_x = impactor_point_RP[0]
1454    center_y = impactor_point_RP[1]
1455    center_z = impactor_point_RP[2]
1456
1457    # Get array of nodes to study
1458    analysis_element = p.elements.getByBoundingSphere(\
```

```
1459         center = (center_x, center_y, center_z), radius = 120.)
1460
1461     for element in analysis_element:
1462         study_elements_labels.append(element.label)
1463
1464     return study_elements_labels
1465
1466
1467
1468
1469 # Export all the stress values in a given odb
1470 def Critical_Frame(odb_name, step_name, study_element_label):
1471
1472     # Open the odb
1473     odb = session.openOdb(name=odb_name + '.odb')
1474
1475     # Voting table
1476     voting_table = {}
1477
1478     # Voting session
1479     fieldVarFrames = odb.steps[step_name].frames
1480     frame_index=0
1481     for frame in fieldVarFrames:
1482         for value in frame.fieldOutputs['S'].values:
1483             element_label = value.elementLabel
1484             if element_label in study_element_label:
1485
1486                 if frame_index == 0:
1487                     voting_table[element_label]={'t_max':value.mises,
1488                     'f_max':frame_index}
1489                 else:
1490                     if value.mises >= \
1491                         voting_table[element_label]['t_max']:
1492                         voting_table[element_label]['t_max']=value.mises
1493                         voting_table[element_label]['f_max']=frame_index
1494         frame_index+=1
1495
1496
1497     voting_poll=[]
1498     for element_table in voting_table:
1499         voting_poll.append(voting_table[element_table]['f_max'])
1500     from collections import Counter
1501     occurence_count = Counter(voting_poll)
1502     study_frame = occurence_count.most_common(1)[0][0]
1503
1504     return study_frame
1505
1506
1507
1508
1509
1510
1511
1512 # Export the nodal displacement from the impactor
```

```
1513 def Export_nodal_displacement(odb_name, my_path):
1514
1515     # Open the odb
1516     odb = session.openOdb(name=odb_name + '.odb')
1517
1518     # Extract displacement from the Load Point in different directions
1519     datalist = session.xyDataListFromField(odb=odb,
1520         outputPosition=NODAL,
1521         variable=(('U', NODAL,),), nodeSets=("STAMP-1.STAMP_RP", ))
1522
1523     # Split the tuple in different directions
1524     data_um, data_ux, data_uy, data_uz = datalist
1525
1526     # Initiate lists
1527     t, um = zip(*data_um.data)
1528     t, ux = zip(*data_ux.data)
1529     t, uy = zip(*data_uy.data)
1530     t, uz = zip(*data_uz.data)
1531
1532     # Text file directory
1533     txt_path = my_path + '\\' \
1534         + 'Deformacao no ponto de aplicacao da carga.txt'
1535
1536     # Initiate lists
1537     path_list = [my_path]
1538
1539     # Save the data into a text file
1540     with open(txt_path, 'w') as f:
1541         writer = csv.writer(f, delimiter='\t')
1542         for x in it.izip_longest(t,
1543             um, ux, uy, uz, path_list, fillvalue=''):
1544             writer.writerow(x)
1545
1546     return max(um)
1547
1548
1549
1550
1551 # Export all the stress values in a given odb
1552 def Export_Stress_Results(odb_name, exclusion_element_list,
1553     step_name, study_frame,
1554     mat_yield_stress, my_path):
1555
1556     # Open the odb
1557     odb = session.openOdb(name=odb_name + '.odb')
1558     myview =session.viewports[session.currentViewportName]
1559     myview.setValues(displayedObject=odb)
1560
1561     # Specifying the step and variable
1562     fieldVar = \
1563        odb.steps[step_name].frames[study_frame].fieldOutputs['S']
1564
1565     # Starting variable values
1566     max_value = 0
```

```
1567     number_of_elem_Outspec = 0
1568     number_of_elem_Outspec_with_sec = 0
1569     safety_factor = 0.8
1570     stress_values_list = []
1571
1572     # Read stress values for every node
1573     for value in fieldVar.values:
1574         if not(value.elementLabel in exclusion_element_list):
1575             element_stress = value.mises
1576             stress_values_list.append(element_stress)
1577             if element_stress > (safety_factor * mat_yield_stress):
1578                 number_of_elem_Outspec_with_sec += 1
1579                 if element_stress > mat_yield_stress:
1580                     number_of_elem_Outspec += 1
1581             if element_stress > max_value:
1582                 max_value = element_stress
1583
1584     # Text file directory
1585     title = 'Valores de tensao'
1586     txt_path = my_path + '\\' + title + '.txt'
1587
1588     # Initiate lists
1589     mat_yield_stress_list = [mat_yield_stress]
1590     number_of_elem_Outspec_list = [number_of_elem_Outspec]
1591     number_of_elem_Outspec_with_sec_list=
1592         [number_of_elem_Outspec_with_sec]
1593     safety_factor_list = [safety_factor]
1594     my_path_list = [my_path]
1595
1596     # Save data into a text file
1597     with open(txt_path, 'w') as f:
1598         writer = csv.writer(f, delimiter='\t')
1599         for x in it.izip_longest(
1600             stress_values_list,mat_yield_stress_list,
1601             number_of_elem_Outspec_list,
1602             number_of_elem_Outspec_with_sec_list,
1603             safety_factor_list, my_path_list, fillvalue=''):
1604             writer.writerow(x)
1605
1606     return max_value
1607
1608
1609
1610
1611 # Export all the plastic strain values in a given odb
1612 def Export_Plastic_Strain_Results(odb_name,
1613     exclusion_element_list, step_name, study_frame,
1614     mat_limit_strain, my_path):
1615
1616     # Open the odb
1617     odb = session.openOdb(name=odb_name + '.odb')
1618     myview =session.viewports[session.currentViewportName]
1619     myview.setValues(displayedObject=odb)
1620
```

```
1621     # Specifying the step and variable
1622     fieldVar =\
1623        odb.steps[step_name].frames[study_frame].fieldOutputs['PE']
1624
1625     # Starting variable values
1626     max_value = 0
1627     number_of_elem_Outspec = 0
1628     strain_values_list = []
1629
1630     # Read plastic strain values for every node
1631     for value in fieldVar.values:
1632         if not(value.elementLabel in exclusion_element_list):
1633             element_strain = value.maxInPlanePrincipal
1634             strain_values_list.append(element_strain)
1635             if element_strain > mat_limit_strain:
1636                 number_of_elem_Outspec += 1
1637             if element_strain > max_value:
1638                 max_value = element_strain
1639
1640     # Text file directory
1641     title = 'Valores de plasticidade'
1642     txt_path = my_path + '\\' + title + '.txt'
1643
1644     # Initiate lists
1645     mat_limit_strain_list = [mat_limit_strain]
1646     number_of_elem_Outspec_list = [number_of_elem_Outspec]
1647     my_path_list = [my_path]
1648
1649     # Save data into a text file
1650     with open(txt_path, 'w') as f:
1651         writer = csv.writer(f, delimiter='\t')
1652         for x in it.izip_longest(strain_values_list,
1653             mat_limit_strain_list,
1654             number_of_elem_Outspec_list,
1655             my_path_list, fillvalue=''):
1656             writer.writerow(x)
1657
1658     return max_value
1659
1660
1661
1662 # Get the element labels from the fixture elements
1663 def Exclusion_Area(job_name):
1664     exclusion_element_list = [1]
1665     exclusion_set_name = 'Porta_Agrafos'
1666     exclusion_element_list = Exclusion_elements(exclusion_element_list,
1667         exclusion_set_name, job_name)
1668     exclusion_set_name = 'Porta_Agrafos_2'
1669     exclusion_element_list = Exclusion_elements(exclusion_element_list,
1670         exclusion_set_name, job_name)
1671     exclusion_set_name = 'Porta_Agrafos_8'
1672     exclusion_element_list = Exclusion_elements(exclusion_element_list,
1673         exclusion_set_name, job_name)
1674     exclusion_set_name = 'Porta_Agrafos_9'
```

```
1675      exclusion_element_list = Exclusion_elements(exclusion_element_list,
1676          exclusion_set_name, job_name)
1677      exclusion_set_name = 'Porta_Agrafos_10'
1678      exclusion_element_list = Exclusion_elements(exclusion_element_list,
1679          exclusion_set_name, job_name)
1680
1681      # Internal variables
1682      p = mymodel.parts['Base']
1683      d = p.datums
1684
1685      # Boundary box Coordinates
1686      bounding_box_point_1_id =\
1687          p.features['Exclusion_Bounding_Point_1'].id
1688      bounding_box_point_2_id =\
1689          p.features['Exclusion_Bounding_Point_2'].id
1690      bounding_box_point_1 = p.getCoordinates(d[bounding_box_point_1_id])
1691      bounding_box_point_2 = p.getCoordinates(d[bounding_box_point_2_id])
1692
1693      # Reordering of the boundary box coordinates
1694      min_x_top = min(bounding_box_point_1[0],bounding_box_point_2[0])
1695      min_y_top = min(bounding_box_point_1[1],bounding_box_point_2[1])
1696      min_z_top = min(bounding_box_point_1[2],bounding_box_point_2[2])
1697
1698      max_x_top = max(bounding_box_point_1[0],bounding_box_point_2[0])
1699      max_y_top = max(bounding_box_point_1[1],bounding_box_point_2[1])
1700      max_z_top = max(bounding_box_point_1[2],bounding_box_point_2[2])
1701
1702      exclusion_elements_from_box = p.elements.getByBoundingBox(
1703          min_x_top,min_y_top,
1704          min_z_top,max_x_top,max_y_top,max_z_top)
1705
1706      for element in exclusion_elements_from_box:
1707          exclusion_element_list.append(element.label)
1708          print(element.label)
1709
1710      return exclusion_element_list
1711
1712
1713
1714
1715  # Get the element labels from a set
1716  def Exclusion_elements(exclusion_element_list,
1717      exclusion_set_name, odb_name):
1718
1719      # Open the odb
1720      odb = session.openOdb(name=odb_name + '.odb')
1721      myview =session.viewports[session.currentViewportName]
1722      myview.setValues(displayedObject=odb)
1723
1724      # Internal Variable
1725      p = mymodel.parts['Base']
1726
1727      # Get all element labels inside the set
1728      exclusion_set = p.sets[exclusion_set_name]
```

```
1729     test= []
1730     #for exclusion_node in exclusion_set.nodes:
1731     for exclusion_element in exclusion_set.elements:
1732 
1733         exclusion_element_list.append(exclusion_element.label)
1734         test.append(exclusion_element)
1735 
1736     #p.Set(elements=test, name='Teste de elementos')
1737 
1738     # return the list of element labels
1739     return exclusion_element_list
1740 
1741 
1742 
1743 
1744 # Create a screenshot of the analysis
1745 def screenshot(odbname,step_name, path):
1746 
1747     # Open database
1748     odb = session.openOdb(name=odbname + '.odb')
1749     myview =session.viewports[session.currentViewportName]
1750 
1751     # Set View for the screenshot
1752     myview.setValues(displayedObject=odb)
1753     myview.odbDisplay.commonOptions.setValues(renderStyle=SHADED,)
1754     myview.odbDisplay.display.setValues(plotState=(
1755     CONTOURS_ON_DEF, ))
1756     myview.enableMultipleColors()
1757     myview.setColor(initialColor='#BDBDBD')
1758     cmap=myview.colorMappings['Default']
1759     myview.setColor(colorMapping=cmap)
1760     myview.disableMultipleColors()
1761     myview.view.setValues(session.views['User-1'])
1762 
1763     # Take the screenshot - Displacement
1764     myview.odbDisplay.setPrimaryVariable(
1765         variableLabel='U', outputPosition=NODAL, refinement=(INVARIANT,
1766         'Magnitude'), )
1767 
1768     file_name = path + '\\' + odbname + '_step_'+ \
1769         str(step_name) + '_Displacement'
1770 
1771     session.printToFile(fileName=file_name, format=PNG, canvasObjects=(
1772         myview, ))
1773 
1774     # Take the screenshot - Stress Mises
1775     myview.odbDisplay.setPrimaryVariable(
1776         variableLabel='S', outputPosition=INTEGRATION_POINT,
1777         refinement=(INVARIANT, 'Mises'), )
1778 
1779     file_name = path + '\\' + odbname + '_step_'+ \
1780         str(step_name) + '_Mises'
1781 
1782     session.printToFile(fileName=file_name, format=PNG, canvasObjects=(
```

```
1783        myview, ))
1784
1785    # Take the screenshot - Plastic Strain
1786    myview.odbDisplay.setPrimaryVariable(
1787        variableLabel='PE', outputPosition=INTEGRATION_POINT,
1788        refinement=(INVARIANT, 'Max. In-Plane Principal'), )
1789
1790    file_name = path + '\\' + odbname + '_step_'+ \
1791        str(step_name) + '_PLStrain'
1792
1793    session.printToFile(fileName=file_name, format=PNG, canvasObjects=(
1794        myview, ))
1795
1796
1797
1798
1799 # Create Animations of the simulation
1800 def create_animations(odbname, path):
1801
1802     # Open the correspondant MDB
1803     name =  odbname + '.odb'
1804     odb_object = session.openOdb(name=name)
1805
1806     # Set common viewport properties
1807     myview.setValues(displayedObject = odb_object)
1808     myview.odbDisplay.display.setValues(plotState = (CONTOURS_ON_DEF, ))
1809
1810     # Set specific viewport properties - U
1811     myview.odbDisplay.setPrimaryVariable(
1812         variableLabel='U', outputPosition=NODAL,
1813         refinement=(INVARIANT, 'Magnitude'), )
1814     myview.animationController.setValues(animationType = TIME_HISTORY)
1815     myview.animationController.play(duration = UNLIMITED)
1816     session.imageAnimationOptions.setValues(vpDecorations = OFF,
1817         vpBackground = ON, compass = OFF, timeScale = 1)
1818
1819     # Create a video of the current viewport - U
1820     file_name = path + '\\' + odbname + 'U'
1821     session.writeImageAnimation(fileName = file_name,
1822         format = QUICKTIME , canvasObjects = (myview, ))
1823
1824     # Set specific viewport properties - S_Mises
1825     myview.odbDisplay.setPrimaryVariable(
1826         variableLabel='S', outputPosition=INTEGRATION_POINT,
1827         refinement=(INVARIANT, 'Mises'), )
1828     myview.animationController.setValues(animationType = TIME_HISTORY)
1829     myview.animationController.play(duration = UNLIMITED)
1830     session.imageAnimationOptions.setValues(vpDecorations = OFF,
1831         vpBackground = ON, compass = OFF, timeScale = 1)
1832
1833     # Create a video of the current viewport - S_Mises
1834     file_name = path + '\\' + odbname + '_S_Mises'
1835     session.writeImageAnimation(fileName = file_name,
1836         format = QUICKTIME , canvasObjects = (myview, ))
```

```
1837
1838     # Set specific viewport properties - PL_Strain
1839     myview.odbDisplay.setPrimaryVariable(
1840         variableLabel='PE', outputPosition=INTEGRATION_POINT,
1841         refinement=(INVARIANT, 'maxInPlanePrincipal'), )
1842     myview.animationController.setValues(animationType = TIME_HISTORY)
1843     myview.animationController.play(duration = UNLIMITED)
1844     session.imageAnimationOptions.setValues(vpDecorations = OFF,
1845         vpBackground = ON, compass = OFF, timeScale = 1)
1846
1847     # Create a video of the current viewport - S_Mises
1848     file_name = path + '\\' + odbname + '_PL_Strain'
1849     session.writeImageAnimation(fileName = file_name,
1850         format = QUICKTIME , canvasObjects = (myview, ))
1851
1852     # Stop the animation
1853     myview.animationController.setValues(animationType=NONE)
1854
1855
1856
1857
1858 # Define a callback function jobMonitorCallback()
1859 # That continuously checks the job monitor and if
1860 # The job has concluded successfully
1861 def jobMonitorCallback (job_name, messageType, data, userData):
1862
1863     # Monitor message that tells the prgram that
1864     # the simulation has failed
1865     if((messageType==ABORTED) or (messageType==ERROR)):
1866
1867         # Email message
1868         email_message = "Mas noticias - a simulacao: \n " \
1869             + str(job_name) + "  FALHOU."
1870
1871         # Email subject message
1872         subject_message = "[FAIL] " + str(job_name)
1873
1874         # Send email
1875         sendEmailMessage(subject_message = subject_message ,
1876             email_message = email_message)
1877
1878         # Stop monitoring the job
1879         monitorManager.removeMessageCallback(jobName = job_name,
1880             messageType = ANY_MESSAGE_TYPE, callback=jobMonitorCallback,
1881             userData = None)
1882
1883         # Tag that tells the main function that
1884         # the job has failed
1885         jobMonitorCallback.job_completed = False
1886
1887
1888     elif (messageType == JOB_COMPLETED):
1889
1890         # Email message
```

```
1891         email_message = "Boas noticias - a simulacao: \n " \
1892             + str(job_name) + " CORREU SEM PROBLEMAS "
1893 
1894         # Email subject message
1895         subject_message = "[GOOD] " + str(job_name)
1896 
1897         # Send email
1898         sendEmailMessage(subject_message = subject_message ,
1899             email_message = email_message)
1900 
1901         # Stop monitoring the job
1902         monitorManager.removeMessageCallback(jobName = job_name,
1903             messageType = ANY_MESSAGE_TYPE,
1904             callback = jobMonitorCallback,
1905             userData=None)
1906 
1907         # Tag that tells the main function that
1908         # the job was successfull
1909         jobMonitorCallback.job_completed = True
1910 
1911 
1912 
1913 
1914 
1915 # Define a function sendEmailMessage() to send the email
1916 def sendEmailMessage(subject_message, email_message):
1917 
1918     sender = 'python.script.alarm.thesis@gmail.com'
1919     recipient = 'python.script.alarm.thesis@gmail.com'
1920     subject = subject_message
1921     contents = email_message
1922 
1923     # Create a text/plain message
1924     msg = MIMEText(contents)
1925 
1926     # Specify the email subject, sender and recipient
1927     msg['Subject'] = subject
1928     msg['From'] = sender
1929     msg ['To'] = recipient
1930 
1931     # This is Googles Outgoing Mail (SMTP) server
1932     gmail_smtp_server = 'smtp.gmail.com'
1933 
1934     # This is the port used by Gmail server for outgoing Mail
1935     gmail_smtp_port = 587 # Leave this number as is
1936 
1937     # Gmail username (SMTP username)
1938     gmail_username = 'python.script.alarm.thesis@gmail.com'
1939 
1940     # Gmail password
1941     gmail_password = 'Slbenfica99'
1942 
1943     # Create an SMTP object.
1944     # The SMTP connect() method is called using the name and port
```

```
1945     session = smtplib.SMTP(gmail_smtp_server, gmail_smtp_port)
1946
1947     # Identify ourselves to the ESMTP server using EHLO
1948     session.ehlo()
1949
1950     #Put the SMTP connection in Transport Layer Security (TLS) mode using
1951     #SMTP.starttls() so all SMTP commands that follow will be encrypted
1952     session.starttls()
1953
1954     # Call EHLO again
1955     session.ehlo()
1956
1957     # Login to the server using SMTP.login()
1958     session.login(gmail_username, gmail_password)
1959
1960     # Send the email using SMTP.send,mail()
1961     session.sendmail(sender,[recipient], msg.as_string())
1962
1963     # End the session
1964     session.close()
1965
1966 # sendEmailMessage() definition ends here
1967
1968
1969
1970
1971 # Deletes unnecessary files so that
1972 # it doesn't clough up the disc space
1973 def Delete_files(path):
1974     extension=('odb')
1975     for file in os.listdir(path):
1976         if not (file.endswith(extension)):
1977             os.remove(os.path.join(path, file))
1978
1979
1980
1981
1982 # Report Card analysis
1983 # Checks if a given geometry passes all
1984 # the tests it was subjected to
1985 def Result_check(my_dict, yield_stress, limit_strain):
1986   result_confirmation = True
1987   for geometry in my_dict:
1988
1989     # Passing Criteria for the static test
1990     if geometry['test'] =='Estatico':
1991       if not(float(geometry['stress'])<=yield_stress and
1992          float(geometry['plastic_strain'])<=limit_strain):
1993            result_confirmation = False
1994            break
1995
1996     # Passing Criteria for the Ball Drop test
1997     if geometry['test']=='Ball Drop':
1998         if not(float(geometry['stress'])<=yield_stress  and
```

```
            float(geometry['plastic_strain'])<=limit_strain):
            result_confirmation = False
            break


    # If a new test is added to the program
    # Write it's pass criteria here
  return result_confirmation



# Filter the report card for geometries with the same parameters
def Filter_dict(my_dict, size, height, thickness):

  new_dict=[my_dict[x] for x in my_dict if \
        (my_dict[x]['size']==size) and
        (my_dict[x]['height']==height) and
        (my_dict[x]['thickness']==thickness)]

  return new_dict



# Checks what geometry, from the ones that pass all the tests
# has the least weight amongst them
def Minimum_weight(weight, min_weight, geometry, min_geometry):
  if weight <= min_weight:
    min_weight = weight
    min_geometry = geometry
  return min_weight, min_geometry



# Classifies a geometry  - "Did not pass the tests"/
# "It passed the tests" / "It's the best geometry"
# and saves that data
def Analise_data(final_report, yield_stress, limit_strain):

    # Starting null values
    min_weight = 0
    min_geometry = ''

    # Check if a given geometry passed all it's tests and classify it
    for x in final_report:

        # Starting variables
        weight = float(final_report[x]['weight'])
        size = final_report[x]['size']
        height = final_report[x]['height']
        thickness = final_report[x]['thickness']
        geometry = [size, height, thickness]
```

```
2053
2054            # Filter the dictionary by the geometry
2055            new_dic = Filter_dict(final_report, size,
2056                height, thickness)
2057
2058            # Check if it passed the tests
2059            result_confirmation = Result_check(new_dic,
2060                yield_stress, limit_strain)
2061
2062            # Classification based on performance
2063            if result_confirmation:
2064
2065                # Check if it is the lightest geometry
2066                # that appeared up until now
2067                # and save its value if it is
2068                if min_weight == 0:
2069                    min_weight = weight
2070                    min_geometry = geometry
2071                else:
2072                    min_weight, min_geometry = Minimum_weight(weight,
2073                        min_weight, geometry, min_geometry)
2074
2075                # Green - Passed all the tests
2076                final_report[x]['status']='passed'
2077            else:
2078                # White - Failed the tests
2079                final_report[x]['status']='failed'
2080
2081    # Check wich geometry, from the ones that passed the test, is
2082    # the lightest and classify it as "yellow"
2083    for x in final_report:
2084        weight = float(final_report[x]['weight'])
2085        size = final_report[x]['size']
2086        height = final_report[x]['height']
2087        geometry = [size, height, thickness]
2088        if geometry == min_geometry:
2089            final_report[x]['status']='best'
2090
2091    # Return the new dictionary
2092    return final_report
2093
2094
2095
2096
2097 # Sort the report card by weight
2098 def sort_report(final_report):
2099
2100     sorted_report = OrderedDict(sorted(final_report.items(),
2101        key = lambda x: getitem(x[1], 'weight')))
2102
2103     return sorted_report
2104
2105
2106
```

```
2107
2108 # Save the Report Card data into a text file
2109 def Export_final_report(final_report, my_path):
2110
2111     # Text file directory
2112     title = 'Resultados'
2113     txt_path = my_path + '\\' + title + '.txt'
2114
2115     # Data management - Create lists
2116     size=[]
2117     height=[]
2118     thickness=[]
2119     test=[]
2120     stress=[]
2121     displacement=[]
2122     plastic_strain=[]
2123     weight=[]
2124     status=[]
2125     path=[]
2126     my_path_list = [my_path]
2127
2128     # Fill the lists with the respective values
2129     for x in final_report:
2130         size.append(float(final_report[x]['size']))
2131         height.append(float(final_report[x]['height']))
2132         thickness.append(float(final_report[x]['thickness']))
2133         test.append(final_report[x]['test'])
2134         stress.append(final_report[x]['stress'])
2135         displacement.append(final_report[x]['displacement'])
2136         plastic_strain.append(final_report[x]['plastic_strain'])
2137         weight.append(final_report[x]['weight'])
2138         status.append(final_report[x]['status'])
2139         path.append(final_report[x]['path'])
2140
2141     # Save data into a text file
2142     with open(txt_path, 'w') as f:
2143         writer = csv.writer(f, delimiter='\t')
2144         for x in it.izip_longest(size, height,
2145             thickness, test,
2146             stress,displacement,
2147             plastic_strain, weight, status, path,
2148             my_path_list, fillvalue=''):
2149
2150             writer.writerow(x)
2151
2152
2153
2154
2155 # Get the geometrys that passed the static test
2156 def ball_drop_test_list(my_dict):
2157
2158     new_dict=[my_dict[x] for x in my_dict if (
2159         my_dict[x]['status']=='passed' or
2160         my_dict[x]['status']=='best')]
```

```
2161
2162     return new_dict
2163
2164
2165
2166
2167 def main():
2168     # Global variables
2169     global myview
2170     global mymodel
2171     myview = session.viewports['Viewport: 1']
2172     mymodel = mdb.models['Model-1']
2173     final_report={}
2174     report_counter = 0
2175
2176     # Hex area parameters
2177     A = 43
2178     B = 476
2179     C = 125
2180     D = 160
2181     E = 21.15
2182     F = 81
2183     G = C-E-A
2184     H = B-F-D
2185
2186     # Call the GUI window and get a dictionary with all the simulation
2187 values
2188     GUI_values_main = GUI_Frisos()
2189
2190     # Get a list with all the values that each of the hexagon variables
2191 can have
2192     list_val_lenght = Create_Geometry_Variable_List(
2193         GUI_values_main['Lenght_max_GUI'],
2194         GUI_values_main['Lenght_min_GUI'],
2195         GUI_values_main['Lenght_n_val_GUI'])
2196     list_val_thickness = Create_Geometry_Variable_List(
2197         GUI_values_main['Thickness_max_GUI'],
2198         GUI_values_main['Thickness_min_GUI'],
2199         GUI_values_main['Thickness_n_val_GUI'])
2200     list_val_height = Create_Geometry_Variable_List(
2201         GUI_values_main['Height_max_GUI'],
2202         GUI_values_main['Height_min_GUI'],
2203         GUI_values_main['Height_n_val_GUI'])
2204     test_combinations_dict = Geometry_Test_Combinations(
2205         list_val_lenght, list_val_thickness, list_val_height)
2206
2207     # Create folder structure
2208     print('A criar o sistema de pastas. [1/10]')
2209     dir_path_main = Create_Folder_Structure()
2210     cwd_path_main = Create_Working_Directory_Folder(dir_path_main)
2211     model_path = Open_base_model(cwd_path_main, 'Estatico')
2212
2213     # Set main variables
2214     base_thickness_main = GUI_values_main['base_thickness_GUI']
```

```
2215    if GUI_values_main['simulation_GUI'] == 1:
2216        solve_model = True
2217    else:
2218        solve_model = False
2219
2220    # Loop throught tests
2221    for test_number in range(len(test_combinations_dict)):
2222
2223        # Hexagon parameters
2224        hex_size_main = test_combinations_dict[test_number]['Lenght']
2225        hex_height_main = test_combinations_dict[test_number]['Height']
2226        hex_thickness_main = \
2227            test_combinations_dict[test_number]['Thickness']
2228        hex_circle_height_main = hex_size_main/np.cos(np.radians(30))
2229        hex_lenght_main = hex_size_main*np.tan(np.radians(30))
2230
2231        # Job names for each test
2232        job_name_static_test = ''+str(GUI_values_main['job_name_GUI'])+\
2233            '_L_' + str(hex_size_main).replace(".",",") + \
2234            '_A_' + str(hex_height_main).replace(".",",") + \
2235            '_E_' + str(hex_thickness_main).replace(".",",") + \
2236            '_Static_Test'
2237        job_name_ball_drop = ''+str(GUI_values_main['job_name_GUI']) + \
2238            '_L_' + str(hex_size_main).replace(".",",") + \
2239            '_A_' + str(hex_height_main).replace(".",",") + \
2240            '_E_' + str(hex_thickness_main).replace(".",",") + \
2241            '_Ball_Drop'
2242
2243        mesh_size_main = GUI_values_main['mesh_GUI']
2244
2245        # Create the folder for a specific geometry
2246        test_name = 'L = ' + str (hex_size_main) + \
2247            ' A = ' + str(hex_height_main) + \
2248            ' E = ' + str(hex_thickness_main) + ''
2249
2250        print('A criar o sistema de pastas para ' \
2251            + test_name  + ' (mm). [2/10] ')
2252        path_static_main, path_ball_drop_main = Create_Test_Folder(
2253            hex_size_main, hex_height_main,
2254            hex_thickness_main, dir_path_main)
2255
2256        # Copy Base file
2257        print(test_name + ': A copiar o ficheiro Base. [3/10]')
2258        openMdb(pathName = model_path)
2259        mymodel = mdb.models['Model-1']
2260        myview = session.viewports['Viewport: 1']
2261
2262        # Draw the base Hexagon
2263        print(test_name + ': A criar o hexagono base. [4/10]')
2264        sketch_name = Hex(hex_size_main,hex_height_main)
2265
2266        # Draw the frizes assembly
2267        print(test_name + ': A desenhar os frisos. [5/10]')
2268        Create_Particion_Face(hex_height_main,base_thickness_main)
```

```
2269        Create_Hex_Assembly(hex_size_main, hex_lenght_main,
2270        hex_circle_height_main, hex_thickness_main,
2271        sketch_name, A,B,C,D,E,F,G,H)
2272
2273        # Trim the excess from the frizes
2274        print(test_name + ': A cortar os frisos. [6/10]')
2275        top_bounding_box_point_1_datum = 'Top_Bounding_Box_P1'
2276        top_bounding_box_point_2_datum = 'Top_Bounding_Box_P2'
2277        bottom_bounding_box_point_1_datum = 'Bottom_Bounding_Box_P1'
2278        bottom_bounding_box_point_2_datum = 'Bottom_Bounding_Box_P2'
2279        particion_base_set = 'Particion_Base'
2280        sketch_plane_bottom = 'Sketch_Plane_Bottom'
2281        Delete_Set(sketch_plane_bottom)
2282        Trim_Hex_Assembly(top_bounding_box_point_1_datum,
2283            top_bounding_box_point_2_datum)
2284        Trim_Hex_Assembly(bottom_bounding_box_point_1_datum,
2285            bottom_bounding_box_point_2_datum)
2286        Delete_Set(particion_base_set)
2287
2288        # Get material data from the database
2289        print(test_name + \
2290            ': A ler a base de dados de materiais. [7/10]')
2291        plasticity_curve_tuple, yield_stress, modulus, \
2292            poisson, density, \
2293            limit_strain = Read_True_Stress_Strain_Values(0.01)
2294
2295        # Create material sections
2296        print(test_name + ': A atribuir o material e seccoes. [8/10]')
2297        Atribute_thickness(density, modulus,
2298            poisson, hex_thickness_main, plasticity_curve_tuple)
2299
2300        # Create Mesh
2301        print(test_name + ': A construir a malha. [9/10]')
2302        Mesh_Part(mesh_size_main)
2303
2304        # Switch directory to static test
2305        print(test_name + ': Switching current directory')
2306        switch_working_directory(cwd_path_main)
2307
2308        # Create Interactions for the static test
2309        print(test_name + ': A realizar o ensaio estatico. [10/10]')
2310        Interaction_Static_Test()
2311
2312        # Static test
2313        final_report, report_counter = Static_test(solve_model,
2314            job_name_static_test,
2315            path_static_main, path_ball_drop_main, final_report,
2316            report_counter, hex_size_main,
2317            hex_thickness_main, hex_height_main,
2318            yield_stress, limit_strain)
2319
2320
2321        # If the user set it to only make the model
2322        # The program will only make the first geometry,
```

```
2323            # And not delete any files
2324            if not(solve_model):
2325                break
2326
2327            # Open a new file
2328            Mdb()
2329
2330            # Delete files
2331            Delete_files(cwd_path_main)
2332
2333
2334        # Analise the final report data
2335        final_report = Analise_data(final_report,
2336            yield_stress, limit_strain)
2337
2338        # Sort the final report by weight
2339        final_report = sort_report(final_report)
2340
2341        # Export the final report data to a file
2342        Export_final_report(final_report, dir_path_main)
2343
2344        # Create a Hashmap for the Ball Drop
2345        ball_drop_list = ball_drop_test_list(final_report)
2346
2347        # Internal Variables
2348        final_report_ball_drop={}
2349        report_counter = 0
2350
2351        # Open base model
2352        model_path = Open_base_model(cwd_path_main, 'Ball Drop')
2353
2354        for test_number in ball_drop_list:
2355
2356            # Hexagon parameters
2357            hex_size_main = test_number['size']
2358            hex_height_main = test_number['height']
2359            hex_thickness_main = test_number['thickness']
2360            hex_circle_height_main = hex_size_main/np.cos(np.radians(30))
2361            hex_lenght_main = hex_size_main*np.tan(np.radians(30))
2362
2363            # Job names for each test
2364            job_name_ball_drop = ''+str(GUI_values_main['job_name_GUI']) + \
2365                '_L_' + str(hex_size_main).replace(".",",") + \
2366                '_A_' + str(hex_height_main).replace(".",",") + \
2367                '_E_' + str(hex_thickness_main).replace(".",",") + \
2368                '_Ball_Drop'
2369
2370            # Create the mesh
2371            mesh_size_main = GUI_values_main['mesh_GUI']
2372
2373            # Create the folder for a specific geometry
2374            test_name = 'L = ' + str (hex_size_main) + \
2375                ' A = ' + str(hex_height_main) + \
2376                ' E = ' + str(hex_thickness_main) + ''
```

```
2377        path_ball_drop_main = test_number['path']
2378
2379        # Copy Base file
2380        print(test_name + ': A copiar o ficheiro Base. [2/9]')
2381        openMdb(pathName = model_path)
2382        mymodel = mdb.models['Model-1']
2383        myview = session.viewports['Viewport: 1']
2384
2385        # Draw the base Hexagon
2386        print(test_name + ': A criar o hexagono base. [3/9]')
2387        sketch_name = Hex(hex_size_main,hex_height_main)
2388
2389        # Draw the frizes assembly
2390        print(test_name + ': A desenhar os frisos. [4/9]')
2391        Create_Particion_Face(hex_height_main,base_thickness_main)
2392        Create_Hex_Assembly(hex_size_main, hex_lenght_main,
2393            hex_circle_height_main, hex_thickness_main,
2394            sketch_name, A,B,C,D,E,F,G,H)
2395
2396        # Trim the excess from the frizes
2397        print(test_name + ': A cortar os frisos. [5/9]')
2398        top_bounding_box_point_1_datum = 'Top_Bounding_Box_P1'
2399        top_bounding_box_point_2_datum = 'Top_Bounding_Box_P2'
2400        bottom_bounding_box_point_1_datum = 'Bottom_Bounding_Box_P1'
2401        bottom_bounding_box_point_2_datum = 'Bottom_Bounding_Box_P2'
2402        particion_base_set = 'Particion_Base'
2403        sketch_plane_bottom = 'Sketch_Plane_Bottom'
2404        Delete_Set(sketch_plane_bottom)
2405        Trim_Hex_Assembly(top_bounding_box_point_1_datum,
2406            top_bounding_box_point_2_datum)
2407        Trim_Hex_Assembly(bottom_bounding_box_point_1_datum,
2408            bottom_bounding_box_point_2_datum)
2409        Delete_Set(particion_base_set)
2410
2411        # Get material data from the database
2412        print(test_name + ': A ler a base de dados de materiais. [6/9]')
2413        plasticity_curve_tuple, yield_stress, modulus, \
2414            poisson, density, limit_strain = \
2415                Read_True_Stress_Strain_Values(16.6)
2416
2417        # Create material sections
2418        print(test_name + ': A atribuir o material e seccoes. [7/9]')
2419        Atribute_thickness(density, modulus, poisson,
2420            hex_thickness_main, plasticity_curve_tuple)
2421
2422        # Create Mesh
2423        print(test_name + ': A construir a malha. [8/9]')
2424        Mesh_Part(mesh_size_main)
2425
2426        # Switch directory to static test
2427        print(test_name + ': Switching current directory')
2428        switch_working_directory(cwd_path_main)
2429
2430        # Create Interactions for the static test
```

```
2431            Interaction_Ball_Drop_Test()
2432
2433            # Ball Drop test
2434            print(test_name + ': A realizar o ensaio do ball drop. [9/9]')
2435            final_report_ball_drop, report_counter, best_result = \
2436                Ball_Drop_Test(solve_model,
2437                job_name_ball_drop,
2438                path_ball_drop_main, final_report_ball_drop,
2439                report_counter, hex_size_main,
2440                hex_height_main, hex_thickness_main,
2441                yield_stress, limit_strain)
2442
2443            # If the user set it to only make the model
2444            # The program will only make the first geometry,
2445            # And not delete any files
2446            if best_result=='best':
2447                break
2448
2449            # Close abaqus current window
2450            Mdb()
2451
2452            # Delete files
2453            Delete_files(cwd_path_main)
2454
2455        #Search_txt_files(dir_path_main)
2456        combined_dict = Combine_dict(final_report_ball_drop,ball_drop_list)
2457        Results(combined_dict, dir_path_main)
2458
2459
2460
2461 # Combine the Static test Hash Map and the
2462 # Ball drop Hash Map into one
2463 def Combine_dict(final_report_ball_drop, ball_drop_list):
2464
2465        # Internal Variables
2466        combined_dict = {}
2467        combined_index = 0
2468
2469        # Add the values from the static test Hash Map
2470        # To the new combined Hash Map
2471        for static_test in ball_drop_list:
2472            combined_dict[combined_index] = {}
2473            combined_dict[combined_index]['size']= \
2474                static_test['size']
2475            combined_dict[combined_index]['height']= \
2476                static_test['height']
2477            combined_dict[combined_index]['thickness']= \
2478                static_test['thickness']
2479            combined_dict[combined_index]['test']= \
2480                static_test['test']
2481            combined_dict[combined_index]['stress']= \
2482                static_test['stress']
2483            combined_dict[combined_index]['displacement']= \
2484                static_test['displacement']
```

```
2485            combined_dict[combined_index]['plastic_strain']= \
2486                static_test['plastic_strain']
2487            combined_dict[combined_index]['weight']= \
2488                static_test['weight']
2489            combined_dict[combined_index]['status']='NC'
2490
2491            combined_index+=1
2492            size=static_test['size']
2493            height=static_test['height']
2494            thickness=static_test['thickness']
2495
2496            # Filter the Ball Drop Hash Map so that the
2497            # only geometry that appears, is the same as the one
2498            # in the static test
2499            filtered_dict = Filter_dict(final_report_ball_drop,
2500                size, height, thickness)
2501
2502            # if the Hash Map value for the Ball Drop isn't Null
2503            # That the value will be added into the new combined
2504            # Hash Map
2505            if filtered_dict:
2506                combined_dict[combined_index] = {}
2507                combined_dict[combined_index]['size']= \
2508                    filtered_dict[0]['size']
2509                combined_dict[combined_index]['height']= \
2510                    filtered_dict[0]['height']
2511                combined_dict[combined_index]['thickness']= \
2512                    filtered_dict[0]['thickness']
2513                combined_dict[combined_index]['test']= \
2514                    filtered_dict[0]['test']
2515                combined_dict[combined_index]['stress']= \
2516                    filtered_dict[0]['stress']
2517                combined_dict[combined_index]['displacement']='NC'
2518                combined_dict[combined_index]['plastic_strain']= \
2519                    filtered_dict[0]['plastic_strain']
2520                combined_dict[combined_index]['weight']= \
2521                    filtered_dict[0]['weight']
2522                combined_dict[combined_index]['status']= \
2523                    filtered_dict[0]['status']
2524
2525                combined_index+=1
2526
2527    # Return the new Combined Hash Map
2528    return combined_dict
2529
2530
2531
2532
2533 # Save the Report Card data into a text file
2534 def Results(combined_dict, my_path):
2535
2536    # Text file directory
2537    title = 'Resultados_finais'
2538    txt_path = my_path + '\\' + title + '.txt'
```

```
2539
2540      # Data management - Create lists
2541      size=[]
2542      height=[]
2543      thickness=[]
2544      test=[]
2545      stress=[]
2546      displacement=[]
2547      plastic_strain=[]
2548      weight=[]
2549      color=[]
2550      for x in combined_dict:
2551          size.append(combined_dict[x]['size'])
2552          height.append(combined_dict[x]['height'])
2553          thickness.append(combined_dict[x]['thickness'])
2554          test.append(combined_dict[x]['test'])
2555          stress.append(combined_dict[x]['stress'])
2556          displacement.append(combined_dict[x]['displacement'])
2557          plastic_strain.append(combined_dict[x]['plastic_strain'])
2558          weight.append(combined_dict[x]['weight'])
2559          color.append(combined_dict[x]['status'])
2560
2570      # Save data into a text file
2571      with open(txt_path, 'w') as f:
2572          writer = csv.writer(f, delimiter='\t')
2572          for x in it.izip_longest(size, height,
2573              thickness, test,
2574              stress,displacement,
2575              plastic_strain, weight, color, fillvalue=''):
2576              writer.writerow(x)
2577
2578
2579
2580
2581 main()
2582
2583
```

6.4 Source code do software de pós-processamento

```
#from tkinter import ON
from abaqus import *
from abaqusConstants import *
from caeModules import *
import numpy as np
from abaqus import getInputs
import regionToolset
import smtplib
from email.mime.text import MIMEText
import math
import shutil
from collections import OrderedDict
import itertools as it
from itertools import islice
from itertools import product
import material
from jobMessage import*
import sys
import os
import signal
import win32api
import win32process
import win32con
from matplotlib import colors
from matplotlib import cm
import scipy as sp
import matplotlib as mpl
mpl.use("TkAgg")
#from matplotlib import pyplot as plt
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import matplotlib.lines as mlines
from matplotlib.pyplot import ion
from matplotlib.pyplot import figure
import visualization
import animation
import csv



# Search for the text files in the study directory
def Search_txt_files(root_path):
    list_of_files = os.listdir(root_path)

    # Search all files in the folder directory
    for entry in list_of_files:
        if (entry != 'Diretoria de Trabalho') and \
            (entry != 'Resultados.txt') and \
                (entry != 'Resultados_finais.txt'):
            sub_path = os.path.join(root_path,entry)
            list_of_files_2 = os.listdir(sub_path)

            # Search all files in the geometry directory
```

```
55              for folder_file in list_of_files_2:
56                  sub_path_2 = os.path.join(sub_path, folder_file)
57                  list_of_files_3 = os.listdir(sub_path_2)
58
59                  # Initiate plot
60                  if folder_file == 'Teste Estatico':
61                      fig, axes = plt.subplots(nrows=3, ncols=1,
62                          figsize=(8.3,11.7), dpi=80)
63                  else:
64                      fig, axes = plt.subplots(nrows=2, ncols=1,
65                          figsize=(8.3,11.7), dpi=80)
66
67                  # Define Plot Title
68                  fig.suptitle(entry, x= 0.7, y = 0.99, fontsize=12)
69
70                  # Search all files in the test directory
71                  for txt_file in list_of_files_3:
72
73                      # Read the files and plot the data
74                      if 'Valores de tensao' in txt_file:
75                          file_path = os.path.join(sub_path_2, txt_file,)
76                          Read_stress_test_values(file_path,
77                              fig, axes, folder_file)
78                      elif 'Deformacao no ponto de aplicacao da carga' \
79                           in txt_file:
80                          file_path = os.path.join(sub_path_2, txt_file)
81                          Read_displacement_test_values(file_path,
82                              fig, axes)
83                      elif 'Valores de plasticidade' in txt_file:
84                          file_path = os.path.join(sub_path_2, txt_file,)
85                          Read_plastic_strain_test_values(file_path,
86                              fig, axes, folder_file)
87
88          # Analise the data from the report card
89          # of the static tests
90          # and split it into blocks of 40 values
91          elif entry == 'Resultados.txt':
92              # Create Report card directory
93              path = os.path.join(root_path,entry)
94
95              # Read Report Card Values
96              final_report = Read_report_card(path)
97
98              # Split the dictionary in multiple pages
99              final_report = Split_dict(final_report)
100
101             # Plot the Report card
102             iterator = 0
103             for n in final_report:
104                 Final_report_card(n, iterator,
105                     root_path, file_name='Resultados')
106                 iterator+=1
107
108         # Analise the data from the report card
```

```
109         # and split it into blocks of 40 values
110         elif entry == 'Resultados_finais.txt':
111             # Create Report card directory
112             path = os.path.join(root_path,entry)
113 
114             # Read Report Card Values
115             final_report = Read_final_results(path)
116 
117             # Split the dictionary in multiple pages
118             final_report = Split_dict(final_report)
119 
120             # Plot the Report card
121             iterator = 0
122             for n in final_report:
123                 Final_report_card(n, iterator,
124                     root_path, file_name='Resultados_finais')
125                 iterator+=1
126 
127 
128 
129 
130 def Read_final_results(file):
131 
132     # Open the text file with the test data
133     f=open(file,"r")
134     lines=f.readlines()
135 
136     # retrieve the values of the file as a dictionary
137     final_report={}
138     counter = 1
139 
140     for x in lines:
141         final_report[counter]={'size':x.split('       ')[0],
142             'height':x.split(' ')[1],
143             'thickness':x.split('     ')[2],
144             'test':x.split('   ')[3],
145             'stress':float(x.split('   ')[4]),
146             'plastic_strain':float(x.split('  ')[6]),
147             'weight':float(x.split('   ')[7]),
148             'status':x.split(' ')[8]}
149 
150         if final_report[counter]['test']=='Ball Drop':
151             final_report[counter]['displacement']=\
152                 x.split('      ')[5]
153         else:
154             final_report[counter]['displacement']=\
155                 float(x.split('       ')[5])
156         counter +=1
157 
158 
159     return final_report
160 
161 
162 
```

```
163
164 # Read the Stress values data base
165 def Read_stress_test_values(file, fig, axes,
166     folder_file):
167
168     # Open the text file with the test data
169     f=open(file,"r")
170     lines=f.readlines()
171
172     # Initiate the lists
173     stress_values_list=[]
174
175     # Retrieve each of the variables
176     # from the txt file
177     mat_yield_stress = (float(
178         lines[0].split('       ')[1]))
179     number_of_elem_Outspec = (float(
180         lines[0].split('       ')[2]))
181     number_of_elem_Outspec_with_sec = (float(
182         lines[0].split('       ')[3]))
183     safety_factor = (float(
184         lines[0].split('       ')[4]))
185
186     my_path = (lines[0].split('       ')[5])
187     size = len(my_path)
188     my_path = my_path[:size - 2]
189
190     # Retrieve each line of the stress
191     # collum as a value for the stress list
192     for x in lines:
193         stress_values_list.append(float(
194             x.split('  ')[0]))
195
196     # Close the txt file
197     f.close()
198
199     # Plot the data
200     Stress_Graphs(stress_values_list,
201         mat_yield_stress,
202         number_of_elem_Outspec,
203         number_of_elem_Outspec_with_sec,
204         safety_factor, my_path ,fig, axes,
205         folder_file)
206
207
208
209
210 def Read_plastic_strain_test_values(file, fig,
211     axes, folder_file):
212
213     # Open the text file with the test data
214     f=open(file,"r")
215     lines=f.readlines()
216
```

```
217     # Initiate the lists
218     strain_values_list=[]
219 
220     # Retrieve each of the variables
221     # from the txt file
222     mat_limit_strain = (float(
223         lines[0].split('       ')[1]))
224     number_of_elem_Outspec = (
225         float(lines[0].split(' ')[2]))
226 
227     my_path = (lines[0].split('       ')[3])
228     size = len(my_path)
229     my_path = my_path[:size - 2]
230 
231     # Retrieve each line of the stress
232     # collum as a value for the stress list
233     for x in lines:
234         strain_values_list.append(float(
235             x.split('  ')[0]))
236 
237     # Close the txt file
238     f.close()
239 
240     # Plot the data
241     Plastic_Strain_Graphs(strain_values_list,
242         mat_limit_strain, number_of_elem_Outspec,
243          my_path,fig, axes, folder_file)
244 
245 
246 
247 # Read the displacement data base
248 def Read_displacement_test_values(file, fig, axes):
249 
250     # Open the text file with the test data
251     f=open(file,"r")
252     lines=f.readlines()
253 
254     # Initiate the lists
255     t_list=[]
256     um_list=[]
257     ux_list=[]
258     uy_list=[]
259     uz_list=[]
260 
261     # Retrieve each of the variables
262     # from the txt file
263     my_path = (lines[0].split('       ')[5])
264     size = len(my_path)
265     my_path = my_path[:size - 2]
266 
267     # Retrieve the values from the text file
268     # as a list
269     for x in lines:
270         t_list.append(float(x.split('  ')[0]))
```

```
271         um_list.append(float(x.split('')[1]))
272         ux_list.append(float(x.split('')[2]))
273         uy_list.append(float(x.split('')[3]))
274         uz_list.append(float(x.split('')[4]))
275 
276     # Close the txt file
277     f.close()
278 
279     # Plot the data
280     Plot_nodal_displacement(t_list, um_list,
281         ux_list,uy_list, uz_list,
282         my_path, fig, axes)
283 
284 
285 
286 # Read the report_card database
287 def Read_report_card(file):
288 
289     # Open the text file with the test data
290     f=open(file,"r")
291     lines=f.readlines()
292 
293     # Retrieve each of the variables
294     # from the txt file
295     my_path = (lines[0].split('       ')[7])
296     size = len(my_path)
297     my_path = my_path[:size - 2]
298 
299     # retrieve the values of the file as a dictionary
300     final_report={}
301     counter = 1
302 
303     for x in lines:
304         final_report[counter] = {'size':x.split('     ')[0],
305             'height':x.split(' ')[1],
306             'thickness':x.split('     ')[2],
307             'test':x.split('   ')[3],
308             'stress':float(x.split('  ')[4]),
309             'displacement':float(x.split('    ')[5]),
310             'plastic_strain':float(x.split('  ')[6]),
311             'weight':float(x.split('  ')[7]),
312             'status':x.split(' ')[8]}
313         counter +=1
314 
315     return final_report
316 
317 
318 
319 # Split the report-card HashMap
320 # into pages with a max of 40 values
321 def Split_dict(data, SIZE=40):
322     itd = iter(data)
323     for i in range(0, len(data), SIZE):
324         yield {k:data[k] for k in islice(itd, SIZE)}
```

```
325
326
327
328 # Plot stress values
329 def Stress_Graphs(stress_values_list,
330     mat_yield_stress, n_nodes_above,
331     n_nodes_above_perc, safety_factor,
332     my_path, fig, axes, folder_file):
333
334     # Bin width
335     binwidth = 2
336
337     if folder_file == 'Teste Estatico':
338         mysubplot=plt.subplot(311)
339     else:
340         mysubplot=plt.subplot(211)
341
342     # Get Stress values and number of nodes for each one
343     number_of_nodes_hist, stress_values_hist, patches =  \
344         plt.hist(stress_values_list,
345         bins=np.arange(min(stress_values_list),
346         max(stress_values_list) +  \
347         binwidth, binwidth))
348
349     # Set label names
350     plt.xlabel('Tensao (MPa)', size = 8)
351     plt.ylabel('Numero de elem.', size = 8)
352     plt.title('Tensao por elem.', size = 10)
353     plt.grid(True)
354
355     # Colouring parameters
356     yield_fraction = stress_values_hist/mat_yield_stress
357     norm_Green = colors.Normalize(
358         vmin=yield_fraction.min(),
359         vmax=safety_factor)
360     norm_Orange = colors.Normalize(vmin=safety_factor,
361         vmax= 1.0)
362     norm_Red = colors.Normalize(vmin=1.0,
363         vmax=yield_fraction.max())
364
365     # Colouring
366     for thisfrac, thispatch in zip(yield_fraction, patches):
367         if thisfrac <= safety_factor:
368           color = plt.cm.Greens(norm_Green(thisfrac))
369         elif thisfrac <= 1.0:
370           color = plt.cm.YlOrBr(norm_Orange(thisfrac))
371         else:
372           color = plt.cm.Reds(norm_Red(thisfrac))
373         thispatch.set_facecolor(color)
374
375     # Label of yield stress
376     plt.axvline(x=mat_yield_stress, color='r',
377         label='Tensao de cedencia: ' + \
378         str(mat_yield_stress) + ' (MPa)\nN de elem. acima: ' + \
```

```
379         str(n_nodes_above))
380
381     plt.axvline(x=safety_factor*mat_yield_stress, color='g',
382         label= str(safety_factor*100)+'% tensao de cedencia:\n ' + \
383         str(0.8*mat_yield_stress) +'(MPa)\nN de elem. acima: ' + \
384         str(n_nodes_above_perc))
385
386     # Plot legend
387     mysubplot.legend(bbox_to_anchor=(1.0, 1.0),
388         loc='upper left',
389         labelspacing = 1.0, fontsize = 8)
390
391     # Plot and save the graph
392     fig.tight_layout()
393     fig.savefig(my_path + '\Resultados.pdf')
394     plt.close()
395
396
397
398
399 def Plastic_Strain_Graphs(strain_values_list,
400     mat_limit_strain, n_elements_above,
401     my_path, fig, axes, folder_file):
402
403     # Bin width
404     binwidth = 0.01
405     if folder_file == 'Teste Estatico':
406         mysubplot=plt.subplot(312)
407     else:
408         mysubplot=plt.subplot(212)
409
410     # Get Stress values and number of nodes for each one
411     number_of_elem_hist, stress_values_hist, patches =  \
412         plt.hist(strain_values_list,
413         bins=np.arange(min(strain_values_list),
414         max(strain_values_list) + binwidth, binwidth))
415
416     # Set label names
417     plt.xlabel('Deformacao Plastica', size = 8)
418     plt.ylabel('Numero de elem.', size = 8)
419     plt.title('Deformacao por elem.', size = 10)
420     plt.grid(True)
421
422     # Colouring parameters
423     yield_fraction = stress_values_hist/mat_limit_strain
424     norm_Green = colors.Normalize(vmin=yield_fraction.min(),
425         vmax=1)
426     norm_Red = colors.Normalize(vmin=1,
427         vmax=yield_fraction.max())
428
429     # Colouring
430     for thisfrac, thispatch in zip(yield_fraction, patches):
431         if thisfrac <= 1:
432           color = plt.cm.Greens(norm_Green(thisfrac+0.2))
```

```
433         else:
434           color = plt.cm.Reds(norm_Red(thisfrac))
435         thispatch.set_facecolor(color)
436 
437 
438     plt.axvline(x=mat_limit_strain, color='r',
439         label='Limite de deformacao plastica: ' + \
440         str(mat_limit_strain) + ' (MPa)\nN de elem. acima: ' + \
441         str(n_elements_above))
442 
443     mysubplot.legend(bbox_to_anchor=(1.0, 1.0), loc='upper left',
444         labelspacing = 1.0, fontsize = 8)
445 
446 
447 
448 
449 
450 
451 def Plot_nodal_displacement(t, um, ux, uy, uz,
452     my_path, fig, axes):
453 
454     # Convert list to array
455     t = np.asarray(t)
456     um = np.asarray(ux)
457     ux = np.asarray(ux)
458     uy = np.asarray(uy)
459     uz = np.asarray(uz)
460 
461 
462     # Limit deflection
463     lim_deflection = 2.0
464 
465     # Subplot number
466     myplot = plt.subplot(313)
467     ax = axes[2]
468 
469     # Create Graphs
470     # Change Graph Line Representattion (Global)
471     mpl.rcParams['lines.linestyle'] = '--'
472 
473     # Define the subplot matrix, number of rows,
474     # collums and the size of the figure
475     myplot.plot(t, ux, color = 'b', label='U1')
476     myplot.plot(t, uy, color = 'orange', label='U2')
477     myplot.plot(t, uz, color = 'g', label='U3')
478     myplot.plot(t, um, color = 'k', label='U')
479     plt.xlabel('Tempo (s)', size = 8)
480     plt.ylabel('Deslocamento (mm)', size = 8)
481     plt.title('Deslocamento do ponto de aplicacao da carga',
482         size = 10)
483     plt.grid(True)
484 
485     # Max values
486     # Displacement - U1-max
```

```
487     if np.amax(ux)*-1 < np.amin(ux):
488         max_ux = np.amax(ux)
489     else :
490         max_ux = np.amin(ux)
491     t_max_ux = t[np.where(ux == max_ux)]
492 
493     # Plot displacement - U1-max
494     myplot.plot(t_max_ux[0], max_ux, color = 'b',
495         label = 'Valor maximo de U1\nU1 = '+ \
496         "{:.3e}".format(max_ux) + ' (mm)\nt = '\
497         + "{:.3e}".format(t_max_ux[0]) + ' (s)', \
498         marker = 'P', ms = 10)
499 
500     # Displacement - U2-max
501     if np.amax(uy)*-1 < np.amin(uy):
502         max_uy = np.amax(uy)
503     else :
504         max_uy = np.amin(uy)
505     t_max_uy = t[np.where(uy == max_uy)]
506 
507     # Plot displacement - U2-max
508     myplot.plot(t_max_uy[0], max_uy, color = 'orange',
509         label = 'Valor maximo de U2\nU2 = ' + \
510         "{:.3e}".format(max_uy) + ' (mm)\nt = ' + \
511         "{:.3e}".format(t_max_uy[0]) + ' (s)',
512         marker = 'P', ms = 10)
513 
514 
515     # Displacement - U3-max
516     if np.amax(uz)*-1 < np.amin(uz):
517         max_uz = np.amax(uz)
518     else :
519         max_uz = np.amin(uz)
520     t_max_uz = t[np.where(uz == max_uz)]
521 
522     # Plot displacement - U3-max
523     myplot.plot(t_max_uz[0], max_uz, color = 'g',
524         label = 'Valor maximo de U3\nU3 = ' +
525         "{:.3e}".format(max_uz) + ' (mm)\nt = ' + \
526         "{:.3e}".format(t_max_uz[0]) + ' (s)' ,
527         marker = 'P', ms = 10)
528 
529 
530     # Displacement - U-max
531     if np.amax(um)*-1 < np.amin(um):
532         max_um = np.amax(um)
533     else :
534         max_um = np.amin(um)
535     t_max_um = t[np.where(um == max_um)]
536 
537     # Plot displacement - U-max
538     myplot.plot(t_max_um[0], max_um, color = 'k',
539         label = 'Valor maximo de U\nU = ' + \
540         "{:.3e}".format(max_um)+' (mm)\nt = ' + \
```

```
541         "{:.3e}".format(t_max_um[0]) + ' (s)',
542         marker = 'P', ms = 10)
543
544
545     # Plot the Graph and Legend
546     myplot.legend(bbox_to_anchor=(1.0, 1), loc='upper left',
547         labelspacing = 1.0, fontsize = 8)
548     fig.tight_layout()
549
550
551
552
553 # Plot the table with the final results
554 def Final_report_card(final_report, iterator, my_path, file_name):
555
556     # Data management
557     size=[]
558     height=[]
559     thickness=[]
560     test=[]
561     stress=[]
562     displacement=[]
563     plastic_strain=[]
564     weight=[]
565     for x in final_report:
566         size.append(final_report[x]['size'])
567         height.append(final_report[x]['height'])
568         thickness.append(final_report[x]['thickness'])
569         test.append(final_report[x]['test'])
570         stress.append("{:.3e}".format(final_report[x]['stress']))
571         weight.append("{:.3e}".format(final_report[x]['weight']))
572         plastic_strain.append("{:.3e}"\
573             .format(final_report[x]['plastic_strain']))
574
575         if final_report[x]['displacement']=='NC':
576             displacement.append(final_report[x]['displacement'])
577         else:
578             displacement.append("{:.3e}"\
579                 .format(final_report[x]['displacement']))
580
581
582     # Plot Table
583     fig, ax =plt.subplots(1, 1, figsize=(8.3,11.7), dpi=80)
584     fig.suptitle('Resumo dos resultados do estudo dos frisos',
585         fontsize=14)
586     data=[size,height,thickness,test,stress,
587         displacement,plastic_strain,weight]
588     table_data = np.transpose(data)
589     column_labels=['Largura(mm)','Altura(mm)','Esp.(mm)',
590         'Teste','Tens.(MPa)','Def.(mm)',
591         'Def.Pl', 'Peso(Ton)']
592     ax.axis('tight')
593     ax.axis('off')
594     my_table=ax.table(cellText=table_data,
```

```
595         colLabels=column_labels,
596         loc="center", cellLoc = "center")
597 
598 
599     # Coloring of the geometries that passed the test,
600     # and the lightest one out of them
601     for x in final_report:
602         status = final_report[x]['status']
603         #print(status[:-2])
604         if final_report[x]['status'] == 'passed':
605             face_color = "#daf7d4"
606             my_table[x-iterator*40,0].set_facecolor(face_color)
607             my_table[x-iterator*40,1].set_facecolor(face_color)
608             my_table[x-iterator*40,2].set_facecolor(face_color)
609             my_table[x-iterator*40,3].set_facecolor(face_color)
610             my_table[x-iterator*40,4].set_facecolor(face_color)
611             my_table[x-iterator*40,5].set_facecolor(face_color)
612             my_table[x-iterator*40,6].set_facecolor(face_color)
613             my_table[x-iterator*40,7].set_facecolor(face_color)
614 
615         elif final_report[x]['status'] == 'best':
616             face_color = "#ffe700"
617             my_table[x-iterator*40,0].set_facecolor(face_color)
618             my_table[x-iterator*40,1].set_facecolor(face_color)
619             my_table[x-iterator*40,2].set_facecolor(face_color)
620             my_table[x-iterator*40,3].set_facecolor(face_color)
621             my_table[x-iterator*40,4].set_facecolor(face_color)
622             my_table[x-iterator*40,5].set_facecolor(face_color)
623             my_table[x-iterator*40,6].set_facecolor(face_color)
624             my_table[x-iterator*40,7].set_facecolor(face_color)
625 
626         elif status[:-2] == 'best':
627             face_color = "#ffe700"
628             my_table[x-iterator*40,0].set_facecolor(face_color)
629             my_table[x-iterator*40,1].set_facecolor(face_color)
630             my_table[x-iterator*40,2].set_facecolor(face_color)
631             my_table[x-iterator*40,3].set_facecolor(face_color)
632             my_table[x-iterator*40,4].set_facecolor(face_color)
633             my_table[x-iterator*40,5].set_facecolor(face_color)
634             my_table[x-iterator*40,6].set_facecolor(face_color)
635             my_table[x-iterator*40,7].set_facecolor(face_color)
636 
637 
638             my_table[x-iterator*40-1,0].set_facecolor(face_color)
639             my_table[x-iterator*40-1,1].set_facecolor(face_color)
640             my_table[x-iterator*40-1,2].set_facecolor(face_color)
641             my_table[x-iterator*40-1,3].set_facecolor(face_color)
642             my_table[x-iterator*40-1,4].set_facecolor(face_color)
643             my_table[x-iterator*40-1,5].set_facecolor(face_color)
644             my_table[x-iterator*40-1,6].set_facecolor(face_color)
645             my_table[x-iterator*40-1,7].set_facecolor(face_color)
646 
647 
648     # Font size and scaling
```

```
649     my_table.auto_set_font_size(False)
650     my_table.set_fontsize(10)
651     my_table.scale(1, 1)
652     fig.tight_layout()
653 
654     # Legend handels and labels
655     green_patch = mpatches.Patch(color="#daf7d4",
656         label='Passaram nos ensaios')
657 
658     yellow_patch = mpatches.Patch(color="#ffe700",
659         label='Geometria mais leve e que passou os ensaios')
660 
661     # Plot Legend
662     plt.legend(handles = [green_patch, yellow_patch],
663         bbox_to_anchor=(0.9, 0.1), loc='upper right',
664         ncol= 2, labelspacing = 1.0, fontsize = 10)
665 
666     # Save the table
667     my_path = os.getcwd()
668     fig.savefig(my_path + '\\'+ str(file_name) + \
669         '_' + str(iterator+1) + '.pdf')
670 
671 
672 
673 def main():
674     dir_path_main = os.getcwd()
675     Search_txt_files(dir_path_main)
676 
677 
678 
679 
680 main ()
```