Victor Siqueira Chaim

Simulação e otimização mecânica de polímeros para uso em protetores faciais

Brasil

2021

Victor Siqueira Chaim

Simulação e otimização mecânica de polímeros para uso em protetores faciais

Trabalho de conclusão de curso de Engenharia Mecatrônica para obtenção do título de Engenheiro na Escola Politécnica da Universidade de São Paulo

Universidade de São Paulo – USP

Escola Politécnica da USP

Bacharelado

Orientador: Rafael Traldi Moura

Coorientador: Larissa Driemeier

Brasil

2021

Catalogação-na-publicação

Victor Siqueira Chaim

Simulação e otimização mecânica de polímeros para uso em protetores faciais/ Victor Siqueira Chaim. – Brasil, 2021-, 203 p.

Orientador: Rafael Traldi Moura

Trabalho de Conclusão de Curso – Universidade de São Paulo – USP
Escola Politécnica da USP
Bacharelado, 2021.

1. Simulação Numérica 2. Otimização 3. Elementos Finitos 4. Modelização

---

Victor Siqueira Chaim

Simulação e otimização mecânica de polímeros para uso em protetores faciais

Trabalho de conclusão de curso de Engenharia Mecatrônica para obtenção do título de Engenheiro na Escola Politécnica da Universidade de São Paulo

Rafael Traldi Moura
Orientador

Brasil
2021

Este trabalho é dedicado à todos que sempre
estiveram ao meu lado durante todos esses anos

Agradecimentos

Agradeço em primeiro lugar a Deus, que sempre me escutou nos momentos mais difíceis e sempre me deu forças para prosseguir. Agradeço à minha família e amigos por sempre estarem lá durante todos os momentos, proporcionando memórias incríveis. Agradeço ao meu professor orientador por todo o auxílio e orientação. Agradeço à Universidade por todas as oportunidades e experiências que ela pôde me proporcionar. Agradeço a todos que fizeram parte direta ou indiretamente deste trabalho. Muito Obrigado

"Não vos amoldeis às estruturas deste mundo,
mas transformai-vos pela renovação da mente,
a fim de distinguir qual é a vontade de Deus:
o que é bom, o que Lhe é agradável, o que é perfeito.
(Bíblia Sagrada, Romanos 12, 2)

Resumo

Protetores faciais são dispositivos recentes e ainda pouco utilizados no âmbito dos esportes para atletas que se recuperam de lesões. Apesar dos primeiros estudos demonstrarem uma boa eficiência desses objetos, ainda faltam estudos acerca do comportamento dos materiais do dispositivo sob situações de impacto. Uma técnica muito útil que pode ser utilizada para estudos aprofundados é a Simulação Numérica baseada no Método dos Elementos Finitos (MEF), que consiste um uma modelização de sistemas mecânicos. É possível se implementar modelos de material como viscoelasticidade e hiperelasticidade em conjunto com um modelo MEF para se atingir uma situação muito próxima da realidade. Tendo isso em vista, estuda-se a técnica acima com o objetivo calibrar e simular o material utilizado em ensaios experimentais realizados anteriormente, otimizar os parâmetros do material, validar o modelo e explicar os fenômenos envolvidos através das analises das simulações.

Palavras-chaves: Elementos Finitos, Otimização, Protetores Faciais, Modelo de Material

Abstract

Sports Face Masks are recent devices and yet little used in the sports scope for athletes that are recovering from injuries. Despite the early studies have shown a good eficiency of these objects, deep studies are still missing concerning the behavior of theese devices's materials under impact situations. A technique very useful which can be utilizes for those deep studies is the Numerial Simulation based on Finite Elements Method (FEM), which consists in a modelization of mechanical systems. It is possible to implement material models such as viscoelasticity and hiperelasticity together with a FEM model to reach a situation very close from the reality one. With this in mind, the technique presented above are studied aiming the correct material calibration and the simulation of experimental essays made in past studies, optimization of the material parameters, validation of the model and understanding of the phenomena involved using the simulation analysis.

Key-words: Finite Elementes, Optimization, Face masks, Material Model

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

MEF	Método dos Elementos Finitos
LHC	Latin Hypercube
DOE	Design of Experiment
EVA	Ethylene Vinyl Acetate
CAE	Computer Aided Engineering
SNRD	Smooth Noise-Robust Differentiation
MDF	Método das Diferenças Finitas
MDC	Método das Diferenças Centrais
LER	Lesões por Esforço Repetitivo

Lista de símbolos

T_f	Temperatura de fusão
T_g	Temperatura de transição vítrea
E	Módulo de Young
σ_y	Limite elástico do material
ν	Coefficiente de Poisson
ϵ	Deformação do Material
$\dot{\epsilon}$	Taxa de deformação
g_i	Módulo de Relaxamento em Cisalhamento na Série de Prony
k_i	Módulo de Relaxamento Volumétrico na Série de Prony
τ_i	Constante de tempo de Relaxamento Série de Prony
μ_0	Módulo de Cisalhamento inicial do material no modelo de Arruda-Boyce
λ_m	Alongamento com o qual as cadeias do polímero travam no modelo de Aruda-Boyce

Sumário

Trabalho de Conclusão de Curso - Mecatrônica

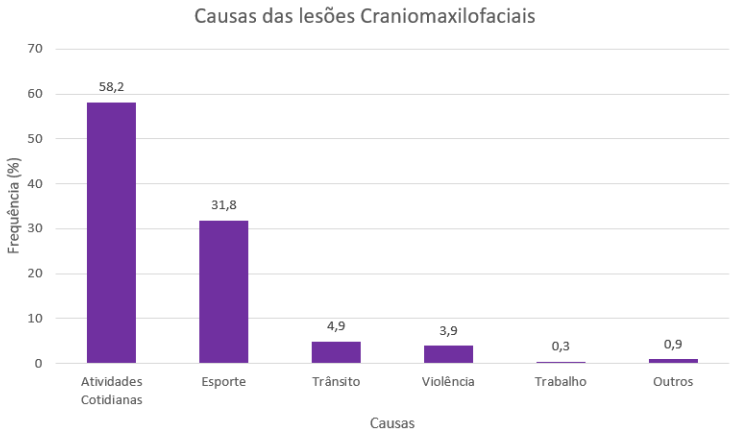
1 Introdução

1.1 Motivação

Lesões físicas são comuns no cotidiano das pessoas. Segundo a Organização Mundial da Saúde (OMS) as principais causas de traumatismos tanto leves quanto graves são acidentes de trânsito e domésticos (OMS, 2012). Esses acidentes podem levar à danos desde luxações e câimbras até mesmo fraturas da coluna e distensões musculares e em casos mais raros, à morte (BUCHOLZ et al., 1979). Além disso, com o avanço da industrialização e padronização do trabalho, as lesões por esforço repetitivo (LER) vem crescendo, sendo até mesmo classificadas como doença ocupacional (BRASIL, 2009).

Dentre as lesões, os traumas faciais podem ser considerados uma das agressões mais devastadoras (WULKAN; JR; BOTTER, 2005). As sequelas devidas à um ferimento no rosto e/ou caixa craniana têm potencial de envolver danos emocionais e funcionais (SILVA et al., 2011). Mesmo sendo mais raras que outras lesões, devido à facilidade de consequências graves, uma contusão nessa região precisa de muita atenção e cuidados especiais. A figura abaixo, adaptada de (GASSNER et al., 2004) mostra as principais causas desse tipo de trauma em crianças.

Figura 1 – Histograma que mostra as principais causas de lesões craniomaxilofaciais em crianças



Fonte: adaptado de (GASSNER et al., 2004)

O gráfico acima mostra que o âmbito dos esportes representa boa parte dos acidentes com injúrias craniomaxilofaciais em crianças. Este fato porém não se restringe a essa faixa etária, tornando o esporte uma das grades causas de danos à caixa craniana (MALADIÈRE

et al., 2001) também em adultos (SILVA et al., 2011). Assim, este tipo de lesão representa de 4% a 41% dos traumas no esporte (BLACK et al., 2017). Destes, 26% são resultados de impactos com o solo ou com outros jogadores (FROMMER; BHATT, 2016). Nesse contexto, estes tipos de impacto são comuns nos chamados "esportes de contato", e podem levar ao afastamento do atleta, causando prejuízos em sua carreira (MOREIRA; GENTIL; OLIVEIRA, 2003).

Os atletas que sofrem traumas faciais, em geral, estão mais propensos a fraturar os ossos nasais e zigomáticos (DELILBASI et al., 2004). Esses ossos estão localizados em regiões proeminentes da face e são mais suscetíveis a impactos (MOUROUZIS; KOUMOURA, 2005). A intensidade dos danos se deve pelas altas taxas de deformação na região facial, durante uma situação de impacto (DYER, 1999). Assim, para valores elevados desta grandeza, o tecido ósseo acaba não resistindo e se rompe, acarretando na fratura.

Após uma ocorrência de fratura craniomaxilofacial em geral são necessários procedimentos médicos para garantir a boa recuperação da vítima.

1.2 Revisão Bibliográfica

1.2.1 Histórico

Existem diversas soluções para a recuperação de um atleta que sofreu um trauma facial, como por exemplo as cirurgias (ALMEIDA, 2012) e os procedimentos de reabilitação. Estas podem levar ao afastamento do mesmo e impactar negativamente em seu desempenho físico, pessoal e financeiro. Isso, por sua vez, também conduzirá à prejuízos aos clubes que patrocinam o atleta. Dessa forma, a solução de cirurgias só se mostra eficiente em casos extremos de lesões.

Uma outra possível solução alternativa para o tratamento de lesões maxilofaciais são as chamadas barras de arco de Erich. Há relatos de utilização deste tipo de equipamento desde a Primeira Guerra Mundial (QURESHI et al., 2016) e se trata portanto de um método clássico. Neste tipo de abordagem, barras como as mostradas na figura 2 são instaladas na maxila e na mandíbula do paciente que sofreu o trauma para se fixar o local da fratura. Apesar de ser um método eficaz e versátil, a utilização de barras de arco de Erich para a correção de lesões maxilofaciais pode acarretar em problemas de lesão devido ao contato com o aço, prolongamento do tempo de recuperação e até prejudicar a higiene bucal (QURESHI et al., 2016). Além disso, este método possui aplicações relativamente restritas, não podendo ser aplicado em indivíduos que sofreram traumas craniomaxilofaciais, apenas em casos com lesões envolvendo mandíbula e maxila.

Figura 2 – Exemplo de barra de arco de Erich

Retirado de (KIRK et al., 2016)

Por fim, um último possível tratamento para lesões maxilofaciais é a utilização de parafusos auto-roscantes. Estes equipamentos foram desenvolvidos em 1989 (JONES, 1999) e tratam-se de parafusos que são implantados na mandíbula e/ou maxilar do paciente para auxiliar na fixação das partes fraturadas. Esta técnica diminui muito os riscos de lesões em decorrência de sua própria implantação, o que demonstra uma melhora em relação ao método anterior. Entretanto, utilizar estes parafusos pode em certos casos prejudicar as raízes dos dentes do paciente (JONES, 1999), o que pode acarretar em tempos de recuperação mais longos, prejudicando o desempenho dos atletas.

Além disso, nenhum dos métodos citados acima previne possíveis traumas futuros no atleta. Isso quer dizer que os métodos acima são corretivos de lesões, mas não protegem a integridade do atleta quanto à novas lesões, expondo assim o atleta a possíveis novos traumas no esporte.

1.2.2 Estado da Arte

Uma solução alternativa aos métodos mostrados acima para evitar o afastamento prolongado do atleta, além de prevenir possíveis acidentes futuros, é a utilização dos protetores faciais. Estes são dispositivos que auxiliam na recuperação do tecido ósseo fraturado de maneira eficiente (COTO, 2009). Os materiais mais comuns de fabricação dos protetores são polímeros e materiais derivados da fibra de carbono. Entretanto, não existe uma padronização dos protetores faciais, quanto à forma, à estrutura, e aos materiais para sua confecção (COTO, 2009).

Para se garantir a segurança dos usuários das máscaras, algumas características precisam ser satisfeitas. A máscara deve ser rígida o suficiente para se proteger o nariz e ao mesmo tempo conseguir amortecer a maior quantidade possível de energia mecânica enquanto minimiza a transmissão de força para os ossos zigomáticos. Esse efeito pode ser alcançado combinando-se as características mecânicas de polímeros rígidos e flexíveis. A

função do material rígido é direcionar a força do nariz para os pontos de apoio (como por exemplo os ossos zigomático e/ou da testa).

Já o material flexível possui o papel de melhorar o contato do protetor facial com o rosto e absorver energia através do processo de atenuação que acontece em maior evidência em altas frequências. Dessa forma, o material flexível funciona analogamente à um filtro passa-baixas mecânico devido à sua característica viscoelástica. O processo de atenuação de energia está ligado às deformações sofridas pelo material flexível, que dissipam parte da energia mecânica que iria para o rosto do usuário.

Figura 3 – Exemplo de protetor facial produzido pela FOUSP

Fonte: imagem retirada de (DIAS,)

Uma possível matéria prima para os protetores faciais para atletas esportivos que respeita as condições acima é o EVA (acetato-vinilo de etileno, em português). Este polímero apresenta uma boa resistência mecânica e se apresenta como um material leve e confortável de usar. Além disso, o custo de se fabricar e trabalhar com EVA é muito inferior quando comparado aos outros polímeros (COTO, 2009). Entretanto, faltam testes experimentais que caracterizem mecanicamente o comportamento do polímero quando submetido às condições de impacto facial.

Além do EVA, um outro material de fácil acesso e alta capacidade de absorção de ondas de impacto mecânico é o silicone médico (QING et al., 2009). Devido às características viscoelásticas, este composto se apresenta como um bom substituto da pele humana, tanto na parte de propagação da onda mecânica quanto na atenuação da mesma, podendo assim ser estudado como análogo à esse órgão humano.

Ainda não há estudos aprofundados sobre o modelo de material e as geometrias necessárias para estudos de atenuação de energia em protetores faciais. É preciso primeiramente definir os materiais e dimensões adequados e que satisfaça os requisitos de segurança

de utilização. Desse modo, encontrando-se um modelo ideal de material, pode-se realizar uma calibração de um corpo de prova que servirá de base para os experimentos.

Um possível desafio de projeto é fazer estudos virtuais sobre o corpo de prova calibrado e obter dados sobre o comportamento do EVA nas aplicações esportivas descritas acima. É possível se obter a combinação ótimas para minimizar as altas taxas de deformação com base nos resultados obtidos utilizando uma combinação de método dos elementos finitos com técnicas de otimização.

Além disso, conforme consta em (SANTOIEMMA, 2018), os resultados dos ensaios experimentais de impacto não se mostraram muito intuitivos, pois a menor transmissão de força para o osso foi com EVA flexível, sem a presença de um material rígido. Para explicar este fenômeno, é preciso um modelo numérico com material que representa muito bem as características viscoelásticas, a fim de se reproduzir com maior exatidão os fenômenos mostrados no ensaio. Para isso, é preciso se realizar um processo de otimização numérica.

2 Objetivo

O objetivo principal deste projeto é estudar simulações numéricas e otimizações, com auxílio de programas de Otimização e Elementos Finitos, dos diferentes tipos de EVA a fim de se cumprir os seguintes critérios:

- Calibrar os materiais EVA Flexível e EVA Rígido de acordo com os ensaios de compressão feitos experimentalmente, comparando-se as curvas de força x tempo,para se obter os parâmetros viscoelásticos do material através das análises numéricas
- Calibrar os materiais que representam o Osso e a Pele de acordo com os ensaios de impacto feitos experimentalmente, também visando-se obter os parâmetros de material de ambos
- Compreender os fenômenos envolvidos e explorar os resultados modificando-se os parâmetros utilizados

3 Embasamento Teórico

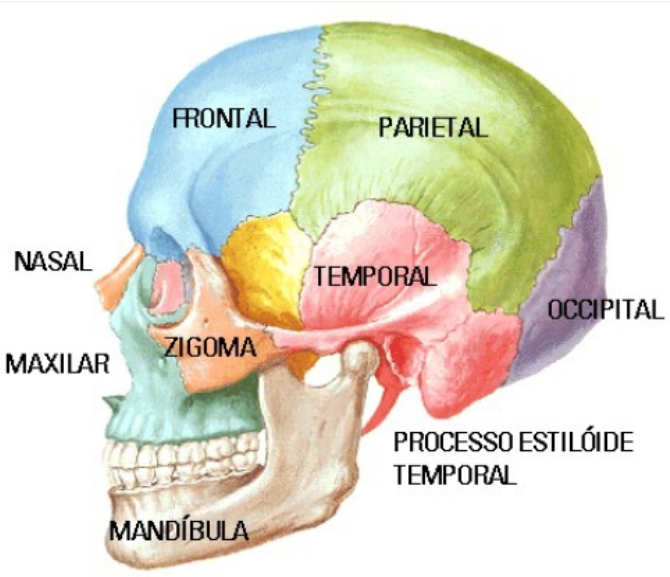
3.1 Revisão Médica

3.1.1 Traumas Faciais

O corpo humano está suscetível a diversos traumas decorrentes das mais distintas fontes. No que diz respeito às fraturas ósseas, os trauma faciais se tornaram relativamente maiores nas quatro últimas décadas. Este fato está associado ao aumento no número de acidentes automobilísticos, da violência urbana e da participação em esportes de altas taxas de impacto (MONTOVANI et al., 2006) (BRAUNSTEIN, 1957). A etiologia dos traumas faciais é heterogênea, ou seja, se relaciona com algumas características da população estudada.

Além disso, existe uma variedade de traumas faciais quanto a complexidade da fratura. Em um trauma ósseo facial, é possível que apenas um dos ossos da face se frature ou então uma ocorrência de uma fratura multi-complexa, onde dois ou mais conjuntos de ossos se rompe.

Figura 4 – Os ossos do crânio humano e os grandes complexos ósseos presentes



Fonte: retirado de (PRAZERES,)

Em acidentes automobilísticos e fraturas decorrentes de violência, os principais conjuntos atingidos são a mandíbula, com 31,92% de ocorrência e o zigomático (25,3%). Estes tipos de lesão podem ser gravíssimos, devido às altas taxas de deformação presentes no impacto (MOURA, 2013), com chance de acometer outros ossos do corpo humano, como os ossos cervicais. Caso tais vértebras sejam atingidas, as consequências podem ser severas, como a paralisia total ou parcial dos movimentos (BRAUNSTEIN, 1957).

3.1.2 Relação com o esporte

Apesar dos traumas faciais graves estarem frequentemente mais relacionados aos acidentes automobilísticos (BRAUNSTEIN, 1957), não é possível desprezar as lesões que se relacionam com o esporte. Este tipo de fratura prejudica não só o indivíduo acidentado como também todos os outros membros do time, comitê esportivo, comissão técnica, clube, etc.

De acordo com (CARROLL et al., 1995), os principais conjuntos ósseos que são afetados por lesões relacionadas aos esportes são o nasal (56% dos casos), zigomático/maxilar (23% dos casos) e os ossos que compõe a mandíbula (13%). Desse modo, conforme mostrado na figura 2, por estarem em regiões proeminentes da face, estes ossos estão mais suscetíveis aos traumas. Quanto aos esportes, os casos registrados estão mais concentrados em esportes como o futebol de campo (64% dos casos) e o basquete (13,6%), sendo caracterizados como esportes de contato e alto impacto (MOUROUZIS; KOUMOURA, 2005).

As lesões presentes nos esportes e que envolvem os conjuntos ósseos citados acima são associadas a choques entre jogadores, impactos com o solo e com objetos esportivos (CARROLL et al., 1995). Justamente por estarem em regiões proeminentes da face, os conjuntos ósseos zigomático, nasal e da mandíbula são os mais fraturados em esportes de alto impacto.

Quando um atleta se acidenta e adquire lesões maxilofaciais, o tratamento é diverso, variando de implantes, cirurgias corretivas, e até mesmo próteses dentárias (PICCININNI et al., 2017). Algumas soluções, como a cirurgia, não são recomendadas para alguns casos mais simples de fraturas maxilofaciais. Este tipo de procedimento envolve o afastamento do atleta, acarretando em perda de desempenho e prejuízos para à clubes e patrocinadores (CARROLL et al., 1995). Além disso, é possível que o atleta não se recupere da maneira esperada e seja impedido de praticar o esporte por mais tempo. Dessa forma, as máscaras de proteção nasal se apresentam como solução para evitar uma refratura do tecido lesionado, evitando a perda de desempenho e que o atleta deixe de competir em sua modalidade.

3.1.3 Revisão médica de fraturas de Le Fort

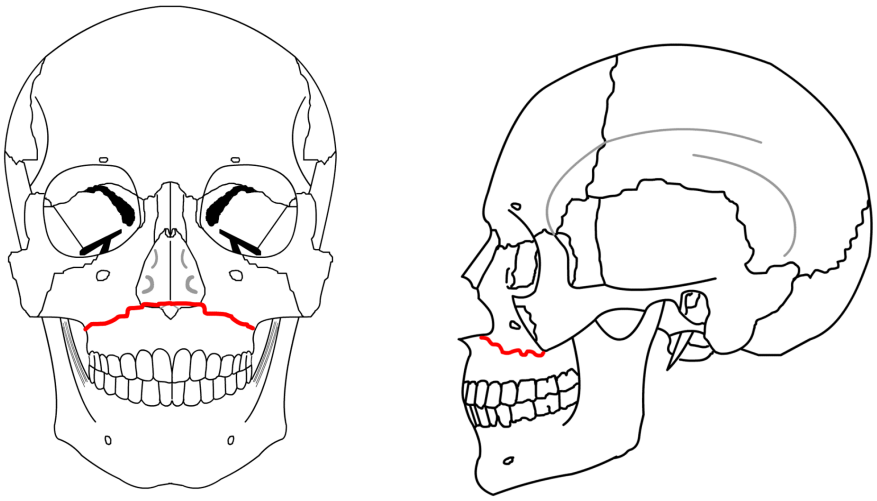
Os principais traumas cranianos, dentais e orofaciais sofridos pelos seres humanos foram catalogados e descritos por René Le Fort no início do século XX. A classificação de Le Fort, embora um pouco ultrapassada, ainda é utilizada para analisar ocorrências nos dias de hoje (MACMILLAN et al., 2018)(SHAH; ROURE, 2019).

A classificação de Le Fort divide as fraturas em três grupos, de acordo com as imagens abaixo:

- Fraturas I de Le Fort: se caracterizam como lesões transversas, envolvendo a maxila,

separando-a do palatino (FORT, 1901).

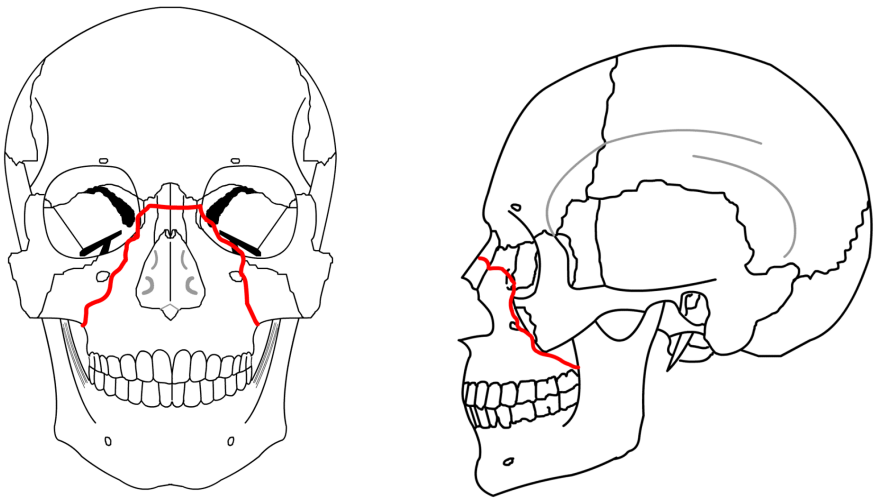
Figura 5 – Figura representativa de lesões do tipo I de Le Fort



Fonte: retiradas de (FUNK,)

- Fraturas II de Le Fort: também chamadas de "Fraturas Piramidais"são aquelas que atravessam os conjuntos ósseos nasal e orbital (FORT, 1901).

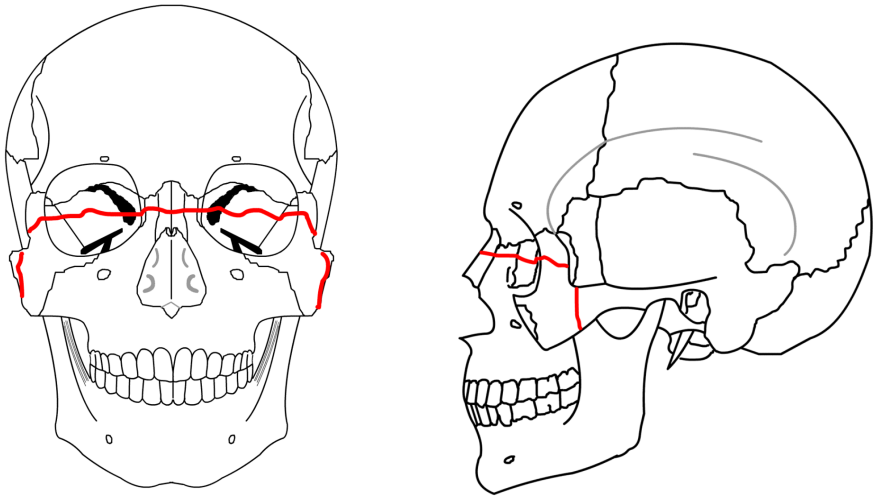
Figura 6 – Figura representativa de lesões do tipo II de Le Fort



Fonte: retiradas de (FUNK,)

- Fraturas III de Le Fort: conhecida como "Disjunção crânio-facial"corresponde às lesões que atravessam a frente da maxila e envolve as suturas zigomaticofrontal, maxilofrontal nasofrontal, os assoalhos das órbitas, a etmóide e o esfenóide (FORT, 1901).

Figura 7 – Figura representativa de lesões do tipo III de Le Fort



Fonte: retiradas de (FUNK,)

Todas os tipos de lesões acima podem ser de diversas gravidades e acontecer de vários modos. Além disso, os estudos feitos por Le Fort representam 10-20% dos traumas faciais (MACMILLAN et al., 2018)(SHAH; ROURE, 2019) sendo considerado um método ultrapassado e simplista, mas que continua a ser largamente aplicado pela medicina. Entretanto, dentro dos esportes com grandes quantidades de colisões, as fraturas orofaciais mais frequentes são as fraturas segmentadas e as do tipo I de Le Fort (PICCININNI et al., 2017), existindo casos registrados dos dois outros tipos também. Assim, o estudo de tais fraturas se torna indispensável para compreender quais áreas são as mais afetadas pelas situações de impacto as quais os atletas estão submetidos.

As fraturas de Le Fort fornecem uma base sólida para os estudos envolvendo os protetores faciais, uma vez que direcionam os objetivos de proteção da máscara para os pontos mais suscetíveis de fraturas. Dessa forma, um bom caminho para que o protetor facial diminua a chance de fratura (ou até mesmo piora de uma fratura pré-existente) de um atleta é assegurando-se que as áreas frágeis do rosto estejam devidamente reforçadas com auxílio deste dispositivo.

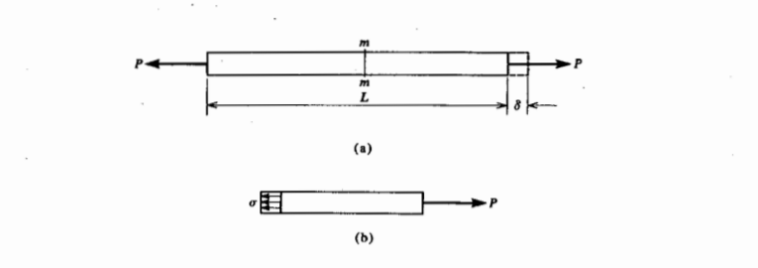
3.2 Revisão de Resistência dos materiais

Conforme dito acima, uma máscara protetora para atletas de alto desempenho deve ser feita a partir de um material propício. Este material deve ser capaz de resistir a choques leves e medianos e ser capaz de amenizar as altas taxas de deformação, responsáveis pela danificação do tecido ósseo. Assim, a teoria de resistência dos materiais torna-se uma ferramenta essencial

3.2.1 Conceitos básicos

Dois conceito básico e extremamente importantes são os conceitos de tensão e deformação. Baseando-se na literatura científica de (TIMOSHENKO, 1984), a tensão e a deformação podem ser definidas através de um exemplo. A figura 8 mostra uma barra homogênea de seção constante e eixo retilíneo.

Figura 8 – Exemplo de barra sujeita à cargas



Fonte: (TIMOSHENKO, 1984)

Nesta ilustração, a barra encontra-se sujeita à forças axiais de módulo P. Pode-se ver facilmente pela figura que a resultante das forças na barra em questão é dada pelo produto da área transversal A pelo valor de tensão, chegando-se em:

$$\sigma = \frac{P}{A}; \quad \epsilon = \frac{\delta}{L} \tag{1}$$

A equação acima nos fornece o módulo da tensão uniforme em uma barra prismática. Este conceito de tensão, embora não represente grande parte das situações de engenharia, é de suma importância para se entender o comportamento de um corpo quando sujeito à uma força.

Além disso, um outro conceito relevante é o de alongamento específico, também chamado de deformação. Para esta compreensão, define-se como deformação a equação que envolve ϵ acima, sendo caracterizada pela razão entre a variação de comprimento e o comprimento inicial do corpo sujeito à esforços. Como é possível perceber, ϵ trata-se de um valor adimensional, podendo ser definida como a equação (1) para o caso de deformação homogênea da barra.

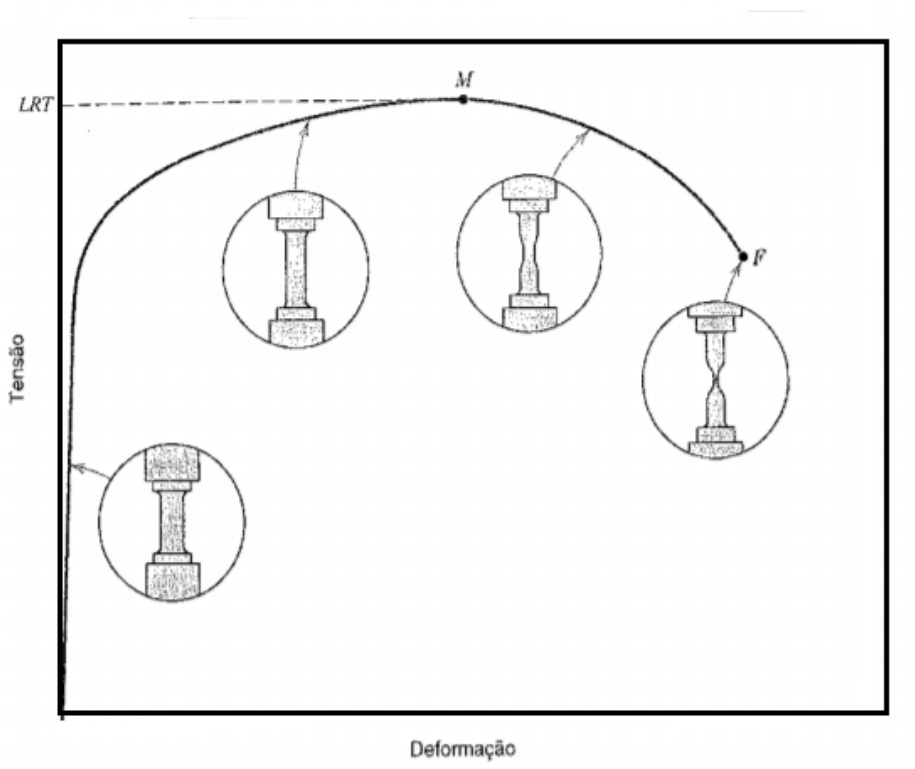
Pode-se definir também a taxa de deformação como a variação temporal da deformação apresentada acima. A taxa de deformação é a parte simétrica do tensor gradiente de velocidades (MOURA, 2013). Assim, temos que a taxa de deformação é dada por:

$$\dot{\epsilon} = \frac{d\epsilon}{dt} \tag{2}$$

3.2.2 Elasticidade e Plasticidade dos Materiais

Com os dois conceitos explicitados acima, é possível construir gráficos de Tensão por Deformação (Tensão x Deformação) para cada material estudado. Como algumas propriedades são dependentes dos materiais (CALLISTER, 2000), cada objeto apresentará uma curva diferente. Um exemplo de curva de Tensão x Deformação é apresentada a seguir:

Figura 9 – Exemplo de gráfico que retrata tensão e deformação em corpos de prova



Adaptado de (CALLISTER, 2000)

Conforme é possível visualizar na figura acima, partindo-se de uma situação de carregamento nulo e aumentando-se a tensão no corpo, a curva tende a crescer. Até um certo limite, o comportamento da curva é aproximadamente linear, caracterizando a primeira região da curva, ou região elástica do material. Nesta região da curva, o material se deforma com carregamento, mas quando a carga aplicada é liberada, o corpo retorna a sua forma original, não ocorrendo deformações permanentes. É possível, nesta região linear, descrever uma lei que rege o comportamento elástico unidimensional dos materiais, chamada de lei de Hooke, apresentada por:

$$\sigma = E\epsilon \tag{3}$$

De modo que o coeficiente E trata-se de uma constante chamada de módulo de elasticidade de Young (ou módulo de elasticidade), e corresponde ao valor numérico da

inclinação da curva mostrada na figura 9 no trecho elástico. Nas atividades práticas de engenharia, nem sempre a região elástica se comporta como uma reta de coeficiente E, mas esta modelização consiste em uma boa aproximação da situação real (KALPAKJIAN, 1984).

Entretanto, a região elástica é muito estreita quando comparada com a outra região, denominada de região plástica. A tensão na qual o material começa a se deformar plasticamente é chamada de Tensão de Escoamento, comumente representada por σ_e.

Dessa forma, na região plástica, o material, ao ser submetido a tensões maiores, passa a se deformar permanentemente. Isso significa que, ao se liberar o carregamento, o objeto não retorna a sua forma original. Durante o processo de plastificação, dois pontos são de importante análise. Um deles é o ponto de máxima tensão (chamado de M na figura 9) que representa o começo do *estiramento* ou empescoçamento. O outro é a tensão de ruptura (ponto F), que representa a quebra do objeto.

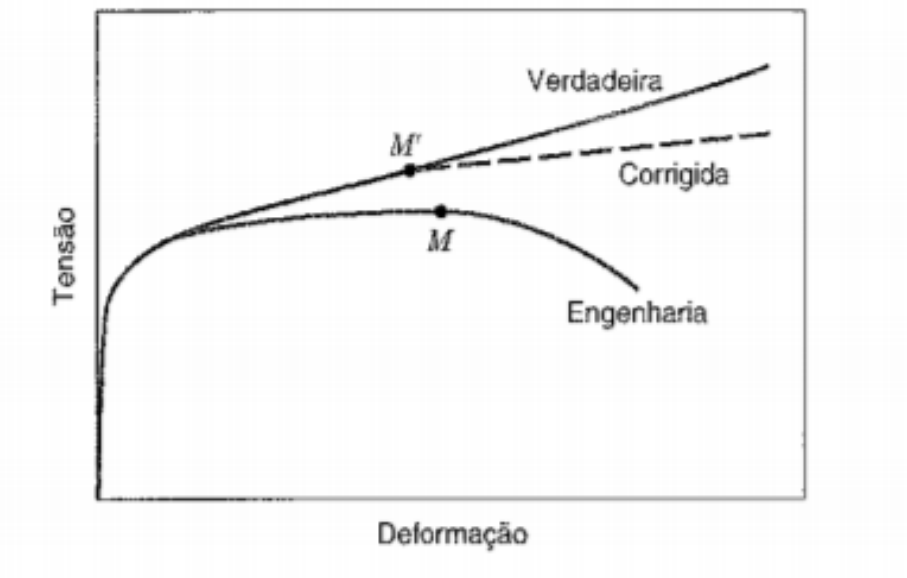
De posse desses conceitos, pode-se classificar os materiais como frágeis ou dúcteis. Materiais são ditos frágeis se a área sob a curva de sua região plástica for pequena. Isso acarreta que a quantidade de energia absorvida pelo objeto durante essa deformação é pequena, o que leva a ruptura catastrófica do mesmo.

Por outro lado, materiais ditos dúcteis possuem uma área grande sob a curva de deformação plástica, absorvendo uma grande quantidade de energia antes de se romper. Em grande parte dos projetos de engenharia mecânica, há preferência pelos materiais dúcteis. Isso é devido à maior segurança em caso de falha do sistema ou quando se deseja maximizar a absorção de energia por um material quando este entre em processo de plastificação.

3.2.3 Tensão e Deformação Real e de Engenharia

Para finalizar a parte de conceitos essenciais da resistência dos materiais, vale ressaltar a diferença entre valores ditos de engenharia e os ditos reais (ou verdadeira). A principal diferença entre os dois conceitos é a área que se utiliza em cada uma das análises. Conforme o objeto é tracionado ou comprimido, a área de sua seção transversal é alterada. Para os cálculos da tensão de engenharia, não se leva em conta a variação instantânea desta área transversal do objeto, utilizando sempre a área inicial. Já para os cálculos da tensão real, leva-se em consideração a área a cada instante. A figura a seguir, retirada de (CALLISTER, 2000) demonstra isso

Figura 10 – Comparação entre tensão de engenharia e a real



Fonte: (CALLISTER, 2000)

Como é possível perceber na imagem, o comportamento real de um material passa a diferir da aproximação de engenharia a partir do regime plástico. Assim, em muitas ocasiões, se faz uma curva corrigida, que representa uma média entre as duas apresentadas na figura 11 acima.

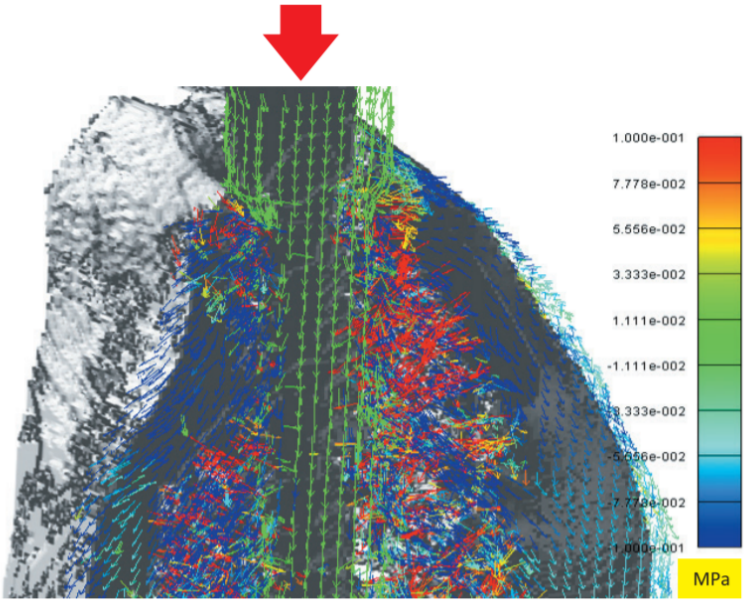
3.2.4 Comportamento mecânico dos ossos Maxilofaciais

Com base na formulação descrita pela mecânica dos sólidos, é possível estruturar modelos que representem os ossos faciais humanos. Isso é possível conhecendo-se as características físico-químicas destes tecidos ósseos. De acordo com a literatura, é plausível se considerar que o osso maxilofacial possui as mesmas características mecânicas dos ossos do tecido trabecular, conforme apresentado por (MATSUNAGA et al., 2011).

Os ossos do maxilar são expostos a diversas pressões funcionais, devido aos dentes e o trabalho destes durante a mastigação. Essas pressões são absorvidas, em sua maioria, pelos músculos maxilofaciais. Assim, existe uma forte relação entre as pressões sofridas pelos ossos do maxilar e os movimentos desta área da face. Ainda resta muito a se pesquisar sobre o comportamento mecânico destes ossos, mas é possível realizar algumas simplificações para que sua análise se torne mais simples.

Conforme feito por (MATSUNAGA et al., 2011), os testes de ossos maxilares foram realizados utilizando-se o método dos elementos finitos e em seguida foram comparados com os resultados dos testes mecânicos do osso trabecular.

Figura 11 – Linhas de tensão no osso da mandíbula quando sujeito a esforços



Retirado de (MATSUNAGA et al., 2011)

Conforme pode ser verificado na figura acima, a carga foi aplicada no sentido vertical, de cima pra baixo. As linhas vermelhas no diagrama representam regiões sob esforços de tração, enquanto as linhas azuis representam compressão. A fixação do implante é diretamente conectada ao osso da mandíbula, causando estresse distribuído ao osso.

A transmissão de carga para o osso da mandíbula foi observados durante o carregamento vertical, indicando o papel biomecânico osso esponjoso impuro. Dessa forma, apesar de não ser um modelo totalmente confiável, aproximar as características biomecânicas dos ossos da mandíbula pelos ossos trabeculares é uma aproximação razoável.

Dito isso, as evidências do estudo, que utilizou um modelo FE incorporando a estrutura trabecular, sugeriu que a estrutura estrutural propriedades do osso esponjoso tiveram um efeito efeito na distribuição de tensões mecânicas de acordo com as condições de carga. Simplificando, foi considerado que um parâmetro de qualidade óssea deve ser introduzido no futuro para melhorar a precisão das análises biomecânicas dos ossos da mandíbula.

3.3 Método dos Elementos Finitos

3.3.1 Conceitos básicos

O Método dos Elementos Finitos (MEF) (*Finite Element Method - FEM)* é um procedimento numérico para determinar soluções aproximadas de problemas de valores sobre o contorno de equações diferenciais. O método consiste na repartição do domínio

estudado em elementos menores (elementos finitos) e na aproximação do comportamento das variáveis em análise por uma função de que depende dos valores nodais das mesmas. "Nó", nesse caso, é o nome dado a junção entre dois elementos. Para determinar os valores das variáveis nodais deve-se resolver um sistema de equações diferenciais que dependem das leis físicas de regem o fenômeno o qual experimenta o objeto em estudo.

A subdivisão de um domínio geral em partes simples tem diversas vantagens, dentre elas:representação precisa de geometrias complexas inclusão de propriedades distintas em materiais dissimilares; identificação de efeitos localizados. Por esses motivos o MEF se tornou um dos métodos mais utilizados para solução de problemas de contorno.

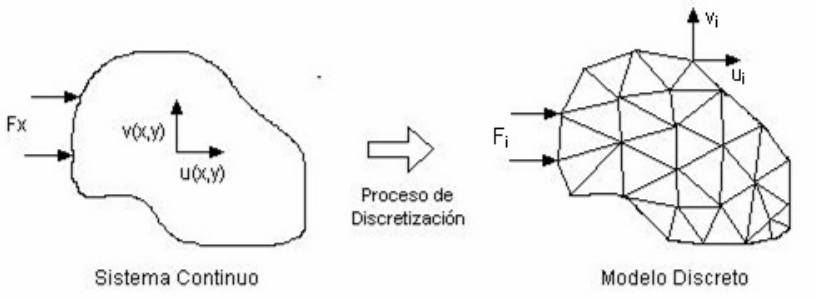
- Dividir o domínio do problema em uma coleção de subdomínios, sendo cada subdomínio representado por um conjunto de equações que são elemento do problema original.

- Recombinar sistematicamente todos os conjuntos de equações do elemento num sistema global de equações para o cálculo final. O sistema global de equações tem técnicas de solução conhecidas, e pode ser calculada a partir dos valores iniciais do problema original, para obter uma resposta numérica.

Portanto, ideia básica é escolher um função aproximadora para a descrição do comportamento da variável de interesse dentro de cada elemento e minimizar o resíduo gerado por essa função quando substituída na equação diferencial que rege o problema. Existem vários métodos para se definir a forma como o erro será minimizado, um deles, bastante usado, é o chamado Método dos Resíduos Ponderados.

A precisão desse método depende de uma maior discretização do problema e aumento do grau da função aproximadora. Portanto, sua viabilidade exige uma interface computacional, já que se trata da solução de sistemas extremamente grandes quando se busca resultados coerentes com a realidade.

Figura 12 – Exemplo de discretização por meio do método dos elementos finitos



Fonte: (UFES,)

3.3.2 MEF aplicado à mecânica

O objetivo do método dos elementos finitos no domínio da mecânica é encontrar uma função aproximada que satisfaça o sistema de equações diferenciais da teoria da elasticidade, obedecendo às condições de contorno do problema específico, com um grau de precisão satisfatório.

Nesse tipo de problema, as variáveis de interesse são os deslocamentos e rotações de cada ponto da estrutura frente a esforços externos. Essas informações podem ser utilizadas para inferir reações, comportamento transiente, análise modal, análise harmônica e etc. Esses dados podem ser extraídos modelando qualquer estrutura mecânica como treliças, vigas, pórticos, sólidos bidimensionais e tridimensionais.

Dessa forma, para este projeto, as variáveis de interesse são as taxas de deformação dos elementos, a energia absorvida pela máscara e a força resultante no rosto do usuário. A partir desses resultados, é possível lançar uma simulação de otimização para que saibamos qual é a espessura ideal de cada camada de material para maximizar a integridade do rosto do usuário.

3.3.3 Métodos Explícito e Implícito

Quando um programa de Elementos Finitos é feito, é preciso escolher um método de resolução das equações diferenciais envolvidas no problema. Os dois métodos mais recorrentes em análises dinâmicas são os Explícitos e os Implícitos.

A abordagem Explícita calcula os deslocamentos e esforços na estrutura no próximo instante de tempo se utilizando do estado da estrutura no instante de tempo atual e anterior. Isso quer dizer que essa abordagem fica condicionada ao passo de tempo, pois um passo de tempo muito grande levaria a grandes erros de integração. Assim, para este método, os passos de tempo utilizados são muito pequenos para se garantir a convergência. São utilizados muitos passos de pequeno custo computacional para se obter a solução. Normalmente, o método explícito é utilizado para problemas de propagação de ondas, impactos, explosões e correlatos.

Por sua vez, a abordagem Implícita calcula os deslocamentos futuros do próximo instante de tempo utilizando os instantes anterior, atual e o próprio estado futuro. Isso significa que a convergência não fica condicionada ao tempo, de modo que a integração se mantém estável com qualquer passo de tempo. Entretanto, o custo computacional de um passo da simulação implícita é muito maior do que de um passo da simulação explícita. Normalmente, análises implícitas são utilizadas em problemas em que a resposta dinâmica se estende por muitos períodos fundamentais.

3.4 Otimização

Os algoritmos de otimização estrutural tem sido intensivamente estudados desde o século XIX (SILVA, 2010) graças às estruturas cada vem mais complexas que vêm surgindo desde então. Essas novas estruturas exigem que modelos cada vez mais complexos sejam criados, para que possamos aplicar o método dos elementos finitos para variar os parâmetros de entrada a fim de se obter os possíveis resultados de saída. Dessa forma, ao se levar em conta o custo computacional e o número de possíveis simulações o tempo necessário para a abordagem "clássica"se torna inviável.

Assim, a otimização se mostrou necessária para que seja possível a análise de grandes sistemas. Nessa abordagem, é realizado uma busca racionalizada da solução através de algoritmos numéricos, o que reduz drasticamente o tempo para encontrar a solução ótima (SILVA, 2010). É preciso portanto partir de pontos iniciais bem escolhidos a fim de se garantir a melhor convergência dos resultados da otimização.

Para uma redução do domínio de análise e melhor escolhas dos pontos iniciais da otimização, é necessário construir um *Design of Experiments* (ou *DOE*). O DOE pode ser visto como um conjunto de pontos iniciais de onde partirá o processo de otimização em si. Assim, o problema só será analisado partindo-se dos pontos do DOE. A grande vantagem da construção de DOEs é a diminuição significativa do número de analises. Se um algoritmo de otimização analisasse todos os pontos do domínio viável como ponto de partida, o custo computacional e de tempo seria impraticável. Além disso, pontos de partida muito próximo uns dos outros tendem a convergir para as mesmas soluções, de modo que não há vantagem prática de se analisar diversos pontos de partidas vizinhos. Com isso, a distribuição de pontos iniciais do DOE visa preencher o mais homogeneamente possível os pontos iniciais de otimização para se diminuir o número de pontos a se analisar e evitar cair nos mesmos pontos ótimos do problema. Deve-se selecionar uma técnica de preenchimento para estabelecer nosso DOE, como as técnicas de hipercubo latino ou Sobol, detalhados em (COSTA, 2019).

Em seguida, é necessário definir as variáveis de entrada e de saída do nosso sistema, ou seja, os parâmetros interessantes para a análise. Essas variáveis são chamadas de variáveis de projeto, que representam os parâmetros que serão estudados durante o processo de otimização. Juntamente com as variáveis de objetivo, temos também num problema de otimização a entidade função objetivo, que representa a função sobre a qual deseja se encontrar os pontos ótimos. A função objetivo deve conter uma relação matemática entre as variáveis de projeto, de modo a poder mensurar um certo valor que deverá ser maximizado ou minimizado, dependendo do objetivo do problema

Por fim, é possível que um problema possua restrições impostas no domínio. Essas restrições representam limitações que nosso modelo deve ter para respeitar alguns critérios.

Isso significa é preciso encontrar o ponto ótimo da função objetivo mas que ao mesmo tempo respeite os limites impostos. Caso o ponto ótimo global se encontre em uma região fora do domínio viável, o ponto ótimo local (dentro do domínio viável) será o objetivo a ser alcançado.

Por exemplo, a força máxima aplicada a uma estrutura deve ser inferior ao seu limite elástico para se evitar deformações plásticas. Assim, nossas variáveis de projetos podem ser podem exemplo as áreas dos elementos que compõem a estrutura e suas dimensões. A função objetivo pode ser minimizar o peso total da estrutura com a restrição de que a força máxima aplicada deve ser menor ou igual ao limite elástico dela.

Para ilustrar as explicações acima, a imagem a seguir representa um fluxograma de um processo de otimização qualquer:

3.4.1 Algoritmos de Otimização

Podemos dividir os algoritmos numéricos de otimização em três grupos:

- Algoritmos de Ordem Zero: Métodos que utilizam apenas os valores da função objetivo em diversos pontos. Esses métodos são aconselháveis quando a obtenção das derivadas da função não é simples. Assim, a convergência pode ser um pouco mais demorada
- Algoritmos de Primeira Ordem: Representam estratégias onde tanto o valor da função quanto seu gradiente são utilizados. Esses algoritmos apresentam uma taxa de convergência linear, sendo ferramentas muito robustas na busca por pontos ótimos
- Algoritmos de Segunda ordem: Utilizam os valores da função, seu gradiente e sua matriz Hessiana. Possuem uma ótima taxa de convergência mas tem a desvantagem de precisar da matriz Hessiana, que por diversas vezes não é trivial de se obter

Dentre os métodos de ordem zero, o algoritmo de Powell se destaca por sua taxa de conversão e alta confiabilidade. Nesse método, a função a ser minimizada é aproximada localmente por uma função quadrática da forma:

$$f(x) = \frac{1}{2}x^T Q x + b^T x + c \tag{4}$$

Assim, é mostrado em (SILVA, 2010) que se existem direções s_i, s_j Q-conjugadas tais que:

$$s_i^T Q s_j = 0, \quad i \neq j \tag{5}$$

Então se f(x) for otimizada nas direções s_i, s_j haverá convergência para o ponto ótimo em até n passos, onde n representa o número de variáveis do prolema, uma vez que os erros de arredondamento não sejam acumulados. Entretanto, obter as direções Q-conjugadas não é algo trivial de se fazer. Assim, o algoritmo de Powell tem por objetivo encontrar essas direções e realizar um processo iterativo de otimização para se localizar o ponto ótimo. Uma grande vantagem do algoritmo é que ele converge independentemente do ponto de partida, o que é muito valioso quando estudamos otimizações baseadas em DOEs, pois não estamos condicionados a cair em pontos de DOE que não convergem para uma solução.

3.4.2 Diferenciação Numérica

Diversas vezes, não é fácil ou trivial se obter as derivadas de um função objetivo ou de uma série de dados qualquer. É possível até que não se tenha uma função sobre a qual trabalhar, mas se tenha apenas uma série de pontos que se deseja calcular as derivadas. Por exemplo, se um experimento de queda livre for realizado, é relativamente simples se mapear a posição do corpo em queda ao longo do tempo, porém se mostra uma tarefa bem mais difícil mapear sua velocidade ao longo da trajetória.Dessa forma, a melhor maneira de se obter esses valores de velocidade é através de uma técnica de diferenciação numérica utilizando as informações da posição ao longo do tempo.

Existem diversas técnicas para se obter derivadas numericamente, dentre as quais uma das mais utilizadas é o método das diferenças finitas. Este método consiste em realizar uma estimativa da derivada dos pontos utilizando os próprios pontos da base de dados. Para isso, podem ser utilizados 3, 5, 7 ou até mesmo 9 pontos para se realizar esta estimativa. Em teoria, quanto maior o número de pontos, mais acurado será o resultado da diferenciação. Entretanto, a desvantagem de se utilizar métodos com grande número de pontos é a diminuição da base de dados, uma vez que o gráfico da derivada não terá o mesmo número de pontos do gráfico original. Dessa forma, para três pontos, o método das diferenças finitas pode ser escrito como:

$$\frac{df(x_n)}{dx} \approx \frac{f(x_{n+1}) - f(x_{n-1})}{2h} \tag{6}$$

Onde h é o passo entre os pontos (x+1) e (x-1). Analogamente, para cinco pontos, temos:

$$\frac{df(x_n)}{dx} \approx \frac{f(x_{n-2}) - 8f(x_{n-1}) + 8f(x_{n+1}) - f(x_{n+2})}{12h} \tag{7}$$

E assim sucessivamente, o processo pode ser extrapolado para diversos outros número de pontos e as deduções das formas são análogas às mostradas acima.

Um segundo método alternativo para o cálculo das derivadas é o método de Lanczos, que consiste em encontrar as derivadas de uma curva minimizando uma função de custo. Novamente, é possível utilizar diversos números de pontos para refinar ainda mais a solução da diferenciação, sendo mostrado abaixo as formas com 5 e 7 pontos, respectivamente:

$$\frac{df(x_n)}{dx} \approx \frac{f(x_{n+1}) - f(x_{n-1}) + 2(f(x_{n+2}) - f(x_{n-2})}{10h} \tag{8}$$

$$\frac{df(x_n)}{dx} \approx \frac{f(x_{n+1}) - f(x_{n-1}) + 2(f(x_{n+2}) - f(x_{n-2}) + 3(f(x_{n+3}) - f(x_{n-3})}{28h} \tag{9}$$

Por fim, um método muito robusto de se encontrar as derivadas de um gráfico numérico é o chamado *Smooth noise-robust differentiation* (SNRD). Esté método é interessante pois ele reduz significativamente a interferência de ruídos da base de dados no resultado da diferenciação. A forma de diferenciação dada por esté método, para 5 e 7 pontos é mostrada abaixo:

$$\frac{df(x_n)}{dx} \approx \frac{2(f(x_{n+1}) - f(x_{n-1})) + f(x_{n+2}) - f(x_{n-2})}{8h} \tag{10}$$

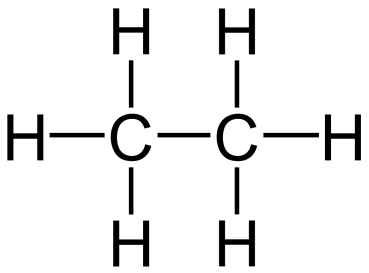
$$\frac{df(x_n)}{dx} \approx \frac{5(f(x_{n+1}) - f(x_{n-1})) + 4(f(x_{n+2}) - f(x_{n-2}) + f(x_{n+3}) - f(x_{n-3})}{32h} \tag{11}$$

Vale ressaltar também que os três métodos apresentados acima possuem também suas formas para derivadas de ordem superior caso se deseje estimá-las. A ideia consiste na mesma, estimar a derivada segunda, por exemplo, a partir dos pontos originais obtidos experimentalmente.

3.5 Polímeros

A palavra polímero remete à toda matéria composta de longas cadeias ligadas formadas pela ligação de monômeros devido ao fornecimento de calor e/ou catalisadores (MOURA, 2013). Estes materiais vem sendo estudados devido às suas propriedades diferentes dos demais materiais utilizados em engenharia, como o aço e o alumínio, por exemplo.

Figura 13 – Exemplo do monômero Etano, componente do Polietano



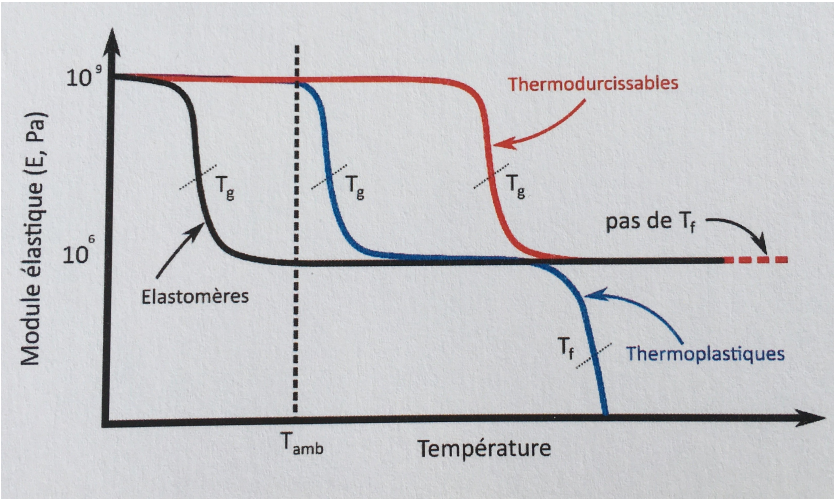
Fonte: https://maestrovirtuale.com/etano-estrutura-propriedades-usos-e-riscos/

Os polímeros vem sendo cada vez mais utilizados em aplicações mecânicas, como a indústria automobilística e aeroespacial. As propriedades físico-químicas deste material variam de acordo com o tamanho da cadeia polimérica e efeitos externos, como radiosidade e calor. Assim, sua aplicação em projetos deve ser feita mediante um estudo prévio detalhado a respeito das condições de utilização. Além disso, é possível associar os polímeros com outros materiais, para maximizar as melhores características de cada material, formando assim uma nova mistura chamada de compósito (COLEMAN et al., 2006).

3.5.1 Classificações dos polímeros

Os polímeros podem ser classificados quanto à diversos fatores, tanto físicos quanto químicos. As duas principais características para a classificação de um polímero são com respeito à presença ou não de uma temperatura de fusão e do grau de organização da molécula. Para a primeira característica, ao analisar o comportamento de um polímero sob diferentes temperaturas, podemos dividi-los em três grandes grupos: Termoplásticos, Termofixos e Elastômeros. A imagem a seguir mostra as características desses três grupos:

Figura 14 – Gráfico do Módulo de Young pela Temperatura de cada um dos grupos de polímeros: Termoplásticos (em azul), termofixos (em vermelho) e Elastômeros (em preto)

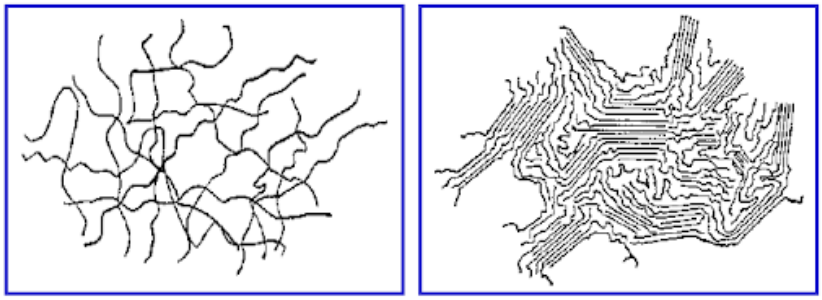


Fonte: retirado de (ROLAND, 2018-2019)

Na imagem acima, podemos ver que todos os grupos apresentam um comportamento em comum, que é uma temperatura na qual o módulo de elasticidade cai notavelmente. Essa temperatura é chamada de temperatura de transição vítrea (chamada comumente de T_g) e é caracterizada como uma transição entre estados de equilíbrio termodinâmico. Assim, T_g caracteriza o que são polímeros.

A segunda classificação geral dos polímeros é quanto sua organização molecular. Polímeros ditos Amorfos são aqueles que não possuem uma organização aparente de suas lamelas moleculares enquanto aqueles que possuem uma organização local dessas estruturas são ditos semi-cristalinos. A imagem a seguir mostra a diferença visual entre esses dois grupos

Figura 15 – Diferenças entre cadeias poliméricas amorfa e semi-cristalina



Fonte: retirado de (ROLAND, 2018-2019)

Com essas duas classificações, podemos nos aprofundar em cada uma das classificações expressas acima

3.5.1.1 Termoplásticos

Para os materiais termoplásticos, temos dois pontos onde há uma grande variação de módulo elástico. O primeiro deles é a característica do polímero em si, a temperatura de transição vítrea T_g. Já a segunda é representada por T_f na figura 14 e representa a temperatura de fusão do polímero. Isso quer dizer que materiais ditos termoplásticos passam por um processo de fusão quando a temperatura atinge T_f. Além disso, termoplásticos podem ser tanto amorfos de cadeia linear (sem reticulações) ou semi-cristalinos lineares.

Para termoplásticos amorfos, a estrutura molecular passa para a forma fluida sem fusão, mas sim graças a um "desembaraçamento"da cadeia. Já a fase cristalina dos polímeros semi-cristalinos realmente passa por fusão para passar à fase líquida, justamente devido à organização das lamelas moleculares.

Essa família de polímero é altamente utilizada graças à facilidade de confecção, por serem altamente recicláveis (devido à presença da T_f) e pela sua alta capacidade de

deformação plástica (ROLAND, 2018-2019). Geralmente esse tipo de polímero é utilizado em temperaturas abaixo de T_g, ou seja, antes do estado de "borracha".

3.5.1.2 Termofixos

Os materiais termofixos por sua vez não possuem temperatura de fusão. Desse modo, quando expostos a altas temperaturas esses materiais simplesmente se degradam e portanto suas cadeias poliméricas são decompostas e o material é inutilizado. Dessa forma, materiais termofixos não são recicláveis, devido à esta característica.

Esses materiais se apresentam sempre em fase amorfa uma vez que se organizam em forma de rede altamente reticulada. Dessa forma se caracterizam como materiais mais rígidos e com pouca capacidade de alongamento. São utilizados geralmente em aplicações que demandem altas capacidades mecânicas, como por exemplo solados de calçados, amortecedores e equipamentos industriais.

3.5.1.3 Elastômeros

Elastômeros são materiais poliméricos que à temperatura ambiente já se encontram em um estado de "borracha". Isso quer dizer que eles possuem uma temperatura T_g muito baixa. Além disso, elastômeros não possuem temperatura de fusão e assim como os termofixos eles simplesmente se degradam em altas temperaturas.

A estrutura molecular dos elastômeros é do tipo amorfa com poucas reticulações. Isso garante ao material uma grande elasticidade reversível além de grandes extensões (podendo chegar até a 1000%). Devido à essas características, os materiais elastômeros são amplamente utilizados em aplicações como borrachas, pneus e amortecedores.

3.5.2 Modelos Reológico dos Polímeros

Um modelo reológico pode ser entendido como uma analogia mecânica para melhor visualizar o comportamento dos materiais. Assim, fenômenos complexos podem ser traduzidos por associações em série e paralelo de elementos simples de mecânica, como molas e amortecedores.

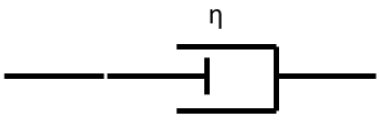
O modelo reológico mais simples é o que obedece os fenômenos de Hook, que representa a elasticidade dos materiais. Nesse tipo de abordagem, o material possui apenas elasticidade, de modo que a deformação sofrida é diretamente proporcional à tensão aplicada (conforme equação (3) acima. Além disso, as deformações não são permanentes, de modo que ao se retirar a tensão imposta, a deformação volta a ser nula. Esse tipo de modelo pode ser representado por uma mola de constante elástica E (Módulo Elástico) como na figura abaixo

Figura 16 – Modelo Reológico para a representação da Elasticidade dos materiais

Fonte: autoria própria

Entretanto, diversos materiais não possuem apenas comportamento elástico, apresentando também comportamentos plásticos. Nesse tipo de fenômeno, para uma imposição de tensão, ocorrem deformações permanentes e irreversíveis no material. Assim, podemos representar um material com comportamento unicamente plástico linear como um amortecedor de constante de amortecimento η, conforme mostrado na figura abaixo

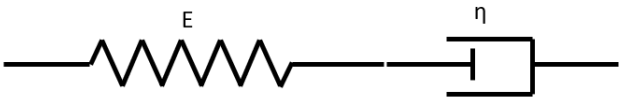
Figura 17 – Modelo Reológico para a representação da Plasticidade dos materiais



Fonte: autoria própria

Com base nos dois elementos acima, é possível se pensar em modelos mais complexos que representem melhor os materiais reais e os fenômenos que os acompanham. Um desses fenômenos é a viscoelasticidade, que se trata do fenômeno de se possuir tanto deformações elásticas quanto viscosas quando submetido à tensões. Além disso, o material viscoelástico dissipará melhor a energia do que um material puramente elástico, devido à presença do amortecedor viscoso. Um modelo clássico para a representação de materiais viscoelásticos é o modelo de Maxwell, que pode ser visto abaixo

Figura 18 – Modelo Reológico de Maxwell

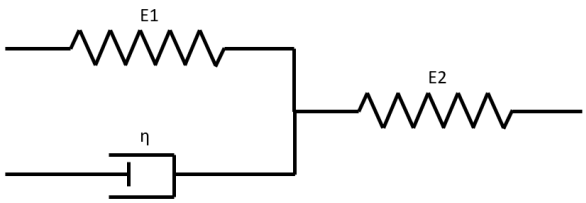


Fonte: autoria própria

Entretanto, o modelo de Maxwell representa bem apenas viscoelasticidade linear, pois há presença de deformação elástica instantânea e deformação plástica dependente do tempo, de acordo com a associação em série acima. Para polímeros, este tipo de representação não é muito eficaz, uma vez que as deformações elásticas, assim como a recuperação de deformação (quando retiramos a tensão) não são lineares (BARRA,). Com

isso, modelos mais robustos são utilizados para representar materiais como os polímeros. Um tipo de modelo muito utilizado é o de Arruda-Boyce, que em conjunto com o modelo de viscoelasticidade de Maxwell, consegue descrever o comportamento visco-hiperelástico típico dos polímeros emborrachados, como o EVA flexível deste relatório. O modelo de Arruda-Boyce é o mostrado na figura a seguir

Figura 19 – Modelo Reológico de Arruda-Boyce



Fonte: autoria própria

Com o modelo de Arruda-Boyce acima e adicionando-se as contribuições viscoelásticas do modelo de Maxwell pode-se representar um material cuja deformação e a recuperação elástica são retardadas pelos elementos viscosos do sistema, representando mais fielmente assim o comportamento desse grupo de polímero.

Uma possível modelização do comportamento viscoelástico de um material é obtida através da série de Prony. Este tipo de modelização leva em conta o tempo de relaxação do material (GHOREISHY, 2012), e pode ser escrito da seguinte forma:

$$g_R = 1 - \sum_{i=1}^{N} g_i(1 - e^{-t/\tau_i}) \tag{12}$$

Onde g_i é a constante de viscoelasticidade do material e τ_i o tempo de relaxamento respectivo. Além disso, g_R se trata do módulo de cisalhamento adimensional, que pode ser escrito como:

$$g_R = \frac{G(t)}{G_0} \tag{13}$$

Onde $G(t)$ e G_0 representam respectivamente os módulos de cisalhamento instantâneo e inicial. Em geral, para a série de Prony consiga representar bem o comportamento viscoelástico de um material, é aconselhável que N esteja entre 3 e 8. Dessa forma, teremos mais informações sobre o relaxamento do material em diferentes situações, melhorando a definição dos resultados do modelo.

3.5.3 EVA e Silicone

Como dito anteriormente, o EVA e o silicone são dois dos possíveis materiais que podem ser utilizados em máscaras esportivas devido às suas características mecânicas e térmicas. Dessa forma, é preciso conhecer essas características para que seja possível modelizar os materiais no *softwware* dedicado à isso. Para a obtenção das características do material, é necessário a realização de diversos ensaios experimentais.

Esses ensaios já foram realizados por (SANTOIEMMA, 2018) em anos anteriores e portanto os resultados serão utilizados aqui para os processos de modelização e otimização. O silicone é classificado como um termofixo e por ser biocompatível e não possuir uma temperatura de fusão, assim como por possuir boas características mecânicas, esse material é amplamente utilizado em próteses mamárias. Entretanto, suas aplicações são amplas de modo que podem ser aplicados desde colas até em filtros solares.

Já o EVA (tanto rígido como o flexível) foi utilizado nos ensaios em questão para simular a máscara de proteção. O EVA, material de estudo deste trabalho, é caracterizado como um termoplástico, podendo ser semi-cristalino ou amorfo, dependendo de seu processo de resfriamento. Se o EVA durante seu preparo é resfriado muito rapidamente (como por exemplo um molde muito mais frio do que o polímero em estado líquido) o material não tem tempo de formar cadeias cristalinas e portanto ele passa rapidamente ao estado sólido com estrutura molecular amorfa. O inverso também é verdadeiro, ou seja, se resfriamento é relativamente lento, o resultado final é semi-cristalino. Essas características podem ser descobertas através de um ensaio DSC (Differential Scanning Calorimetry), por exemplo, onde é possível verificar se existe um pico de calor à frio, característica de materiais amorfos.

4 Metodologia

4.1 Fundamentos

O principal objetivo deste projeto é se obter um modelo de material coerente com os ensaios experimentais realizados por (SANTOIEMMA, 2018) e explicar os resultados obtidos de uma maneira mais sofisticada. Assim, a grande vantagem de se obter um modelo numérico que explica bem a realidade é a maior capacidade exploratória de se realizar mais ensaios, pois uma vez validado a simulação com a realidade pode-se estimar o comportamento do sistema sob diferentes condições que não necessariamente foram realizadas experimentalmente.

Dadas as explicações acima, foi estabelecida uma metodologia para este projeto, partindo-se de um modelo em elementos finitos para a calibração dos parâmetros dos materiais envolvidos e chegando-se nos ensaios de impacto dos materiais. Escolheu-se

portanto realizar primeiro um modelo simplificado com base nos experimentos relatados em (SANTOIEMMA, 2018). O objetivo dessa primeira modelização é a familiarização com os softwares assim como a obtenção de resultados via otimização do material. Serão utilizados dois softwares para isso: ABAQUS e ModeFrontier.

4.1.1 ABAQUS

O software ABAQUS é um software comercial de CAE (*Computer Aided Engineering* oferecido pela empresa Dassault Systèmes. Uma ferramenta CAE permite ao usuário a realização de simulações de Elementos Finitos, CFD, simulações de impacto entre outras funcionalidades. Dessa forma, o ABAQUS foi escolhido devido à sua facilidade de manipulação e robustez de tipos de simulações que podem ser feitas.

Além disso, para este projeto, escolheu-se realizar a modelização dos sistemas a serem estudados via script escrito em linguagem de programação Python. A grande vantagem deste tipo de abordagem é a facilidade de mudança de parâmetros importantes na simulação (como características dos materiais, coeficientes de atrito, dimensões dos corpos envolvidos, etc), além de garantir maior segurança sobre a modelização, uma vez que todos os comandos de execução estão contidos em um mesmo arquivo. Por fim, a vantagem de se realizar a modelização com um script é a facilidade de se fazer a otimização através do software modeFrontier. Esta conexão será explcada abaixo.

4.1.2 modeFrontier

O software modeFrontier é uma ferramenta de otimização Multi-Objetivo e Multi-disciplinar com integração com diversos softwares de CAE e CAD, notoriamente com o ABAQUS, oferecido pela empresa ESTECO. O software de otimização se utiliza das técnicas mostradas nas seções acima para encontrar pontos ótimos de uma ou mais funções objetivo, dependendo do usuário construir um *workflow* que atenda as necessidades do projeto. É possível ainda utilizar restrições, diferentes tipos de DOEs e de algoritmos de otimização e diferentes objetivos a ser alcançados (onde se destacam minimizar ou maximizar as funções objetivo).

A integração deste software com o ABAQUS, por exemplo, se faz através de scripts de comando. O modeFrontier recebe o arquivo de modelização CAE e o executa (através do ABAQUS, no caso) a fim de se obter os resultados. Uma vez feito isso, o valor da função objetivo será calculado e um processo de otimização será aplicado para se descobrir o próximo ponto de partida da análise. Uma vez encontrado esse ponto, o modeFrontier cria uma cópia do script de modelização, altera os parâmetros necessários para se adequar ao novo ponto inicial e roda a simulação novamente.

O processo descrito acima continua até que o próprio software julgue que um ponto ótimo foi encontrado. Se foi definido que o objetivo de otimização era a minimização da

função objetivo, então o ponto ótimo será um mínimo local. Caso contrário, o ponto será um máximo local.

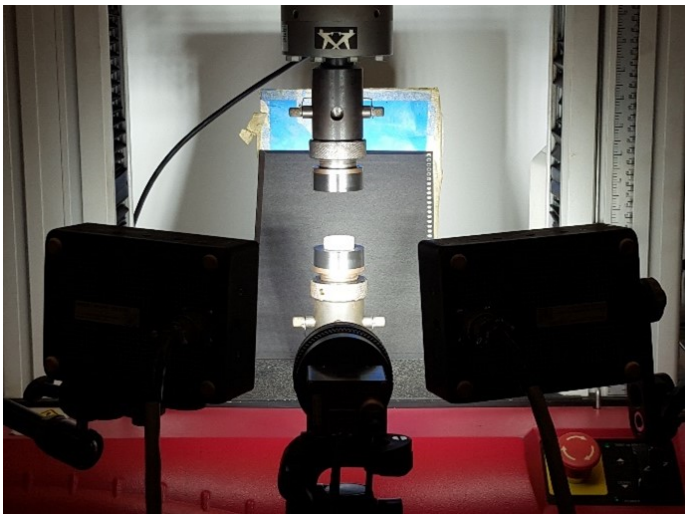
4.2 Calibração do material

4.2.1 Modelização MEF

O primeiro passo da análise deste projeto é calibrar o modelo de material para se aproximar o máximo possível da realidade dos testes experimentais de compressão dos EVAs flexível e rígido realizados por (SANTOIEMMA, 2018). Dentre os resultados obtidos experimentalmente, o mais interessante para este trabalho é o gráfico Força x Tempo obtido, pois ele que mostrará quanto de força será dissipada no protetor facial e quanto chegará aos ossos da face. Desta forma, calibrar o material significa deixar as curvas de Força x Tempo das simulações de elementos finitos as mais próximas possíveis das obtidas experimentalmente, respeitando também as dimensões e as condições dos testes.

Dessa forma, primeiramente foi analisado os parâmetros geométricos dos materiais de EVA utilizados. Nos ensaios de compressão, discos de EVA flexível e rígido de diâmetro 30 milímetros e de espessura 5.5 milímetros foram fabricados e colocados em suportes de compressão de aço-carbono, com 50 milímetros de diâmetro da máquina de ensaios INSTRON 3369. Desta forma, o corpo de prova de EVA é colocado entre esses dois suportes para que o ensaio seja realizado.

Figura 20 – Equipamento montado para a realização dos ensaios experimentais



Fonte: retirado de (SANTOIEMMA, 2018).

Os ensaios foram realizados com quatro taxas de deformação diferentes: 0.1, 0.01, 0.001, 0.0001 s^{-1}. Entretanto, para as simulações numéricas, apenas as três primeiras taxas de deformação foram consideradas, uma vez que a taxa de deformação mais baixa levaria à um tempo computacional muito grande. Além disso, as três primeiras taxas de

deformação são suficientes para a calibração das propriedades do material com uma boa confiabilidade.

Dessa forma, o script de modelização no ABAQUS começou a ser escrito. O primeiro passo foi a definição da geometria dos suportes e dos discos de EVA, idênticas às do procedimento experimental. Em seguida, foram abordados os modelos de material das partes. Os suportes foram admitidos como corpos rígidos (e portanto indeformáveis), o que é uma hipótese razoável para o caso. Para o EVA rígido, um modelo Elástico simples foi adotado, fornecendo-se assim ao software densidade, módulo de Young e coeficiente de Poisson.

Por fim, para o material flexível, um modelo Visco-Hiperelástico foi implementado. Para a parte viscosa, um modelo em série de Prony foi criado com N = 4, enquanto que para a parte Hiperelástica uma modelização do tipo Arruda-Boyce foi proposta. Para essa modelização, o software ABAQUS trabalha com três parâmetros: μ_0 módulo de cisalhamento inicial; λ_m alongamento máximo com o qual as cadeias do polímero travam; D parâmetro do material relacionado com o módulo de compressibilidade. Os valores iniciais foram valores padrões de um polímero genérico encontrado na literatura e podem ser conferidos nas tabelas abaixo

Tabela 1 – Valores iniciais da modelização de Arruda-Boyce

Valor	μ_0	λ_m	**D**
Valores Iniciais	1.0E6	10.0	0.0

Fonte: autoria própria

Tabela 2 – Valores iniciais da modelização da Série de Prony

Termo	**1**	**2**	**3**	**4**
g_i	0.1	0.1	0.1	0.1
k_i	0.1	0.1	0.1	0.1
τ_i	1.0	1.0	1.0	1.0

Fonte: autoria própria

Em seguida, todas as partes foram colocadas nas posições corretas no mesmo *assembly* e as interações de contato foram criadas. Para este projeto, foi admitido um contato generalizado entre discos de EVA e suportes metálicos com coeficiente de atrito tangencial de 0.25 e contato normal do tipo *"Hard"*. Um *step* dinâmico explícito foi criado para a análise e o próximo passo foi adicionar as condições de contorno do sistema, como por exemplo o engastamento do suporte inferior e a limitação de movimentação do suporte superior para apenas a direção z. Uma velocidade de descida do prato superior foi associada a este corpo para representar a compressão do ensaio. O valor da velocidade varia de acordo com a taxa de deformação que se deseja obter. Para as três taxas analisadas no problema, os valores das velocidades em milímetros por segundo do prato superior foram:

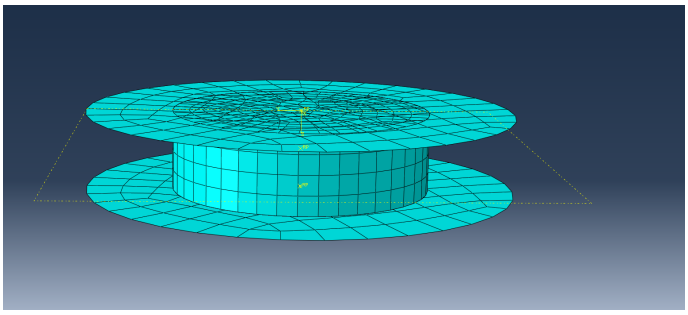
Tabela 3 – Velocidades de compressão dos ensaios realizados

Modelo	$\dot{\epsilon} = 10^{-1}s^{-1}$	$\dot{\epsilon} = 10^{-2}s^{-1}$	$\dot{\epsilon} = 10^{-3}s^{-1}$
Vel. Flexível	0.89 mm/s	0.09 mm/s	0.012 mm/s
Vel. Rígido	0.55 mm/s	0.06 mm/s	0.006 mm/s

Fonte: autoria própria

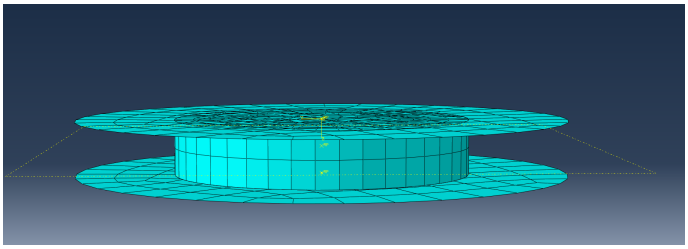
Por fim, uma malha de elementos lineares foi criada utilizando elementos tridimensionais de largura média 2 milímetros. Cada elemento possui 8 nós onde serão calculados os deslocamentos do corpo de prova. O resultado da modelização é mostrado abaixo:

Figura 21 – Modelização do ensaio de compressão do material Flexível



Fonte: autoria própria

Figura 22 – Modelização do ensaio de compressão do material Rígido



Fonte: autoria própria

4.2.2 Modelização para Otimização

Uma vez pronta a modelização em MEF, o ambiente de otimização foi preparado. Para isso, o software modeFrontier foi utilizado. É preciso definir quais são as variáveis de entrada e de saída no modelo, para que a otimização possa ser feita. As variáveis de entrada são aquelas que a cada iteração possuirão seus valores alterados pelo algoritmo de otimização. Dessa forma, como o objetivo neste caso é a calibração de material, os parâmetros escolhidos para serem otimizados (parâmetros de entrada do modeFrontier) foram:

- Material Flexível: Parâmetros da Série de Prony (g_i, τ_i); parâmetros da modelização Arruda-Boyce (μ_0 e λ_m)

- Material Rígido: Módulo de Elasticidade; coeficiente de Poisson

Além disso, algumas restrições quanto aos valores máximo e mínimo que cada variável poderia ter foram impostas. Por exemplo, o ABAQUS impõe que a soma de todos os valores g_i seja menor do que 1. Além disso, alguns limites foram definidos como os máximos e mínimos que cada variável pode assumir. Dessa forma, os seguintes limites foram impostos no software de otimização:

Figura 23 – Restrições aplicadas a cada variável de otimização

	Name	Type	Default Value	Expression	Lower Bound	Upper Bound	Central Value	Delta Value
1	Mu0	Variable	0.0		800000.0	2000000.0	1400000.0	600000.0
2	Lambdam	Variable	0.0		5.0	50.0	27.5	22.5
3	Gi	Variable	0.0		0.0500000000...	0.99	0.52	0.47
4	Taui	Variable	0.0		0.1500000000...	100.0	50.075	49.925

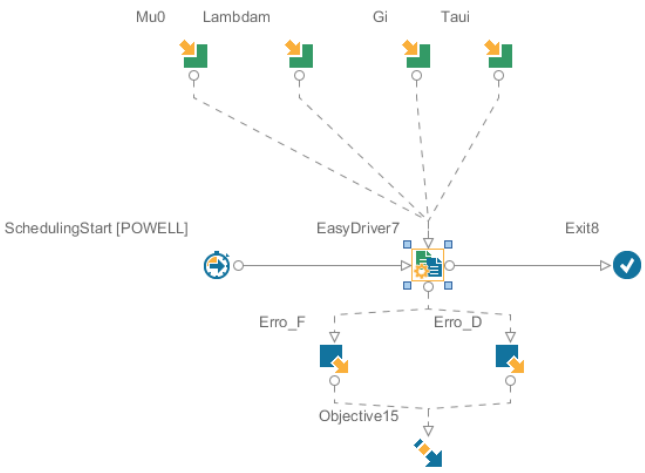
Fonte: autoria própria

Em seguida, o fluxo de trabalho foi criado na interface do software modeFrontier. Para isso, o script de modelização foi adaptado para retornar um gráfico de força por tempo e um segundo gráfico de variação do diâmetro no meio do corpo de prova por tempo. O objetivo desta ação foi de comparar, a cada iteração, o gráfico obtido experimentalmente (e portanto imutável) com o gráfico da iteração (que muda dependendo dos parâmetros de entrada da otimização) e foi definido portanto que o objetivo da otimização seria minimizar o erro ponto a ponto de ambos os gráficos. Neste projeto, "Erro_F"representa a soma dos erros ponto a ponto do diagrama força x tempo experimental com o obtido numericamente. Analogamente, "Erro_D"representa o erro de variação de diâmetro do corpo de prova.

Com essas informações, foi montado um DOE para o problema a fim de se tomar pontos iniciais mais homogeneamente espalhados. Para a criação do DOE, a técnica de Hipercubo Latino (ULH), que pode ser conferida em detalhes em (STEIN, 1987). A técnica de otimização escolhida foi a de direções conjugadas de Powell, uma vez que a obtenção das derivadas da função objetivo não eram de fácil acesso.

O último ponto restante é fazer a ligação entre modelo de otimização. Isso foi feito através de uma interface do modeFrontier chamada *EasyDriver*, responsável por receber o script em Python e executá-lo no software ABAQUS. O comando de execução do software CAE foi feito através do comando do prompt "abaqus cae". Com isso, o resultado do programa de otimização pode ser conferido na imagem abaixo:

Figura 24 – Programa de otimização feito no software modeFrontier



Fonte: autoria própria

4.3 Simulações de Impacto

4.3.1 Calibração Osso e Pele

Como não foram realizados testes de caracterização de material nem para o poliuretano nem para a borracha de silicone, o mesmo procedimento abordado anteriormente foi refeito aqui. Entretanto, como não foram realizados ensaios de compressão para estes materiais, para a calibração numérica foram utilizados os dados do ensaio de impacto. Mais uma vez, o interessante dentro do escopo deste trabalho é comparar os gráficos de força x tempo obtidos nos ensaios experimentais com os obtidos numericamente, para se avaliar a força máxima transmitida para o tecido ósseo e também a absorção de energia pelo protetor facial. Assim, foram feitas simulações de impacto somente com o osso (disco de poliuretano) e com o osso mais a pele (borracha de silicone).

Uma vez calibrados os materiais de EVA, as simulações de impacto foram realizadas. Esta etapa consiste em verificar a eficiência dos materiais de proteção facial quando submetidos à uma situação de impacto. Nos experimentos realizados por (SANTOIEMMA, 2018), uma bola de golfe de massa 45.93 gramas e de diâmetro 42.67 milímetros foi largada do repouso de uma altura de 7.45 metros para se atingir os corpos de prova, simulando-se uma situação de impacto.

Para isso, corpos de prova de 85 milímetros de espessura foram fabricados para se simular o tecido ósseo de um rosto craniomaxilofacial e a pele que o recobre. O corpo de prova representando o osso humano possuía 10,3 milímetros de espessura e foi fabricado de Poliuretano 20 PFC. Já o material escolhido para representar a pele foi a borracha de silicone SQ 8335, com dureza Shore A 18-21, com 12,3 milímetros de espessura.

Para a modelização do disco de poliuretano, foi utilizado um modelo elasto-plástico perfeito, com valores iniciais de σ_y, módulo de elasticidade e coeficiente de Poisson respectivamente iguais à 100 MPa, 30 MPa e 0.48. Além disso, uma parcela de comportamento viscoelástico foi adicionada ao osso para representar a pequena parcela viscosa que existe neste componente. Mais uma vez, a série de Prony foi utilizada para esta modelização, mas com apenas um componente para representar uma viscoelasticidade fraca.

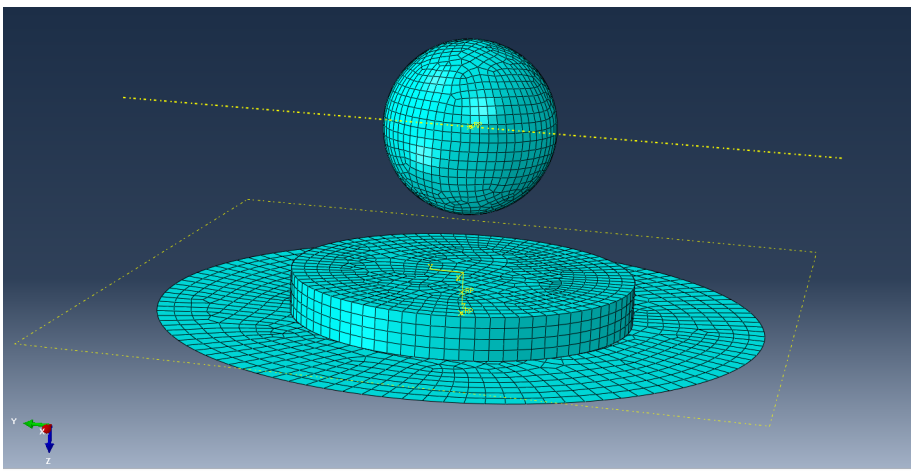
Quanto à modelização do disco de silicone, um modelo viscoelástico mais robusto foi adotado. Os valores iniciais do módulo de elasticidade e do coeficiente de Poisson do material são dados por 1 MPa e 0.49, respectivamente. Para a parte viscosa, uma outra série de Prony foi utilizada mas dessa vez com quatro termos, a fim de representar um comportamento mais viscoso típico de elastômeros.

4.3.2 Simulações Numéricas

Uma vez caracterizados os materiais, o mesmo princípio de otimização aplicado aos materiais Flexível e Rígido foi aplicado aqui, para se otimizar os parâmetros de material para mantê-los o mais próximo possível da realidade. Para isso, a modelização do ensaio de impacto foi feita. Um corpo esférico para representar a massa de impacto foi criada com as mesmas propriedades da bola de golfe utilizada experimentalmente. Para representar a célula de carga, um suporte rígido foi criado e engastado e um ponto de referência onde a força será medida foi criado, se aproximando assim da célula de carga fixa no suporte inferior da análise experimental.

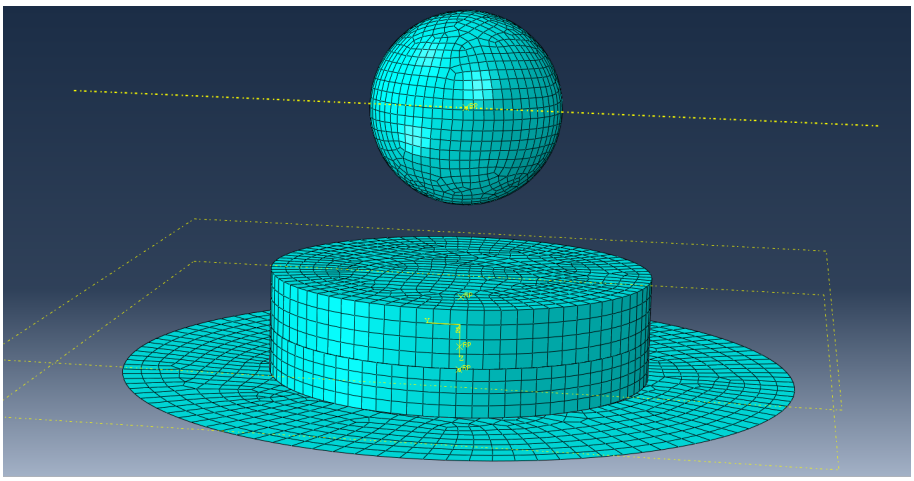
Em seguida, as peças foram colocadas juntas em um mesmo *assembly* e os contatos envolvidos foram criados para representar o atrito de cada uma das partes. Como condição de contorno, a velocidade inicial da esfera de impacto foi definida com base em cada experimento realizado. Por fim, a malha foi criada e o resultado da modelização se encontra abaixo

Figura 25 – Modelização do experimento de impacto para o Osso



Fonte: autoria própria

Figura 26 – Modelização do experimento de impacto para o Osso + Pele (silicone)



Fonte: autoria própria

Por fim, os resultados experimentais foram analisados para se obter as referências necessárias à comparação e validação do modelo numérico. Entretanto, foram realizados diversos ensaios para cada situação de impacto, possuindo assim diversos gráficos para cada experimento, a fim de se coletar dados mais consistentes sobre cada situação. Além disso, os dados coletados durante o experimento foram a posição da massa de impacto ao longo do tempo e a força transmitida à célula de carga. Como seria interessante se avaliar a velocidade da esfera antes e depois do impacto, para se estimar a quantidade de energia mecânica dissipada, seria preciso o cálculo da velocidade da esfera a partir da posição dela.

Com isso, foi desenvolvido um código em linguagem de programação MATLAB para estimar a velocidade e a aceleração da esfera. Para isso, foram utilizados três métodos de diferenciação numérica:

- Método das diferenças finitas (MDF): foram utilizados conjuntos de 3, 5 e 7 pontos para a estimativa da velocidade e 5, 7 e 9 pontos para a estimativa da aceleração
- Método de Lanczos: foram utilizados conjuntos de 5, 7 e 9 pontos para a estimativa da velocidade e 5, 7 e 9 pontos para a estimativa da aceleração
- Método de Smooth Noise-Robust Differentiation (SNRD): foram utilizados conjuntos de 5, 7 e 9 pontos para a estimativa da velocidade e 5, 7 e 9 pontos para a estimativa da aceleração

Após uma análise dos métodos acima, foi concluído que o que possuía a menor variação entre os ensaios, ou seja, resultados mais consistentes entre si, foi o método SNRD. Dessa forma, foi escolhido utilizar esse método como critério de comparação dos resultados. Isso significa que a referência de comparação entre a simulação numérica e os

ensaios experimentais será a simulação SNRD com 7 pontos, devido à característica de estabilidade do método e também por ser tomar na análise um bom número de pontos.

Para a estimativa da força, foi tomada a aceleração obtida via diferenciação numérica e seu produto com a massa da esfera resulta na força transmitida para a célula de carga. Dessa forma, foi calculada uma curva média de deslocamento da esfera no tempo com base nas medições realizadas. A partir dessa curva média, as velocidades e acelerações médias foram calculadas, assim como a estimativa da força transmitida. Com os resultados calculados, é possível ter uma boa referência dos valores que foram obtidos experimentalmente. Dessa forma, após a caracterização das curvas do ensaio, a simulação numérica foi otimizada para que as curvas de velocidade e força transmitida ficassem as mais próximas possíveis das obtidas via métodos de diferenciação numérica.

Por fim, uma vez caracterizado o Osso e a Pele, as simulações de impacto com os materiais de proteção facial foram implementadas e executadas. Para este projeto, as mesmas combinações de material usados por (SANTOIEMMA, 2018) foram utilizadas, isto é:

- Uma camada de material Flexível (F);
- Uma camada de material Rígido (R);
- Uma camada de material Flexível e uma de Rígido (FR);
- Uma camada de material Rígido, uma de Flexível e uma de Rígido (RFR);
- Uma camada de material Flexível, uma de Rígido e uma de Flexível (FRF);

As camadas de material foram padronizadas em 3 milímetros para a camada flexível e 1 milímetro para a rígida. Em seguida a simulação foi executada e os resultados podem ser conferidos abaixo

5 Resultados

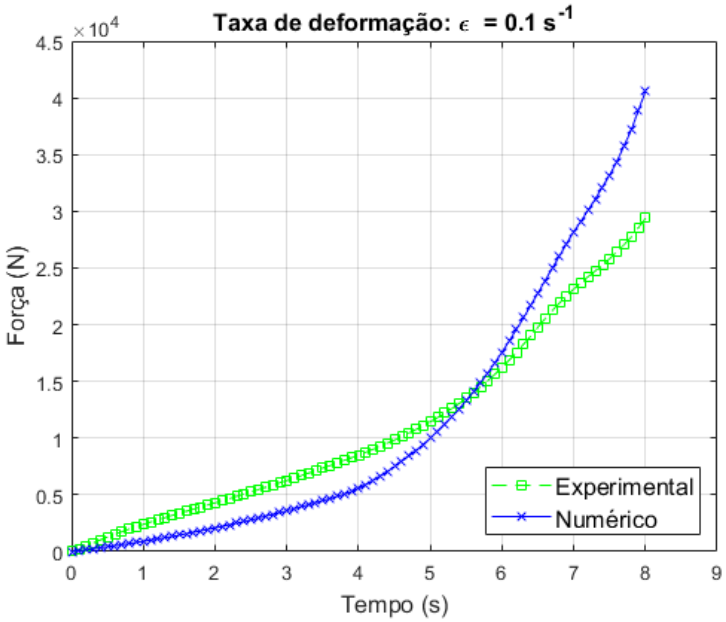
5.1 Calibração de Material

Para os resultados da calibração de material, os resultados são mostrados abaixo:

5.1.1 Material Flexível

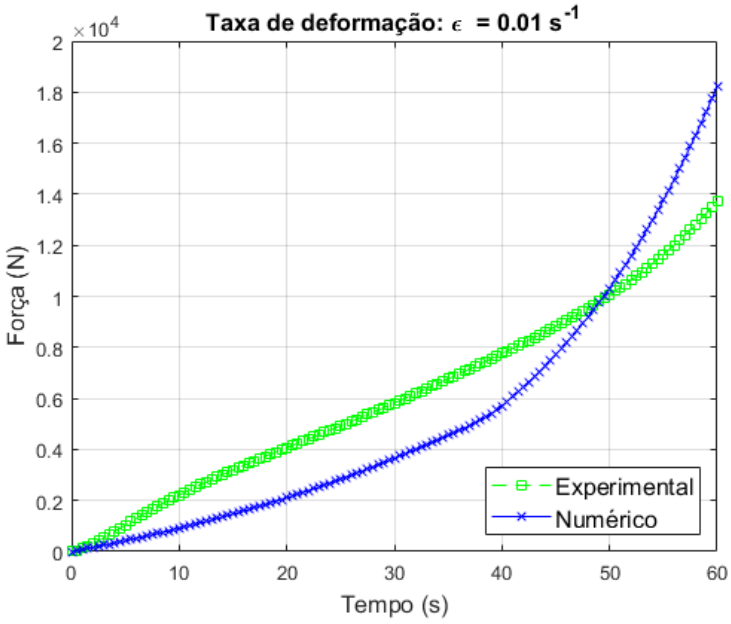
Após a otimização do material flexível, os gráficos abaixo mostram os resultados obtidos via otimização numérica e aqueles obtidos experimentalmente:

Figura 27 – Comparação dos resultados numéricos e experimentais do EVA Flexível para uma taxa de deformação de 0.1 s^{-1}



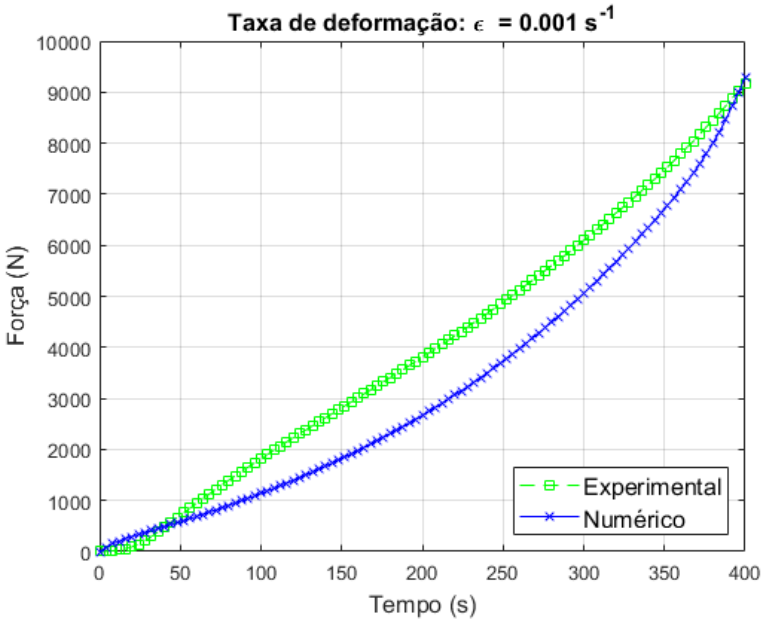
Fonte: autoria própria

Figura 28 – Comparação dos resultados numéricos e experimentais do EVA Flexível para uma taxa de deformação de 0.01 s^{-1}



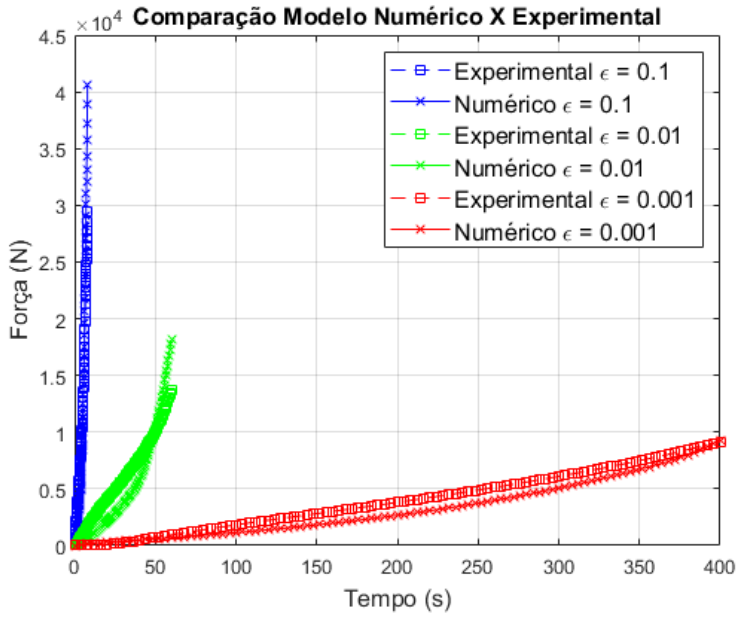
Fonte: autoria própria

Figura 29 – Comparação dos resultados numéricos e experimentais do EVA Flexível para uma taxa de deformação de 0.001 s^{-1}



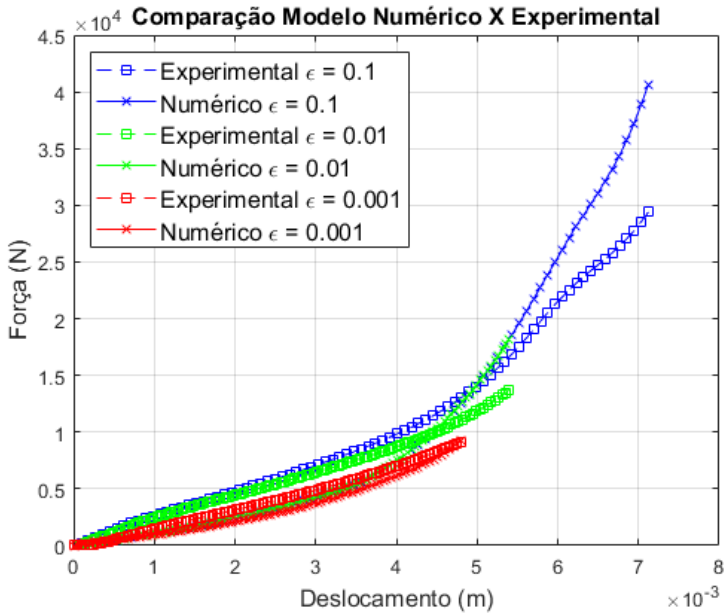
Fonte: autoria própria

Figura 30 – Comparação dos resultados (força x tempo) numéricos e experimentais do EVA Flexível para diferentes taxas de deformações



Fonte: autoria própria

Figura 31 – Comparação dos resultados (força x deslocamento) numéricos e experimentais do EVA Flexível para diferentes taxas de deformações



Fonte: autoria própria

Além disso, os parâmetros iniciais e otimizados do material são mostrados abaixo, a título de comparação:

Tabela 4 – Valores obtidos após a otimização

Valores	μ_0	λ_m	**D**
Iniciais	1.0E6	10.0	0.0
Otimizados	1.755E6	34.78	0.0

Fonte: autoria própria

Tabela 5 – Valores iniciais e otimizados da modelização da Série de Prony do material Flexível

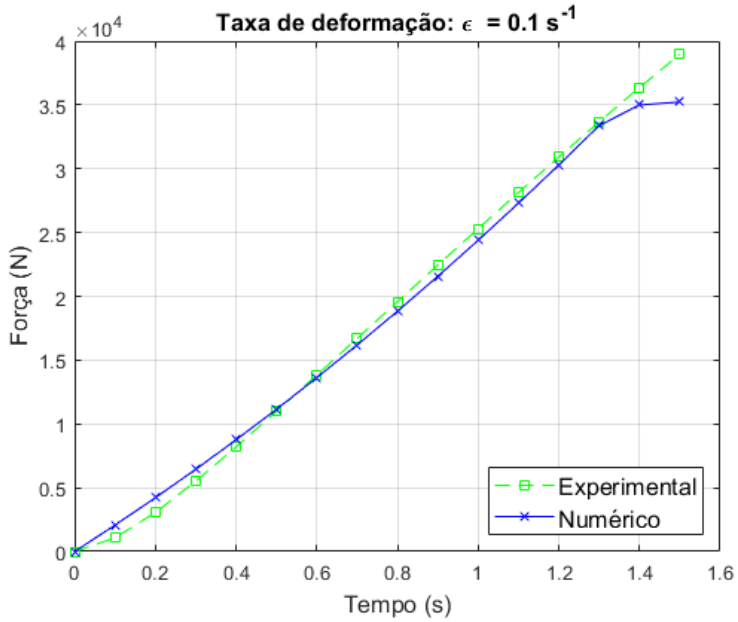
Termo	**1**	**2**	**3**	**4**
g_i **Inicial**	0.1	0.1	0.1	0.1
g_i **Otimizado**	0.28	0.2	0.186	0.1
k_i **Inicial**	0.1	0.1	0.1	0.1
k_i **Otimizado**	0.0	0.0	0.0	0.0
τ_i **Inicial**	1.0	1.0	1.0	1.0
τ_i **Otimizado**	5.0E-5	6.47E-4	0.011	7.321

Fonte: autoria própria

5.1.2 Material Rígido

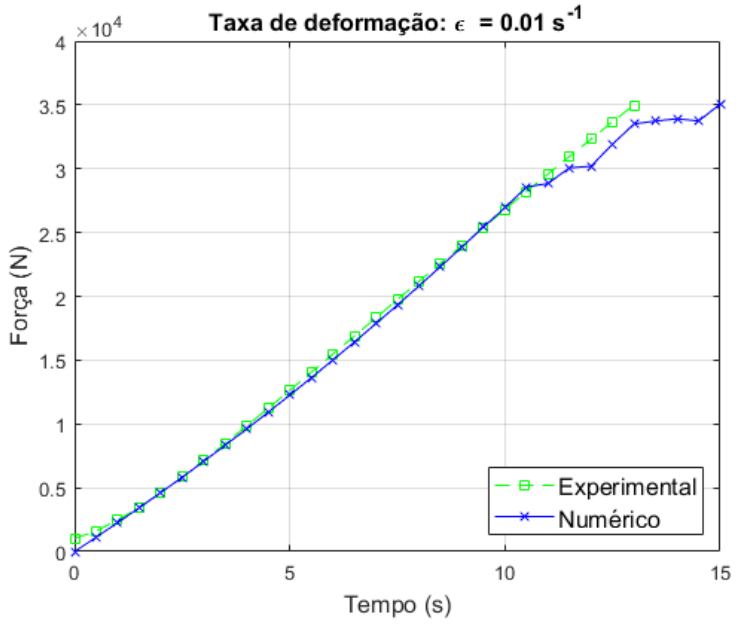
Abaixo podem ser conferidos os resultados do mesmo processo de calibração mas agora feito para o material rígido:

Figura 32 – Comparação dos resultados numéricos e experimentais do EVA Rígido para uma taxa de deformação de 0.1 s^{-1}



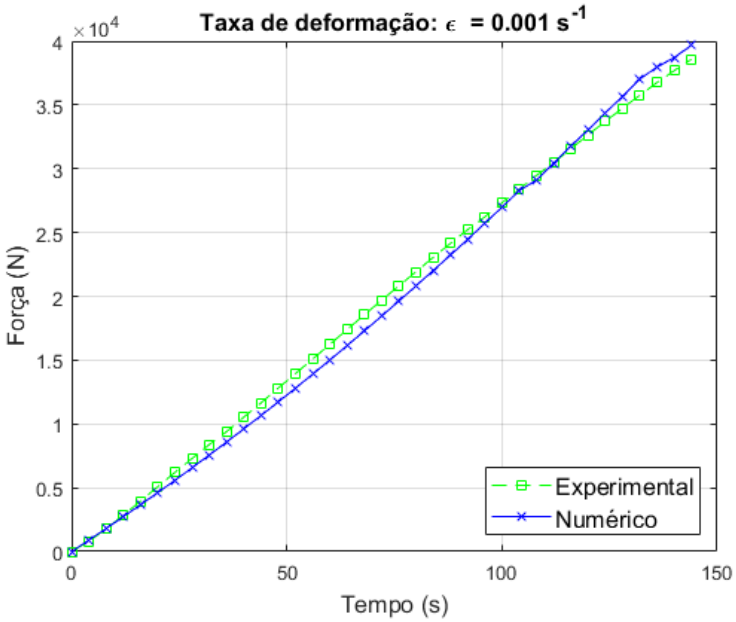
Fonte: autoria própria

Figura 33 – Comparação dos resultados numéricos e experimentais do EVA Rígido para uma taxa de deformação de 0.01 s^{-1}



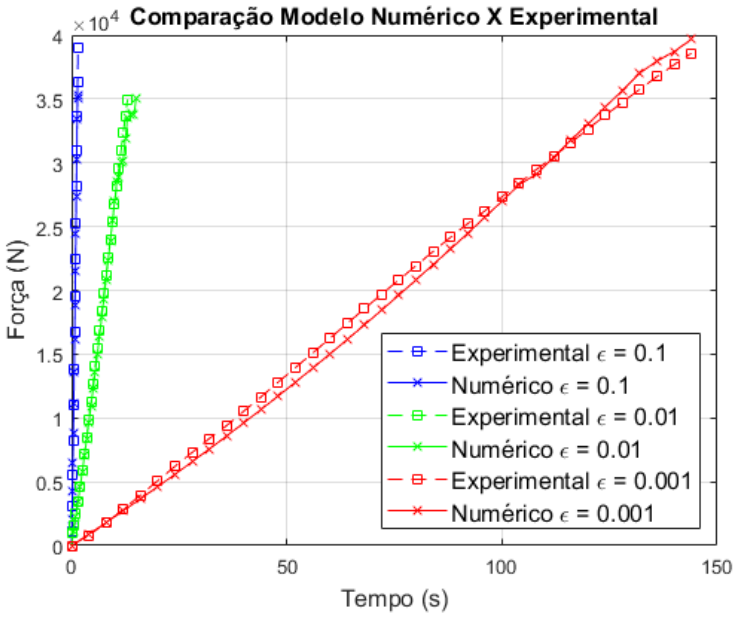
Fonte: autoria própria

Figura 34 – Comparação dos resultados numéricos e experimentais do EVA Rígido para uma taxa de deformação de 0.001 s^{-1}



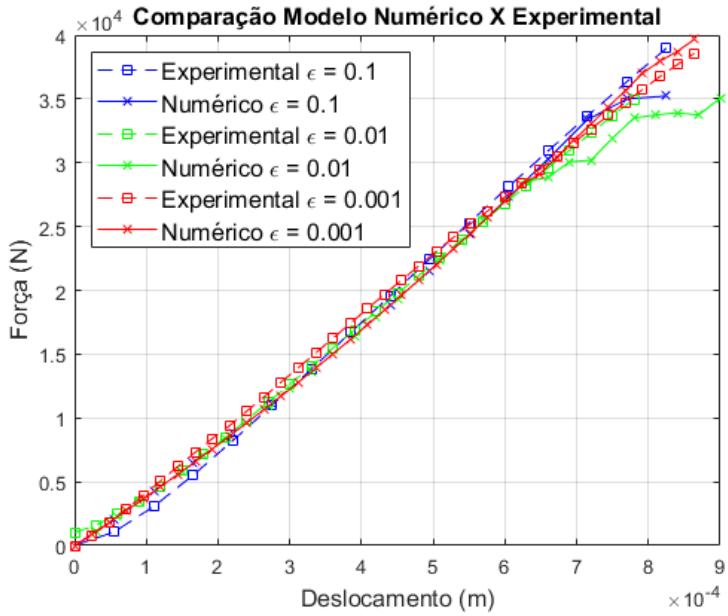
Fonte: autoria própria

Figura 35 – Comparação dos resultados numéricos e experimentais do EVA Rígido para diferentes taxas de deformações



Fonte: autoria própria

Figura 36 – Comparação dos resultados numéricos e experimentais do EVA Rígido para diferentes taxas de deformações



Fonte: autoria própria

E por fim, os resultados da otimização são mostrados abaixo:

Tabela 6 – Valores do material rígido obtidos após a otimização

Valores	**E**	ν
Iniciais	1.0E6	0.49
Otimizados	65.57E6	0.499

Fonte: autoria própria

5.1.3 Osso e Pele

Com relação à caracterização do osso, a análise foi feita com base nos ensaios experimentais de impacto, diferentemente do abordado com os materiais flexível e rígido, devido à falta de dados. Assim, temos que:

Tabela 7 – Valores do Osso obtidos após a otimização

Valores	**E**	ν
Iniciais	45.0E6	0.49
Otimizados	15.32E6	0.49

Fonte: autoria própria

Tabela 8 – Valores iniciais e otimizados da modelização da Série de Prony do Osso

Termo	**1**
g_i **Inicial**	0.5
g_i **Otimizado**	0.305
k_i **Inicial**	0.1
k_i **Otimizado**	0.01
τ_i **Inicial**	1.0
τ_i **Otimizado**	1.0E-5

Fonte: autoria própria

Os resultados para a otimização da Pele foram os seguintes:

Tabela 9 – Valores do Pele obtidos após a otimização

Valores	**E**	ν
Iniciais	1.0E6	0.48
Otimizados	0.7E6	0.499

Fonte: autoria própria

Tabela 10 – Valores iniciais e otimizados da modelização da Série de Prony da Pele

Termo	**1**	**2**	**3**	**4**
g_i **Inicial**	0.1	0.1	0.1	0.1
g_i **Otimizado**	0.051	0.1	0.12	0.019
k_i **Inicial**	0.1	0.1	0.1	0.1
k_i **Otimizado**	0.02	0.15	0.12	0.019
τ_i **Inicial**	1.0	1.0	1.0	1.0
τ_i **Otimizado**	6.3E-4	3.66E-2	13.10	94.56

Fonte: autoria própria

Uma vez validados os materiais, as simulações de impacto foram executadas com as propriedades ótimas do material utilizados

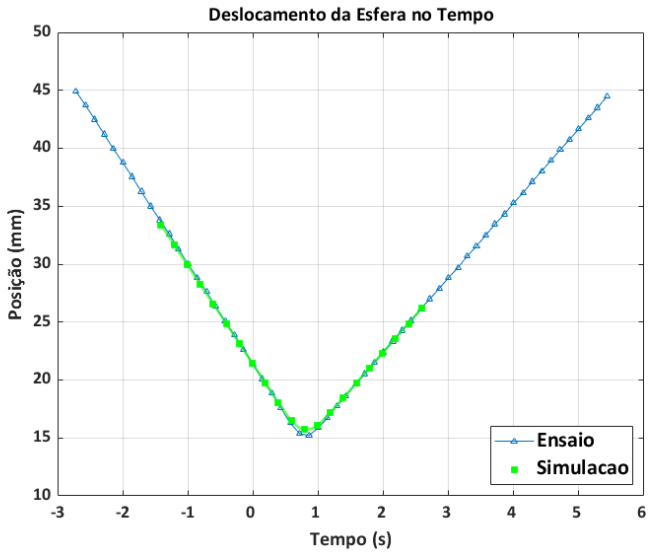
5.2 Simulações de Impacto

Nesta seção são apresentados os resultados obtidos nas simulações de impacto realizadas numericamente. Serão apresentados também os resultados das diferenciações numéricas feitas sobre a posição da esfera de impacto no tempo. Tomou-se como referência de velocidade e força transmitida experimentais os gráficos obtidos com 7 pontos do método SNRD, pois se apresentaram como os resultados com menor variação entre si. Os resultados marcados como **Opt** representam os resultados **otimizados**, enquanto que os outros dados representam os resultados experimentais tratados através dos métodos de diferenciação numérica mostradas acima.

Dessa forma, os resultados obtidos são mostrados abaixo

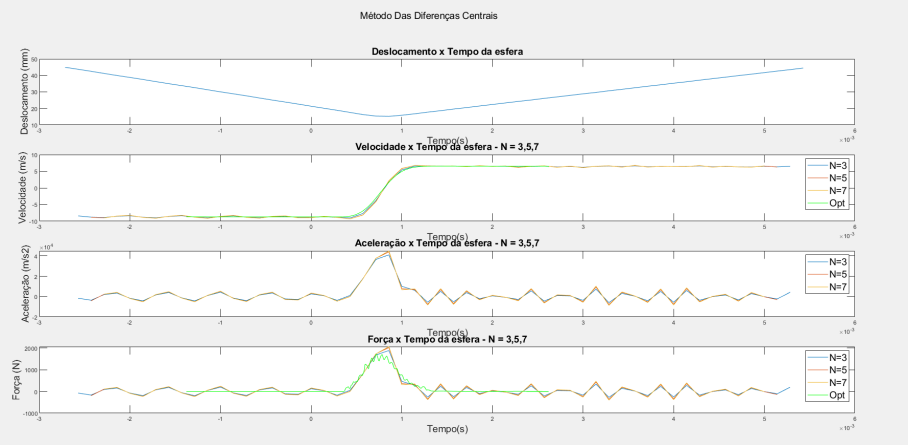
5.2.1 Osso

Figura 37 – Comparação das posições da esfera obtidas nos Ensaios e Numericamente



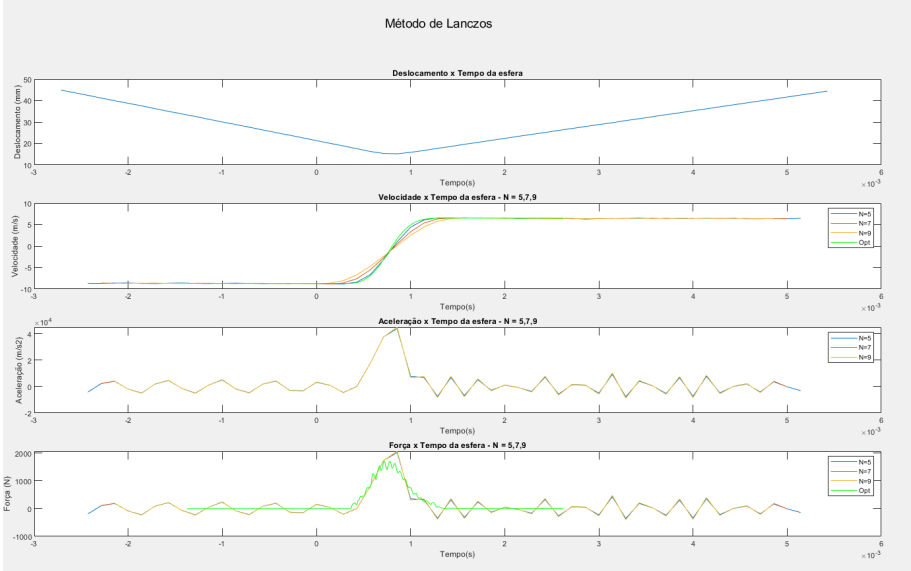
Fonte: autoria própria

Figura 38 – Comparação de valores obtidos Numericamente e pelo método das MDC



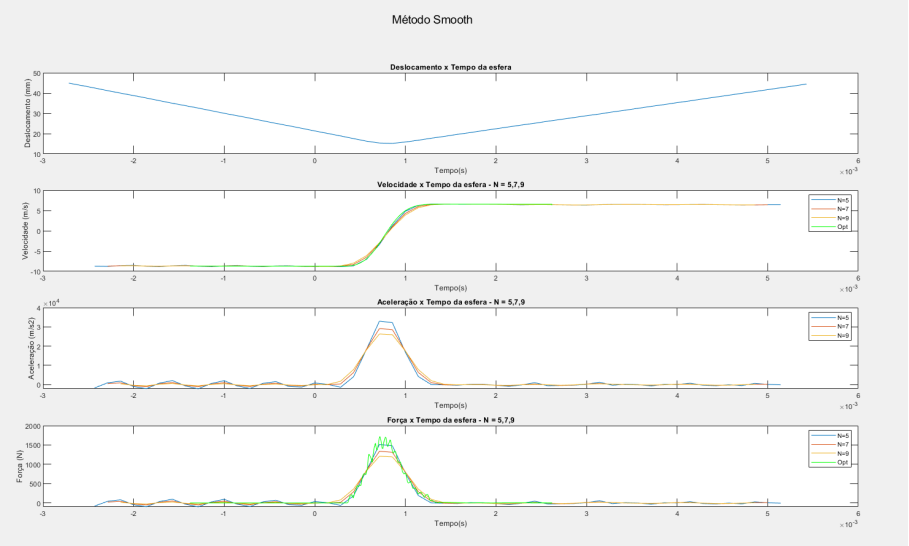
Fonte: autoria própria

Figura 39 – Comparação de valores obtidos numericamente e pelo método de Lanczos



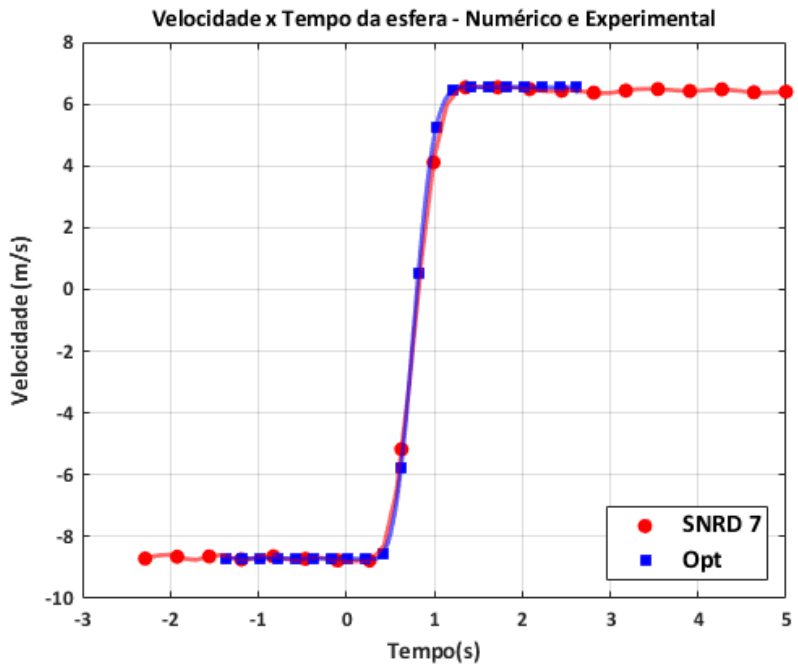
Fonte: autoria própria

Figura 40 – Comparação de valores obtidos numericamente e pelo método de SNDR



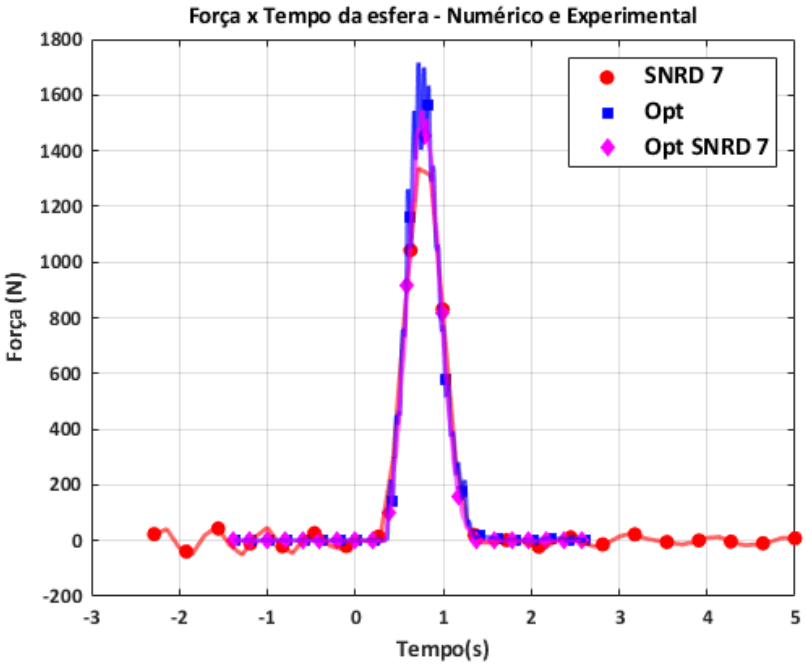
Fonte: autoria própria

Figura 41 – Detalhe sobre a comparação de velocidades entre a simulação numérica e o método SNDR com N = 7



Fonte: autoria própria

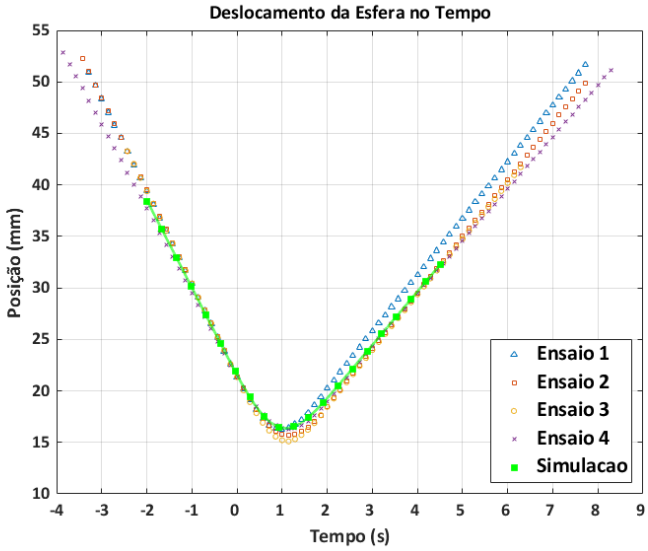
Figura 42 – Detalhe sobre a comparação de força transmitida entre a simulação numérica, força numérica filtrada com SNDR e o método SNDR com N = 7



Fonte: autoria própria

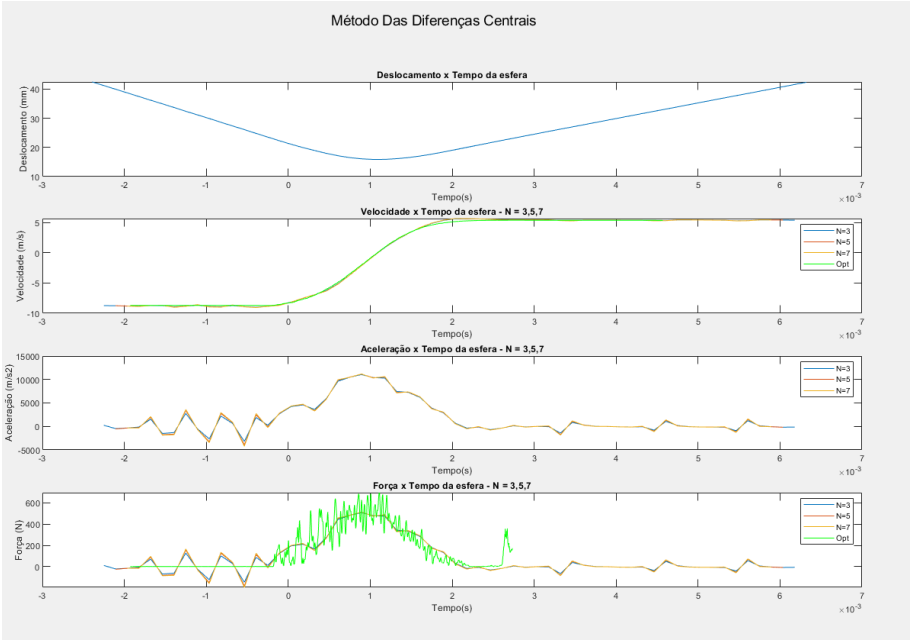
5.2.2 Osso + Pele

Figura 43 – Comparação das posições da esfera obtidas nos Ensaios e Numericamente



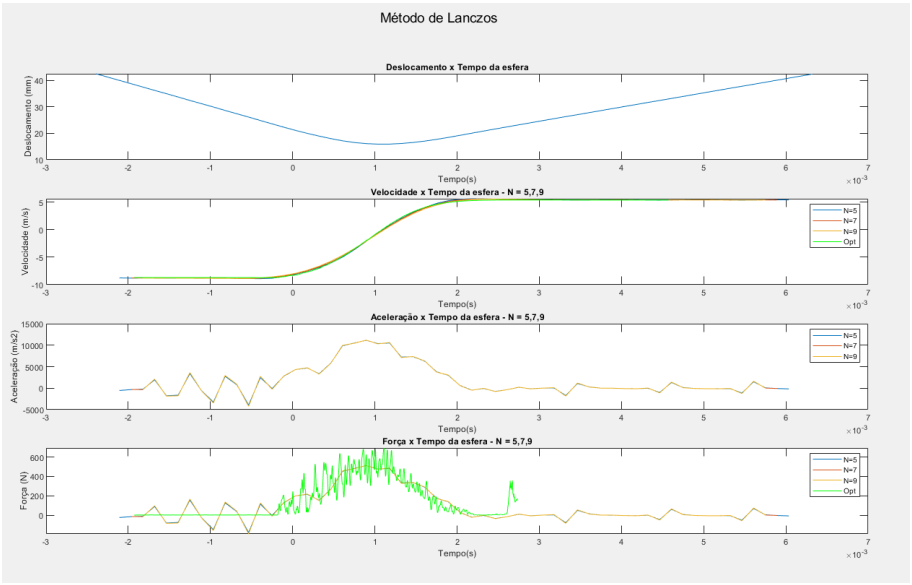
Fonte: autoria própria

Figura 44 – Comparação de valores obtidos Numericamente e pelo método das MDC



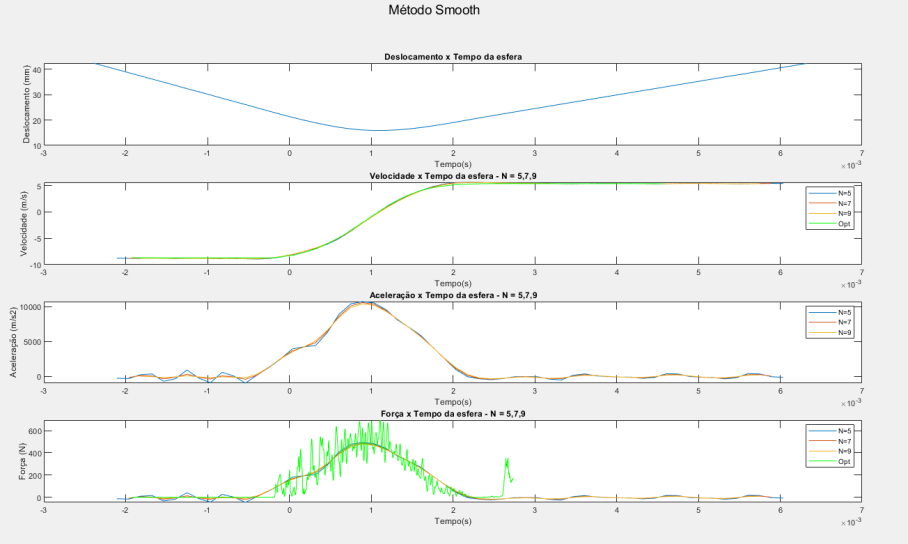
Fonte: autoria própria

Figura 45 – Comparação de valores obtidos numericamente e pelo método de Lanczos



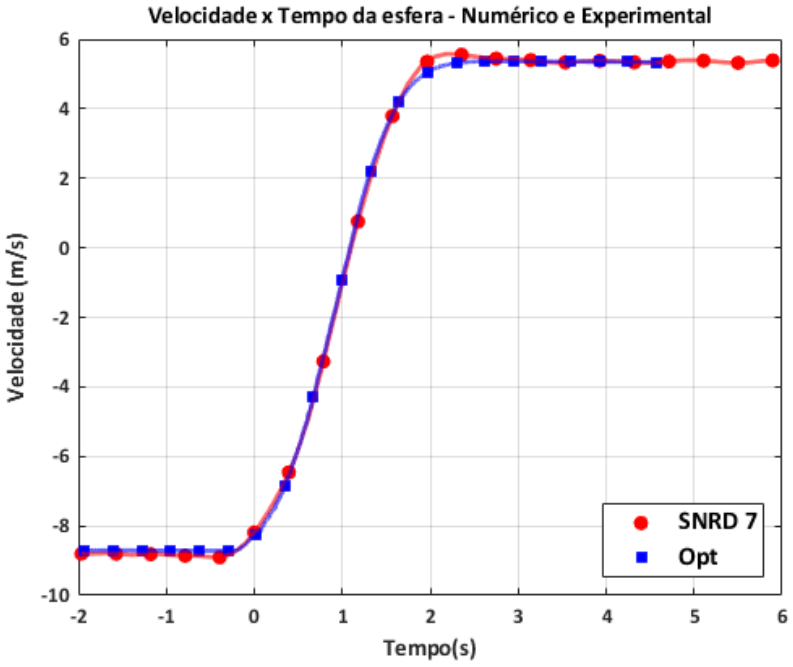
Fonte: autoria própria

Figura 46 – Comparação de valores obtidos numericamente e pelo método de SNDR



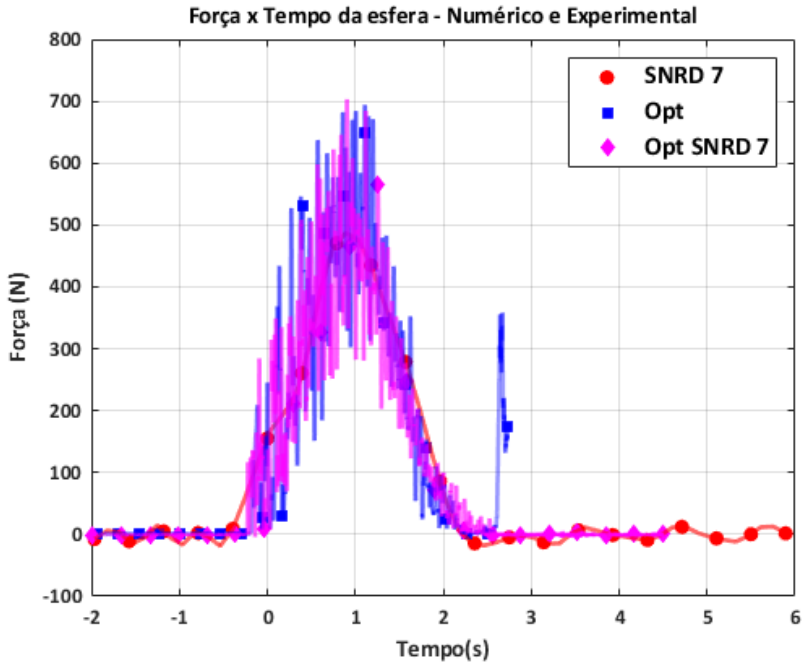
Fonte: autoria própria

Figura 47 – Detalhe sobre a comparação de velocidades entre a simulação numérica e o método SNDR com N = 7



Fonte: autoria própria

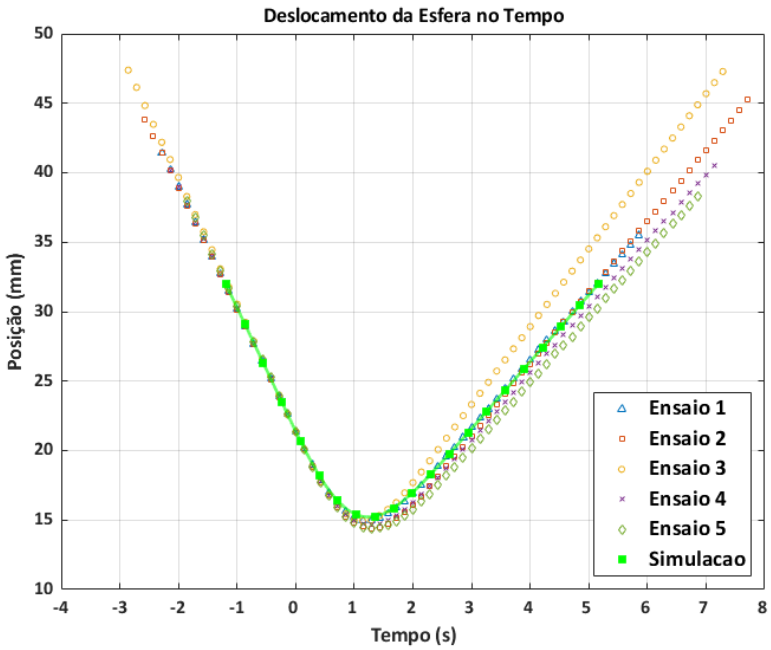
Figura 48 – Detalhe sobre a comparação de força transmitida entre a simulação numérica, força numérica filtrada com SNDR e o método SNDR com N = 7



Fonte: autoria própria

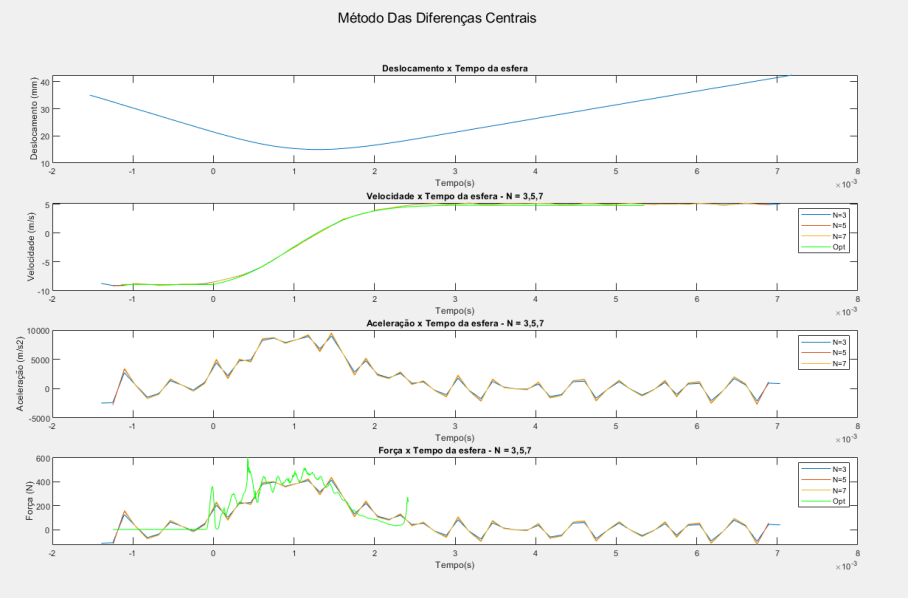
5.2.3 Camada Flexível (F)

Figura 49 – Comparação das posições da esfera obtidas nos Ensaios e Numericamente



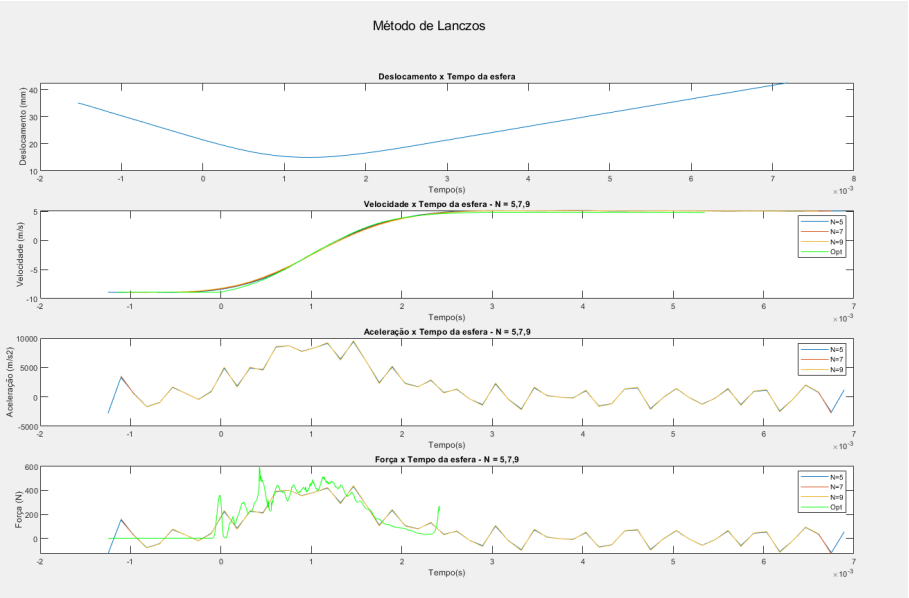
Fonte: autoria própria

Figura 50 – Comparação de valores obtidos Numericamente e pelo método das MDC



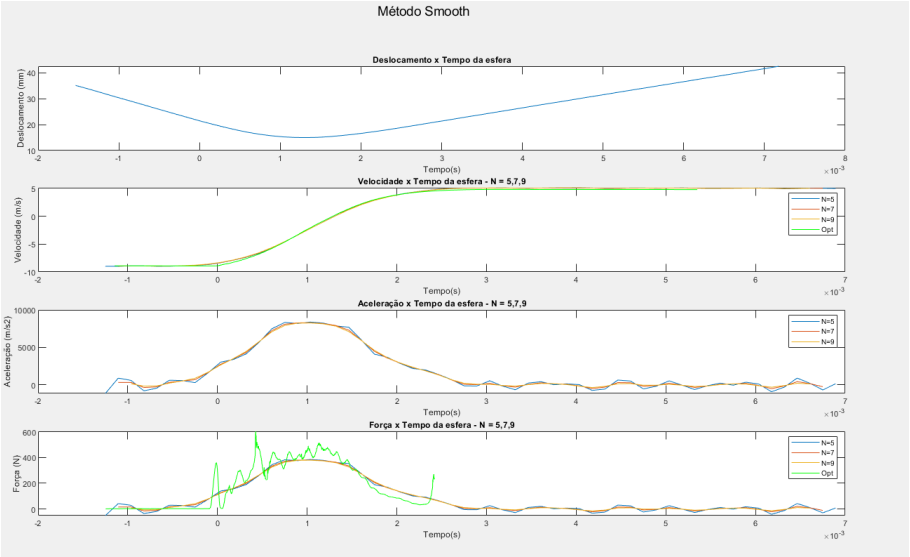
Fonte: autoria própria

Figura 51 – Comparação de valores obtidos numericamente e pelo método de Lanczos



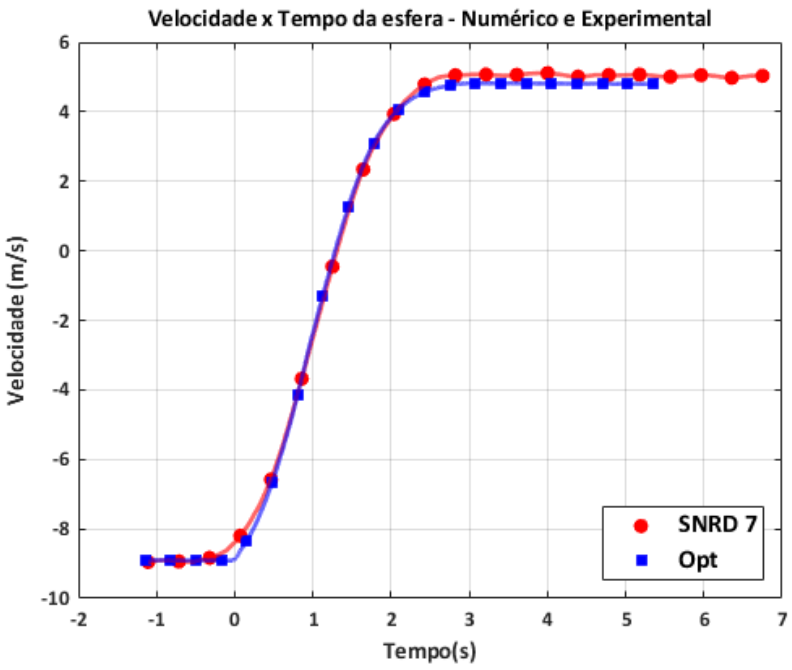
Fonte: autoria própria

Figura 52 – Comparação de valores obtidos numericamente e pelo método de SNDR



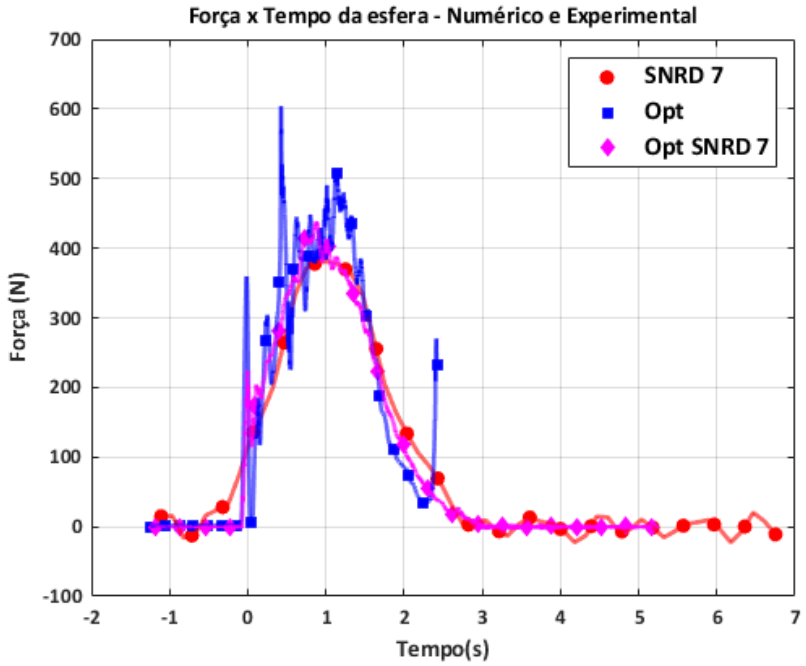
Fonte: autoria própria

Figura 53 – Detalhe sobre a comparação de velocidades entre a simulação numérica e o método SNDR com N = 7



Fonte: autoria própria

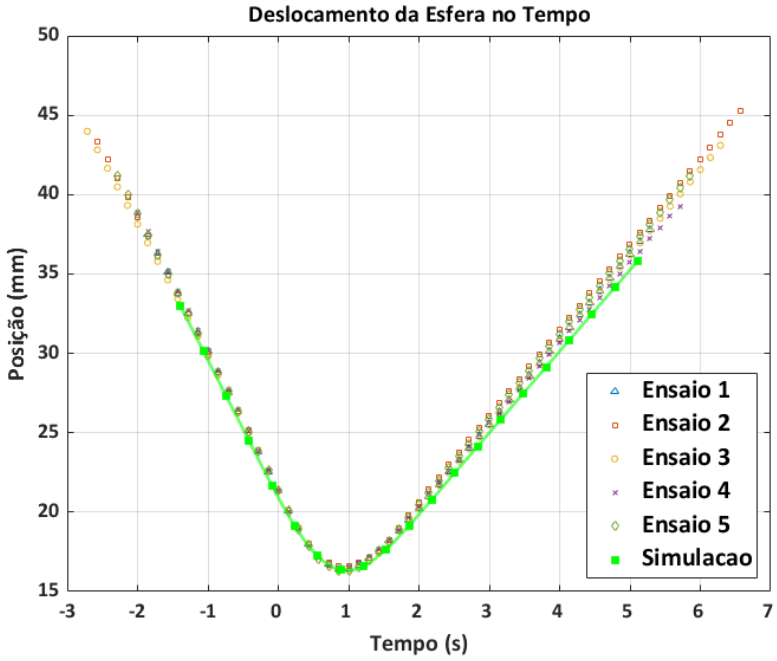
Figura 54 – Detalhe sobre a comparação de força transmitida entre a simulação numérica, força numérica filtrada com SNDR e o método SNDR com N = 7



Fonte: autoria própria

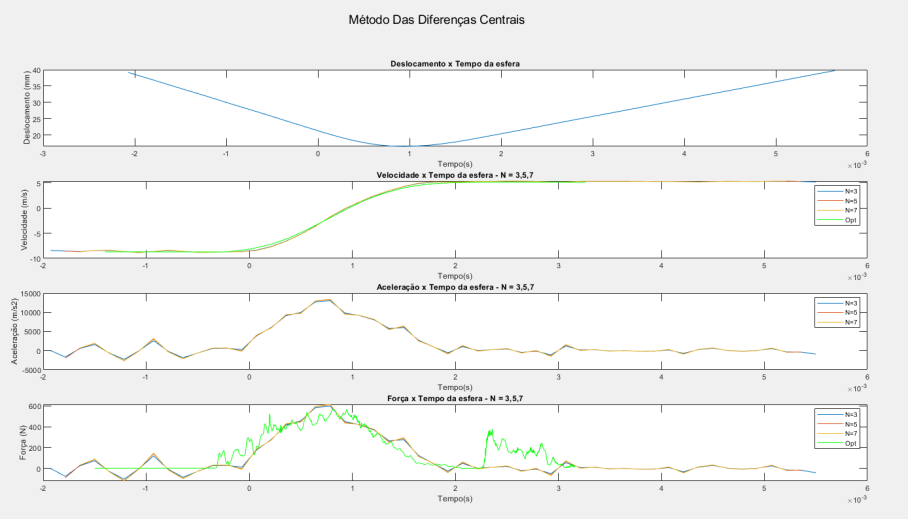
5.2.4 Camada Rígida (R)

Figura 55 – Comparação das posições da esfera obtidas nos Ensaios e Numericamente



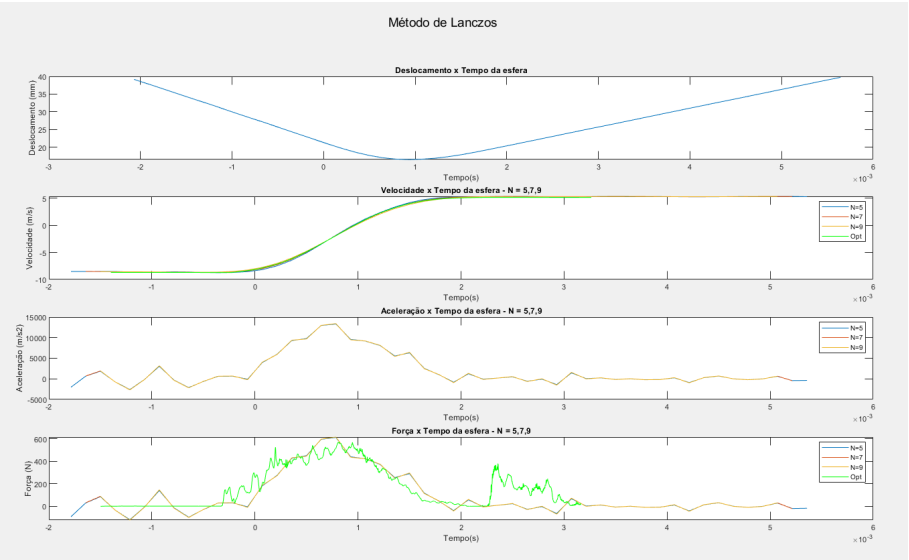
Fonte: autoria própria

Figura 56 – Comparação de valores obtidos Numericamente e pelo método das MDC



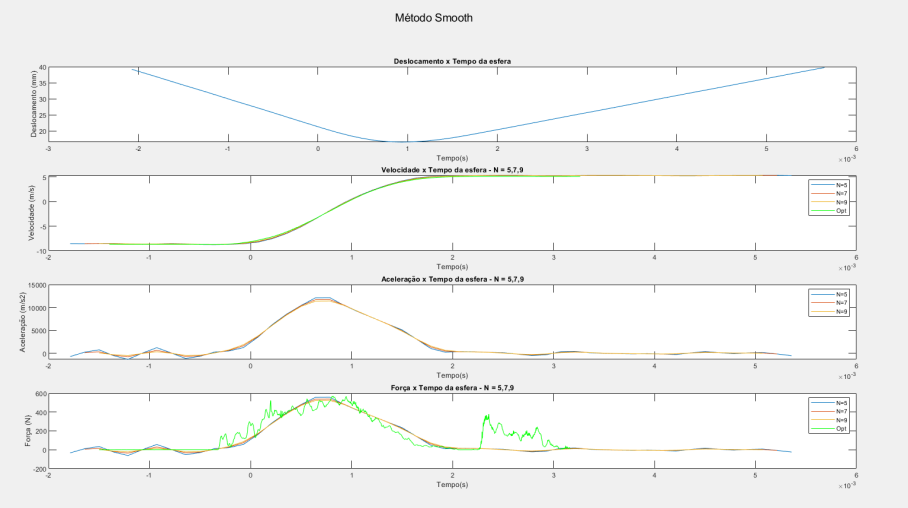
Fonte: autoria própria

Figura 57 – Comparação de valores obtidos numericamente e pelo método de Lanczos



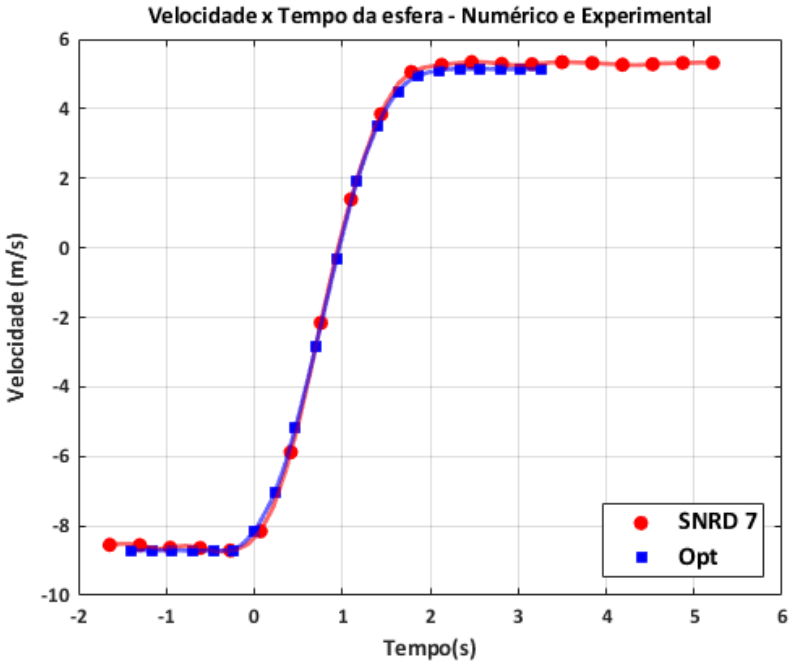
Fonte: autoria própria

Figura 58 – Comparação de valores obtidos numericamente e pelo método de SNDR



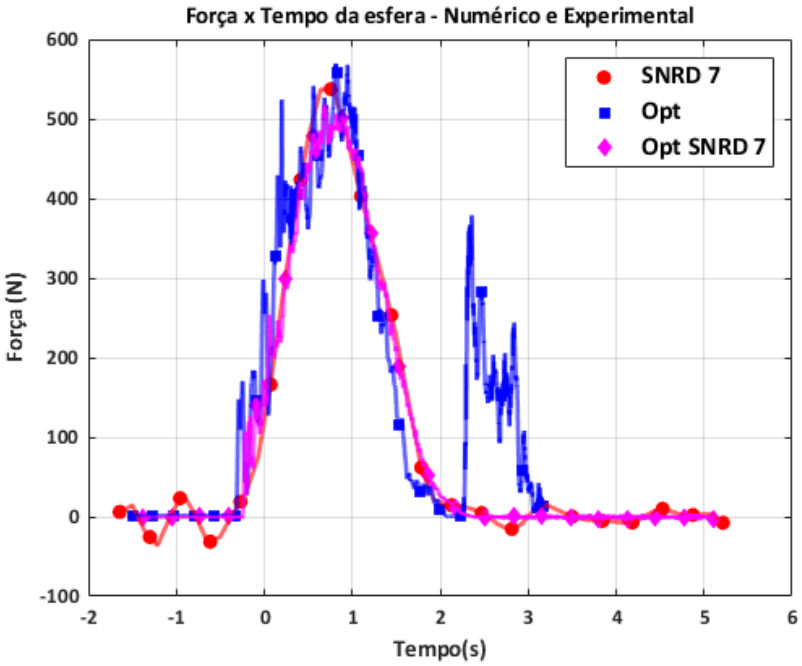
Fonte: autoria própria

Figura 59 – Detalhe sobre a comparação de velocidades entre a simulação numérica e o método SNDR com N = 7



Fonte: autoria própria

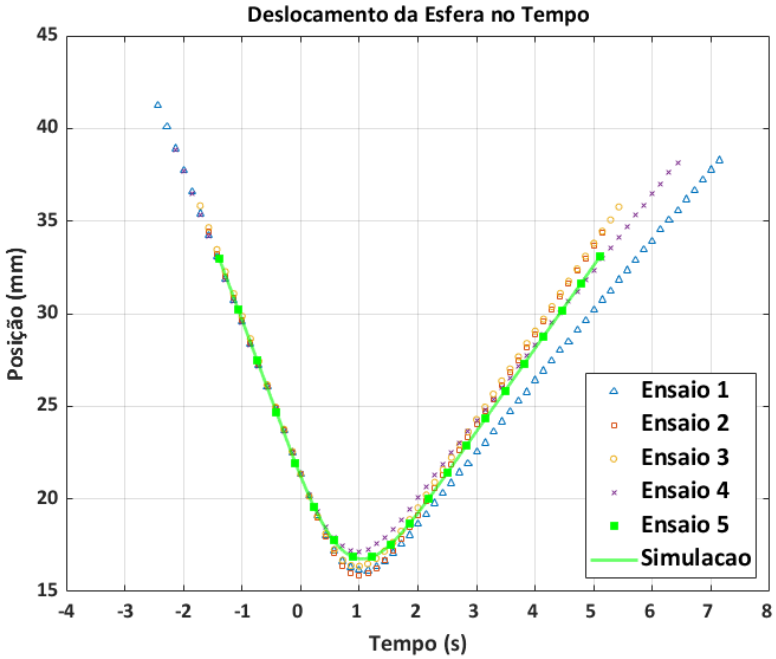
Figura 60 – Detalhe sobre a comparação de força transmitida entre a simulação numérica, força numérica filtrada com SNDR e o método SNDR com N = 7



Fonte: autoria própria

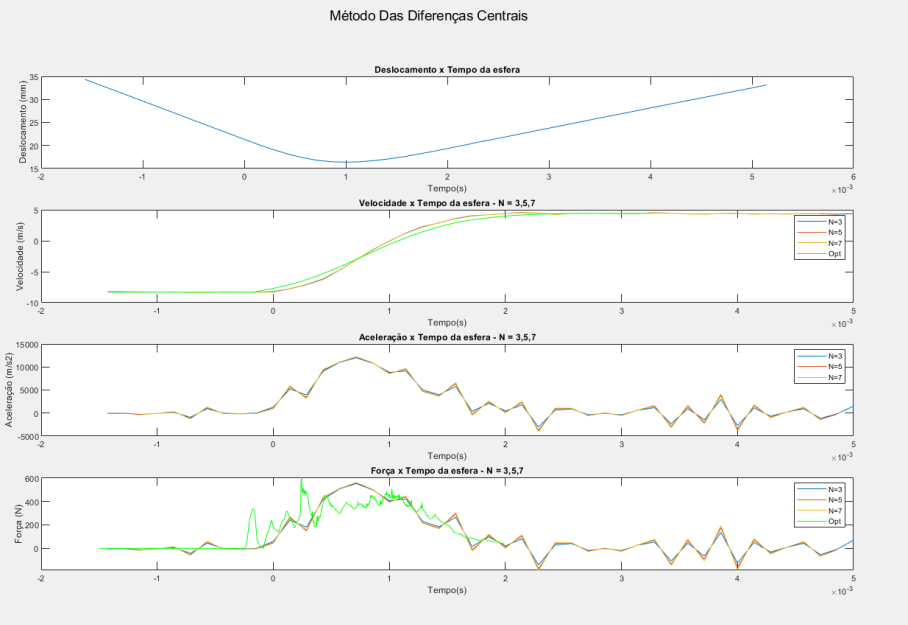
5.2.5 Camada Flexível + Rígido (FR)

Figura 61 – Comparação das posições da esfera obtidas nos Ensaios e Numericamente



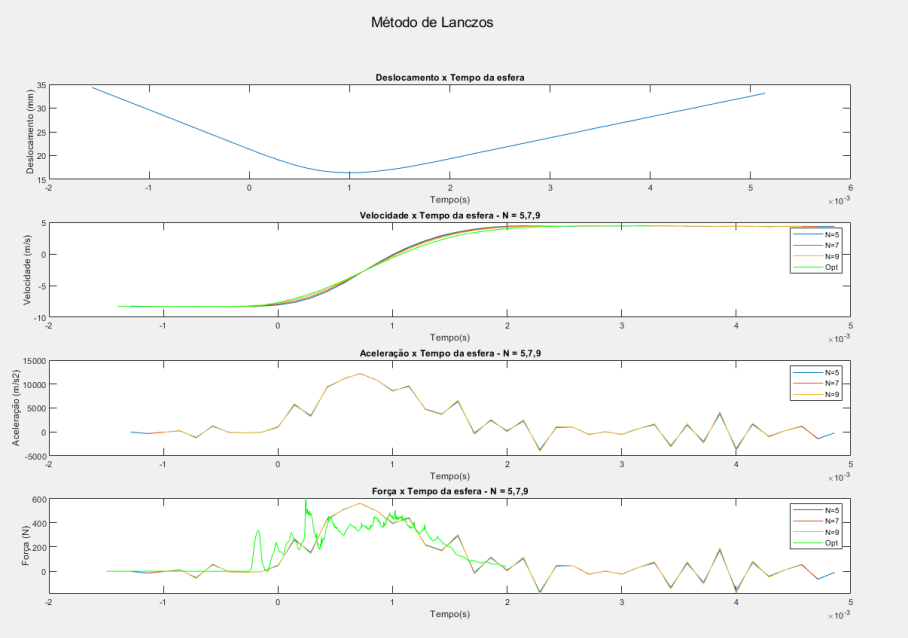
Fonte: autoria própria

Figura 62 – Comparação de valores obtidos Numericamente e pelo método das MDC



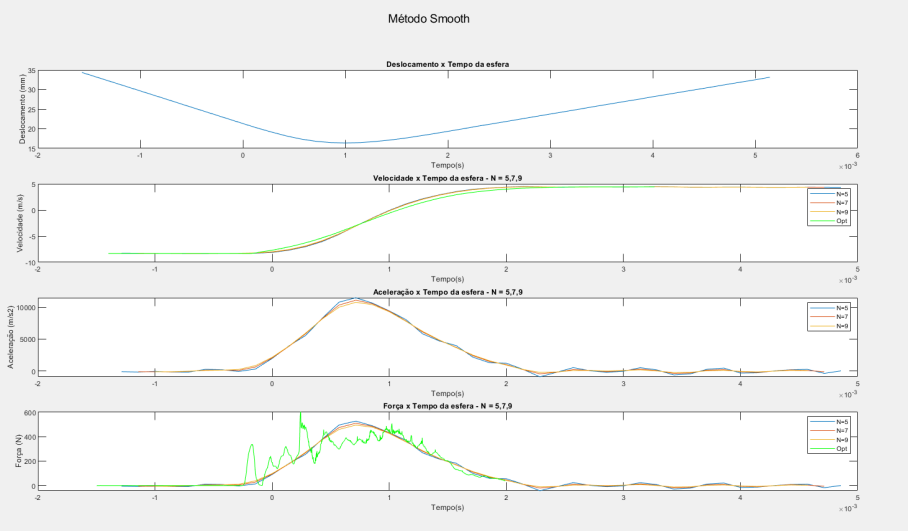
Fonte: autoria própria

Figura 63 – Comparação de valores obtidos numericamente e pelo método de Lanczos



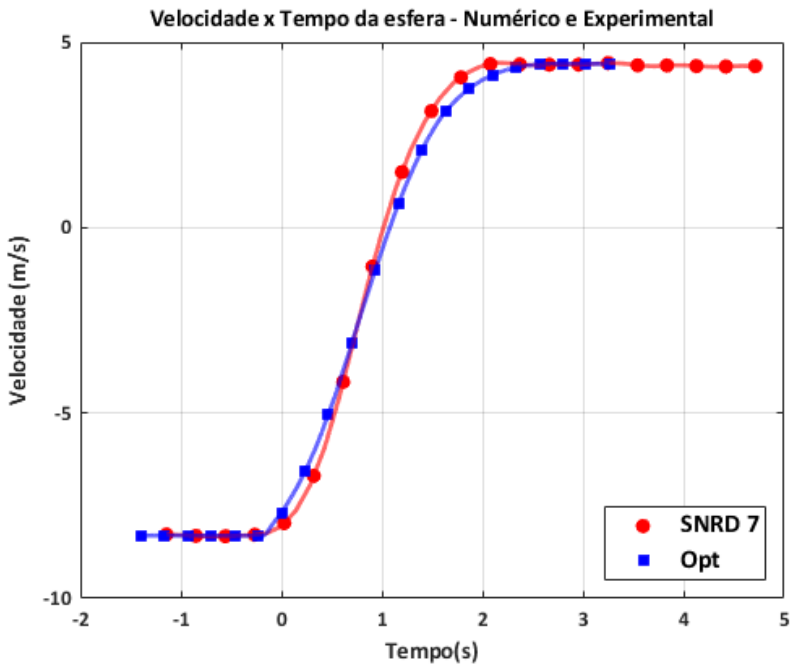
Fonte: autoria própria

Figura 64 – Comparação de valores obtidos numericamente e pelo método de SNDR



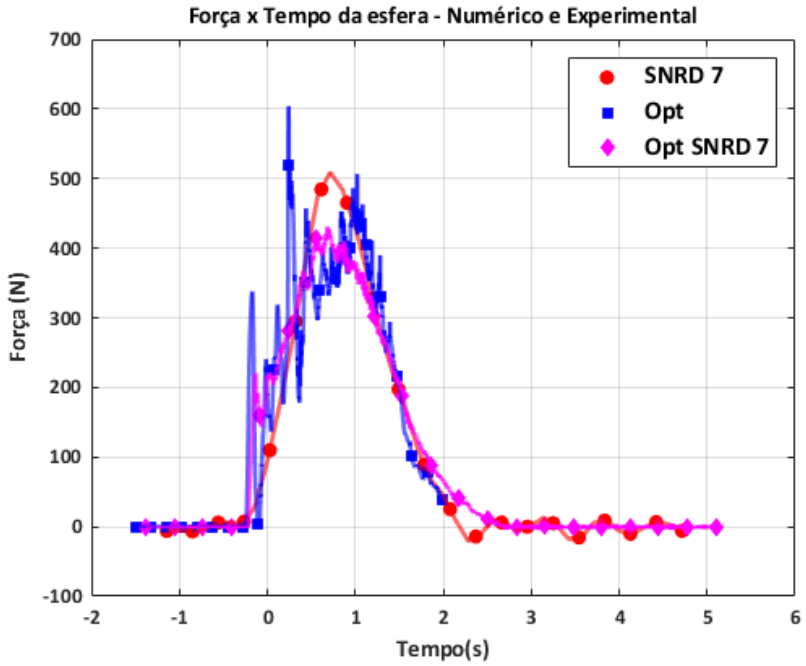
Fonte: autoria própria

Figura 65 – Detalhe sobre a comparação de velocidades entre a simulação numérica e o método SNDR com N = 7



Fonte: autoria própria

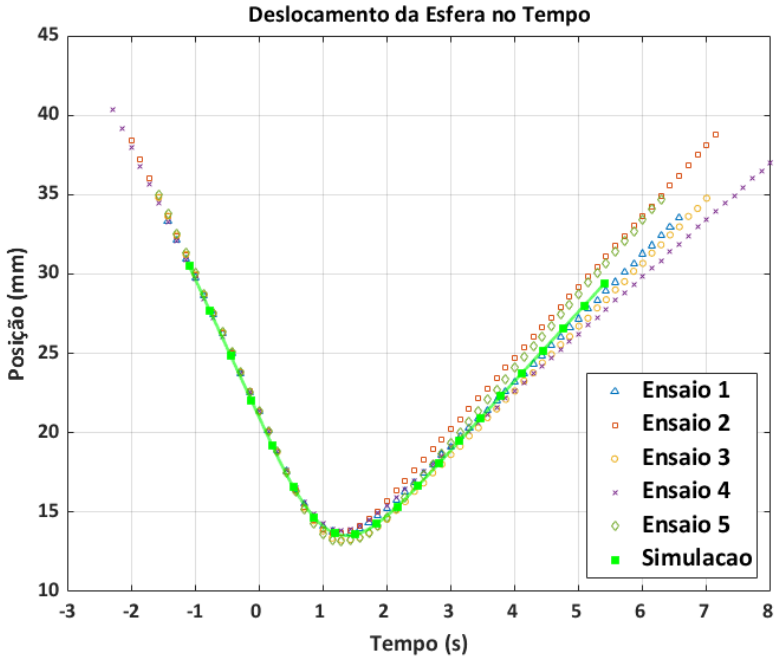
Figura 66 – Detalhe sobre a comparação de força transmitida entre a simulação numérica, força numérica filtrada com SNDR e o método SNDR com N = 7



Fonte: autoria própria

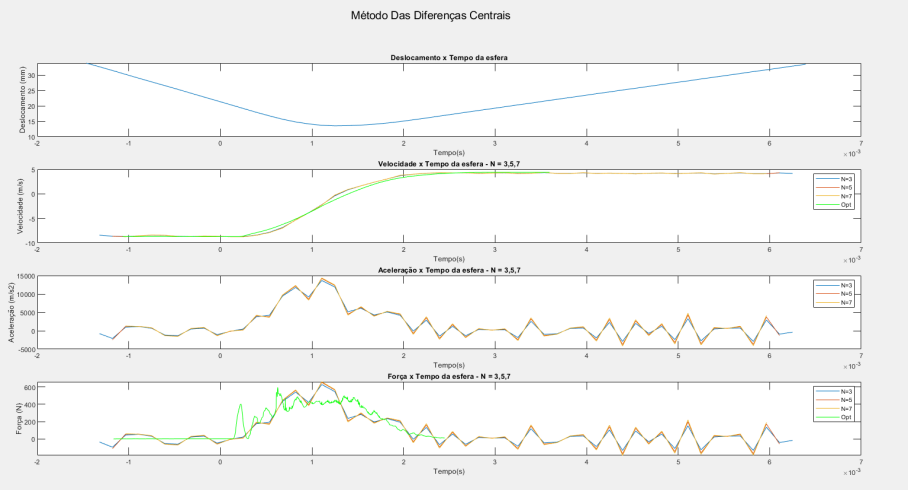
5.2.6 Camada Rígido + Flexível + Rígido (RFR)

Figura 67 – Comparação das posições da esfera obtidas nos Ensaios e Numericamente



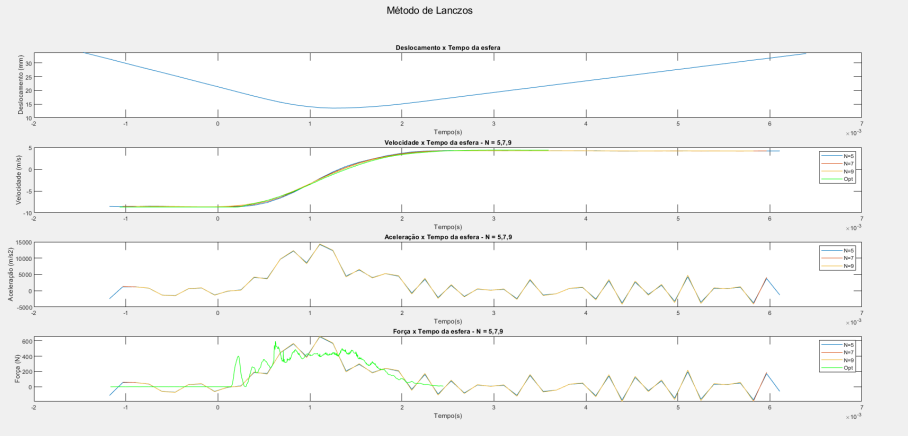
Fonte: autoria própria

Figura 68 – Comparação de valores obtidos Numericamente e pelo método das MDC



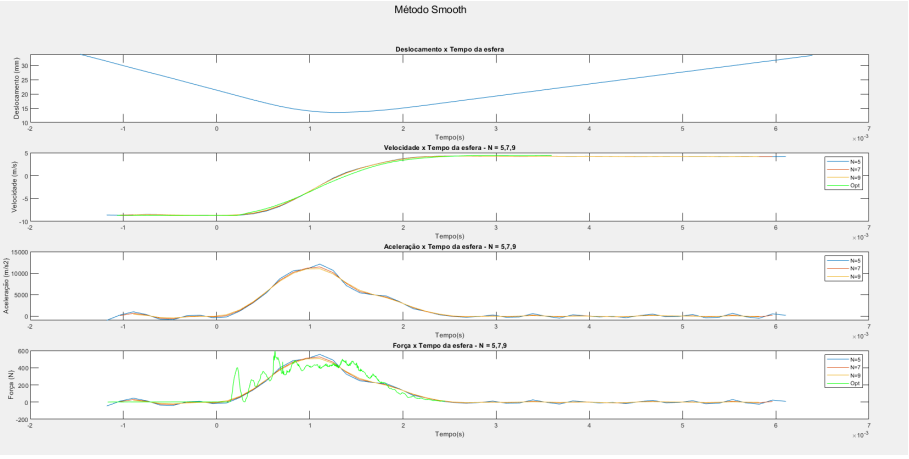
Fonte: autoria própria

Figura 69 – Comparação de valores obtidos numericamente e pelo método de Lanczos



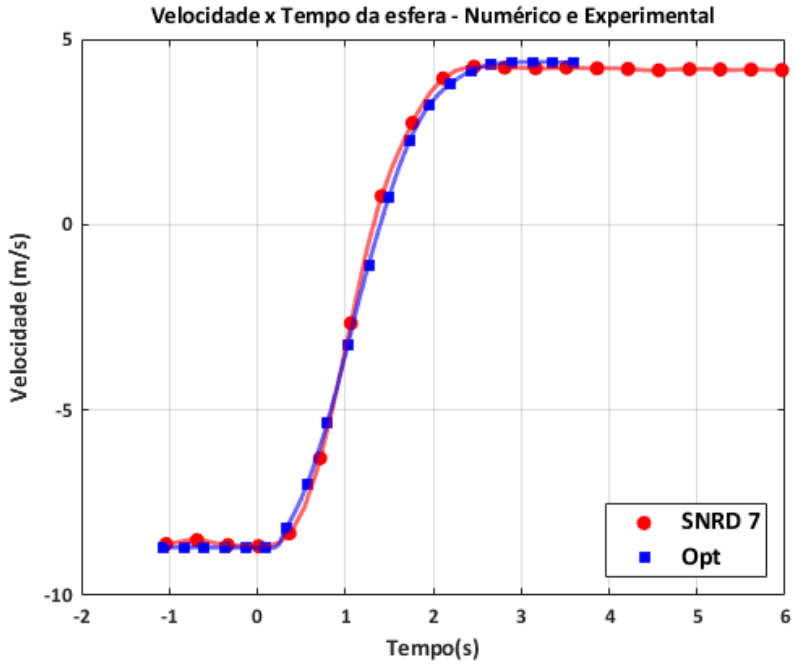
Fonte: autoria própria

Figura 70 – Comparação de valores obtidos numericamente e pelo método de SNDR



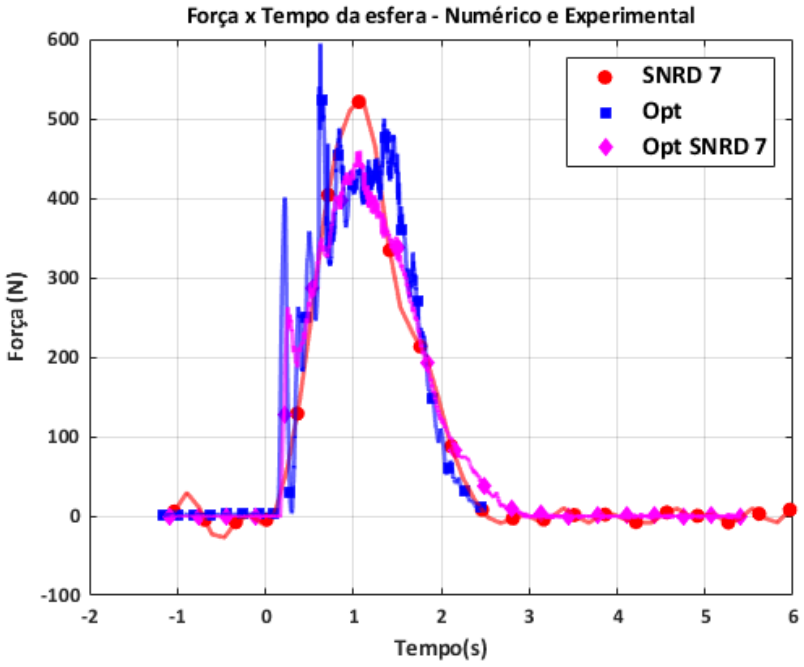
Fonte: autoria própria

Figura 71 – Detalhe sobre a comparação de velocidades entre a simulação numérica e o método SNDR com N = 7



Fonte: autoria própria

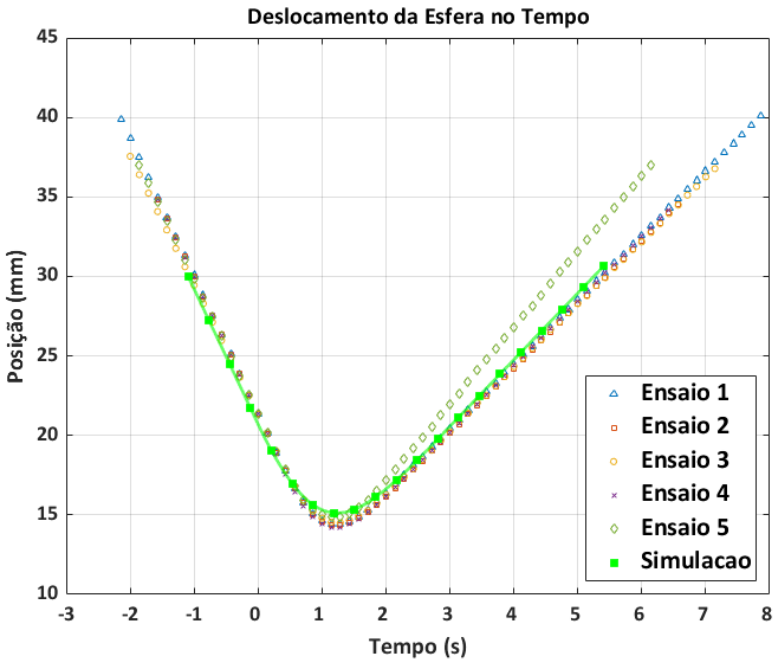
Figura 72 – Detalhe sobre a comparação de força transmitida entre a simulação numérica, força numérica filtrada com SNDR e o método SNDR com N = 7



Fonte: autoria própria

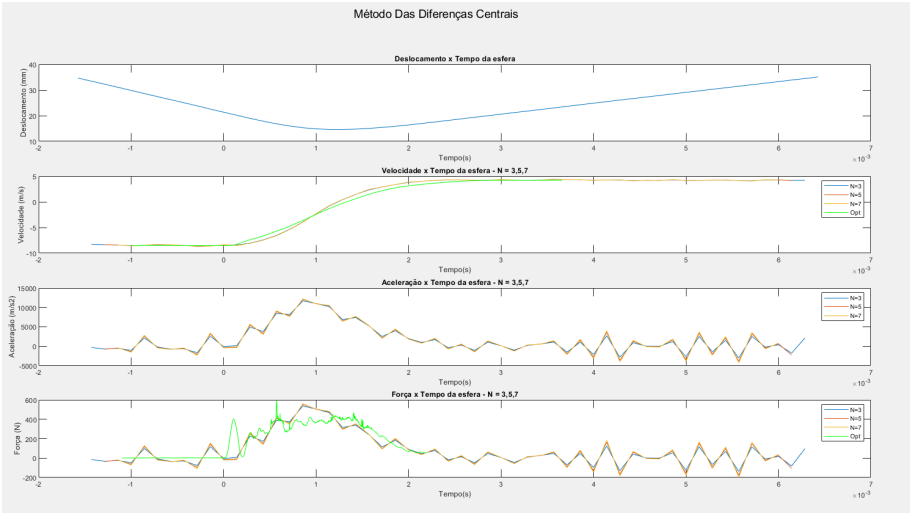
5.2.7 Camada Flexível + Rígido + Flexível (FRF)

Figura 73 – Comparação das posições da esfera obtidas nos Ensaios e Numericamente



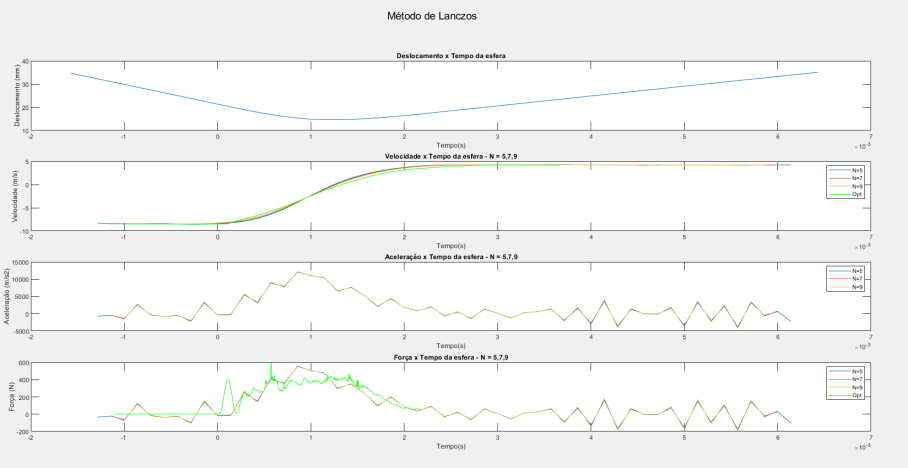
Fonte: autoria própria

Figura 74 – Comparação de valores obtidos Numericamente e pelo método das MDC



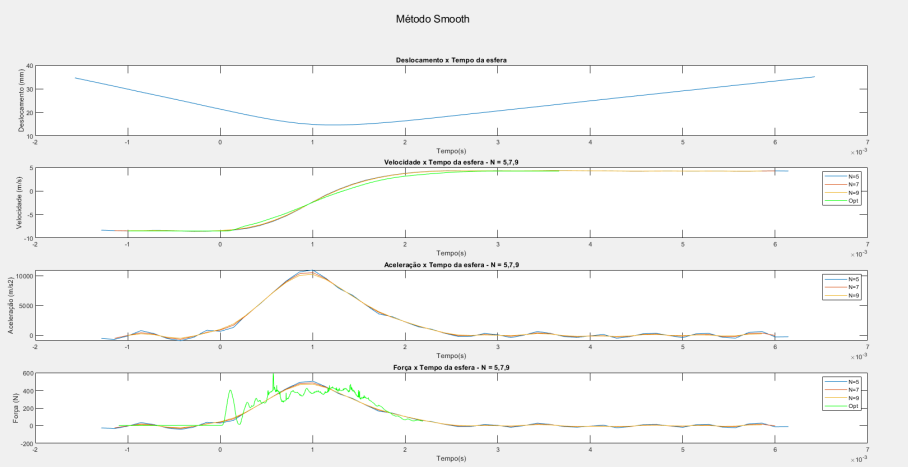
Fonte: autoria própria

Figura 75 – Comparação de valores obtidos numericamente e pelo método de Lanczos



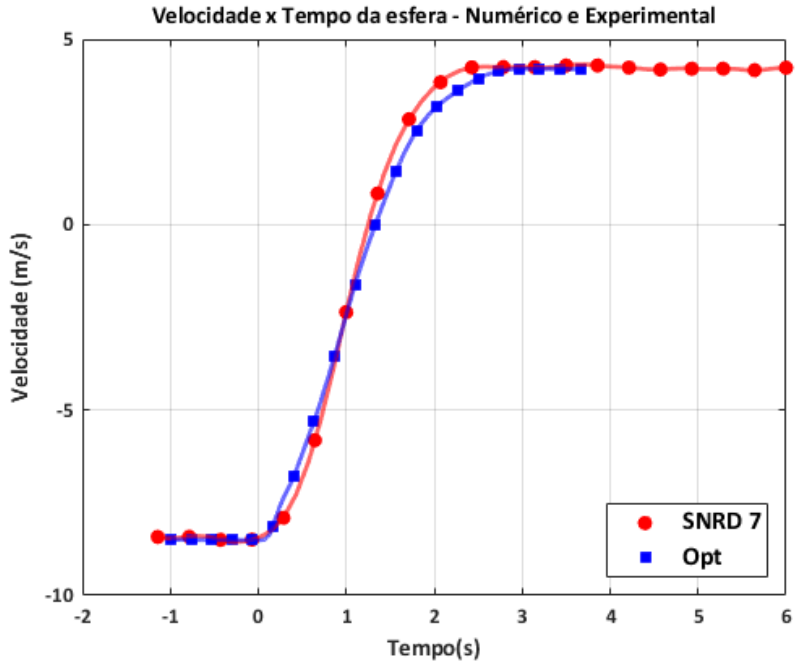
Fonte: autoria própria

Figura 76 – Comparação de valores obtidos numericamente e pelo método de SNDR



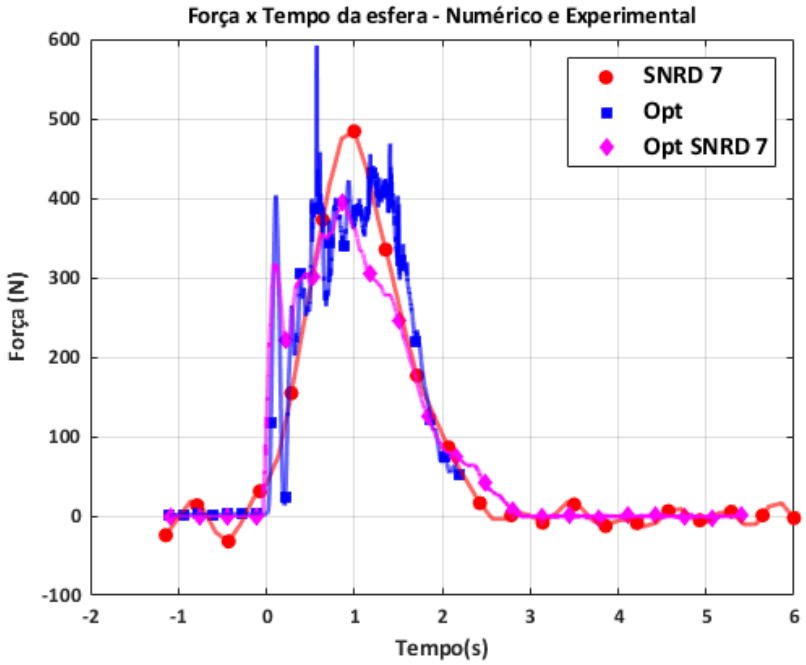
Fonte: autoria própria

Figura 77 – Detalhe sobre a comparação de velocidades entre a simulação numérica e o método SNDR com N = 7



Fonte: autoria própria

Figura 78 – Detalhe sobre a comparação de força transmitida entre a simulação numérica, força numérica filtrada com SNDR e o método SNDR com N = 7



Fonte: autoria própria

6 Discussão

6.1 Calibração EVA

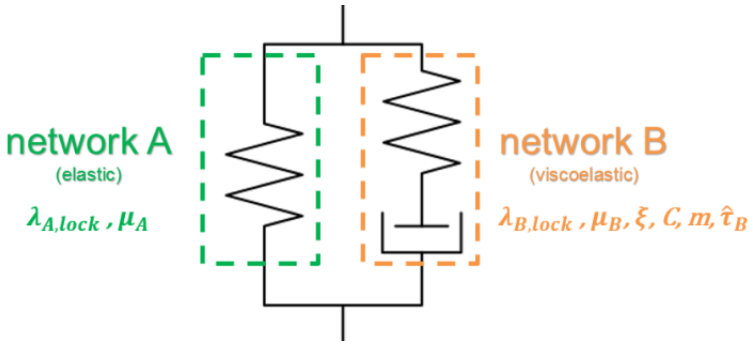
Primeiramente serão abordados nesta seção os resultados referentes à calibração do material flexível e rígido. É possível ver nas figuras 27 até 29 que os valores obtidos com as simulações numéricas foram muito coerentes com obtidos experimentalmente para o material flexível. A viscoelasticidade otimizada via modeFrontier permitiu que o modelo numérico se comportasse de maneira coerente em diferentes taxas de deformação. A parcela viscosa é a responsável por permitir ao modelo apresentar esses comportamentos diferentes, pois a força transmitida ao suporte será diferente dependendo da taxa de deformação imposta, uma vez que a dissipação de energia em cada caso será diferente. Este papel da viscosidade é feito pela parcela g_i do modelo de série de Prony, cuja soma de termos ficou próxima de 1.0, denotando um comportamento viscoelástico forte. Além disso, os termos k_i ficaram nulos, denotando uma característica de incompressibilidade do material, o que condiz com o valor do coeficiente de Poisson próximo do valor 0.5. Por fim, os tempos de relaxamento τ_i foram otimizados de acordo com a minimização da função objetivo. Valores mais baixos de tempos de relaxamento indicam uma espécie de dominância da parte viscosa sobre a parte elástica.

Em relação aos parâmetros do modelo de Arruda-Boyce, é possível perceber que a maior mudança em relação aos parâmetros iniciais foi em relação ao valor de λ_m. Quanto maior o valor encontrado para este parâmetro, maior será o tempo no qual a curva de hipereslasticidade apresentará comportamento exponencial. Pode-se notar este fenômeno mais fortemente em altas taxas de deformação, como o mostrado na figura 27. Além disso, o valor de μ_0 sofreu algumas alterações para se adaptar aos esforços de cisalhamento presentes no ensaio de compressão. Assim, os resultados para o material flexível foram considerados bem satisfatórios.

Para se comparar os resultados em uma mesma escala, os resultados numéricos e experimentais em diferentes taxas de deformação foram plotados juntos em um gráfico de força x tempo (figura 30) e força x deslocamento (figura 31). O principal fenômeno que pode-se observar nesses gráficos é a diminuição da força transmitida conforme a taxa de deformação diminui. Esse efeito atrelado à viscoelasticidade é bem visível na figura 30, tanto nos dados do ensaio quanto nos obtidos numericamente, e condiz com a teoria. Já no gráfico 31 é possível perceber também o mesmo fenômeno: quanto maior a taxa de deformação, maior será a a força obtida para um mesmo deslocamento. Os resultados numéricos, principalmente em baixos deslocamentos, apresentam comportamentos parecidos e destoam dos resultados experimentais. Esse fenômenos pode estar atrelado às simplificações adotadas nos modelos, pois foi utilizado um modelo hiperelástico do tipo Arruda-Boyce, que apesar de se apresentar como um modelo bem completo de polímeros elastômeros, ainda possui

suas restrições e dificuldades de representar o material real. Um possível modelo mais completo que poderia ser utilizado é o de Bergstrom-Boyce, que poderia auxiliar no ajuste fino do comportamento do material. Entretanto, este tipo de modelo não está presente na biblioteca de material e seria necessário a criação de uma subrotina (VUMAT, por exemplo) para implementar este modelo de material, o que foge do escopo deste projeto.

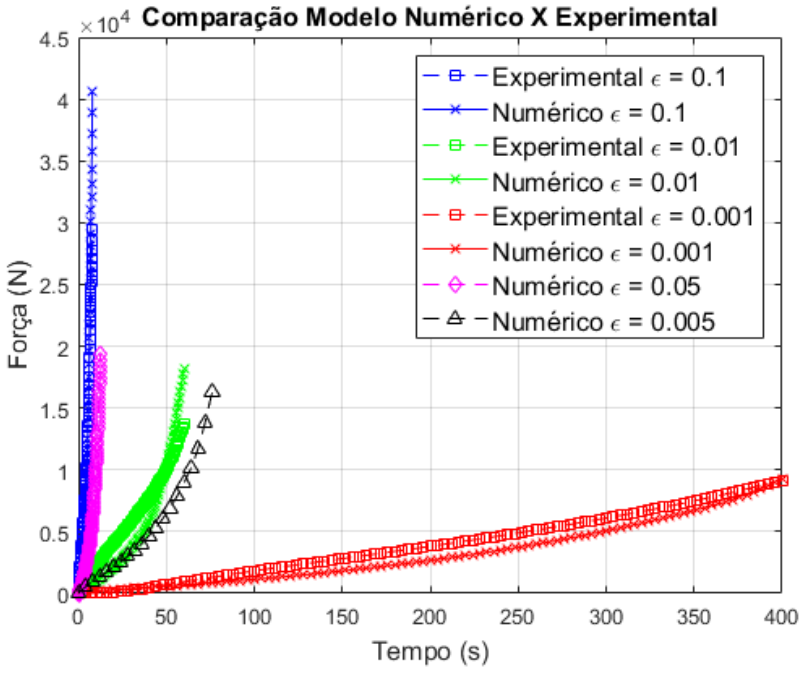
Figura 79 – Modelo reológico de Bergstrom-Boyce



Fonte: retirado de (SIKORA; MICHALCZYK; MACHNIEWICZ, 2018)

A fim de se confirmar a validação do modelo, dois ensaios extras foram realizados, com taxas de deformações intermediárias entre os ensaios realizados. Ou seja, uma taxa de deformação intermediária entre 10^{-1} e 10^{-2} (aproximadamente 5.10^{-2}) e uma entre 10^{-2} e 10^{-3} (aproximadamente 5.10^{-3}). Os resultados podem ser conferidos na imagem abaixo, e pode-se perceber que o padrão descrito acima se mantém: quanto maior a taxa de deformação, maiores serão as forças e menores os tempos de compressão.

Figura 80 – Resultados dos ensaios de compressão intermediários



Fonte: autoria própria

Já em relação ao material rígido, os resultados também foram satisfatórios, pois analogamente ao modelo flexível explicado acima, este apresentou resultados muito próximos aos ensaios experimentais. É possível perceber a similaridade dos resultados numéricos com os ensaios experimentais nas figuras 32 até 34. Como o material rígido apresenta apenas uma parcela elástica, a modelização foi mais precisa do que a parte flexível, devido à baixa presença de não-linearidades.

Com base nas análises, o material rígido apresenta uma característica de redistribuir a força aplicada sobre ele, de modo que quando o suporte superior de compressão tocar o material, leva um certo tempo para que a força seja distribuída na superfície do material (devido à não planicidade do corpo de prova) de modo que leva um tempo para que o suporte encoste realmente na superfície inferior do equipamento e passe a ser comprimido realmente. Esse fenômeno pode ser observado nas curvas experimentais 32 e 33 logo no começo do ensaio, onde o comportamento não é linear devido justamente ao fenômeno de distribuição de força.

Uma vez sendo comprimido, a curva das figuras 32 e 33 seguem lineares como previsto. O mais importante, é que tanto o valor numérico quanto o obtido experimentalmente possuem a mesma inclinação, seguindo paralelas umas às outras, denotando que apesar do comportamento anômalo no começo do ensaio, a simulação numérica é condizente com o fenômeno observado.

Além disso, nas figuras 35 e 36 é possível perceber que as curvas são muito coerentes, seguindo todas na mesma inclinação e com pouco espaço entre si. O efeito da taxa de deformação são menos perceptíveis do que no material flexível, pois o material rígido não apresenta nenhuma parcela de viscoelasticidade, o que corrobora com o esperado teoricamente.

Portanto, o material foi bem calibrado e os resultados estão consistentes com os obtidos experimentalmente. Com isso, foi possível seguir para os ensaios de impacto, a fim de se explicar mais consistentemente os resultados obtidos por (SANTOIEMMA, 2018)

6.2 Calibração do Osso e Pele

Os primeiros passos de caracterização do material utilizado para simular o Osso e a Pele foram bem sucedidos. Com relação à este primeiro, os valores encontrados para as propriedades do material levaram o ensaio de impacto a ser muito coerente com o experimental. As figuras 37 e 41 mostram que a posição e a velocidade da massa de impacto ao longo do tempo é muito consistente com o realizado em laboratório, em todas as abordagens (MDC, Laczos e SNDR).

Pode-se notar que nesse estudo a velocidade final da esfera de impacto se estabiliza em torno de 6.3 m/s, que é muito próximo da velocidade inicial da mesma. Isso significa

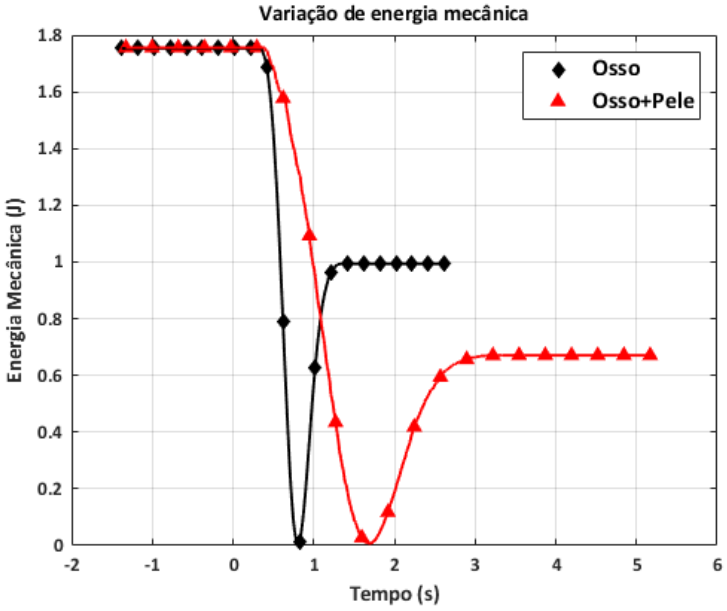
que a energia mecânica antes e após ao impacto é muito próxima, de modo que há pouca dissipação desta energia, o que leva a uma força resultante no osso extremamente elevada, como mostrado em 42, pois toda a energia do impacto é transmitido verticalmente e quase nada é dissipado no material.

A força obtida numericamente é bem consistente com a encontrada a partir dos dados experimentais, ocorrendo apenas pequenas variações entorno do valor de referência. Na verdade, é provável que ocorreram pequenas variações também no método experimental mas que foram imperceptíveis pelos sensores, de modo que a curva se apresenta mais suave, o que valida a otimização e a modelização feitas.

Por fim, este ensaio não se apresenta como uma situação real, pois nenhum impacto na região facial será diretamente com o osso, devendo no mínimo colidir com a pele primeiro. Dessa forma, é preciso analisar a otimização do osso com a pele. É possível perceber nas imagens 43, 47 e 48 que os dados novamente foram consistentes com os experimentos e portanto o modelo foi novamente validado. Além disso, neste caso, a velocidade se estabilizou por volta de 5.3 m/s, menor do que o impacto com apenas o Osso.

A conclusão que se pode tirar deste fenômeno é que a pele, devido à sua viscoelasticidade, funciona como um filtro passa baixas mecânico, de modo a "roubar"a energia mecânica que iria para o rosto. Dessa forma, a esfera fica mais tempo em contato com os corpos de prova, o que provoca uma diminuição do pico da força e um aumento do tempo de contato da esfera com os corpos de prova, conforme pode ser visto ao se comparar os gráficos 42 e 48. Além disso, é possível comparar a energia mecânica antes e depois do impacto, conforme mostra a imagem abaixo

Figura 81 – Comparação da energia mecânica antes e após o impacto com apenas o Osso e Osso + Pele



Fonte: autoria prórpia

É nítido a diferença entre as duas situações de impacto com as mesmas condições iniciais, de modo que a pele humana já serve como uma proteção natural contra impactos que poderiam prejudicar os ossos, já que a energia potencial gravitacional é muito pequena quando comparada com a energia cinética envolvida em situações de impacto. Entretanto, a energia restante ainda é considerável para prejudicar os ossos zigomáticos e nasais, de modo que a presença do protetor facial ainda é justificada.

6.3 Ensaios de Impacto

Uma vez feitas as simulações iniciais, foram realizadas as simulações com as camadas de material flexível e rígido e combinações entre elas. Primeiramente, a simulação com o material flexível foi feita e os resultados foram próximos dos obtidos experimentalmente. As figuras 49, 53 e 54 mostram as comparações e todas ficaram coerentes com as referências, em todas as abordagens adotadas. A velocidade final da esfera de impacto ficou entorno de 4.8 m/s, o que denota que o material flexível auxiliou ainda mais na dissipação de energia. A parte viscoelástico continua exercendo sua função, assim como a parcela modelizada pelo modelo de Arruda-Boyce, que absorve energia através do processo de atenuação que acontece em maior evidencia em impactos de alta frequência. Assim, o material flexível dissipa uma parte do impacto radialmente, funcionando como um "filtro passa baixas", atenuando os componentes de alta frequência da onda de deformação e não permitindo a passagem da totalidade da energia mecânica no sentido radial, o que auxilia na segurança quanto aos danos nos ossos faciais.

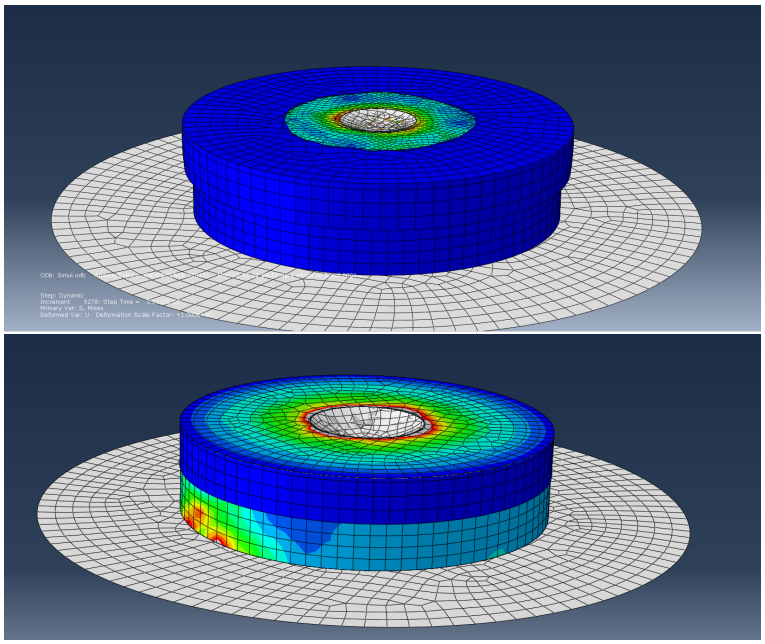
Um outro ponto que complementa este ponto é a força total transferida ao osso. Conforme é possível ver na figura 54, o pico de força é substancialmente menor do que nos casos anteriores, ficando em torno de 400 N, em troca de um maior tempo de impacto. É menos grave possuir um tempo de impacto maior mas uma força de pico menor do que o contrário, uma vez que é mais provável a presença de fraturas com picos elevados do que com forças menores mas mais duradouras.

Para o caso da simulação com o material rígido, é possível ver primeiramente que os resultados numéricos corroboram com os testes de impactos feitos em laboratório, de modo que a modelização também foi validada. A velocidade de estabilização da esfera mostrada em 59 ficou próxima de 5.2 m/s. Isso evidencia que o material rígido não é muito eficiente em termos de dissipação de energia, como o flexível. Entretanto, este tipo de material é muito eficiente no quesito de distribuir uniformemente a energia do impacto, o que pode ser evidenciado pelo gráfico de força 60.

Dessa forma, se justifica que o material rígido possua uma menor eficiência, tanto em dissipação de energia quanto em transmissão de força, que o material flexível, ficando próximo até de uma situação com a pele exposta. Esse fenômeno nada intuitivo deve-se ao fato da viscoelasticidade da pele não se comportar como o material de Arruda-

Boyce modelizado para o EVA flexível, ou seja, na presença de uma força distribuída (redirecionada pelo material rígido) a pele perde um pouco sua capacidade de dissipação de energia radialmente, pois o impacto não consegue se "espalhar"pela superfície da pele, uma vez que já se encontra redirecionado. Entretanto, redirecionar a força na presença do material modelizado com Arruda-Boyce facilitará a atenuação da energia mecânica devido à característica de elastômero desse modelo de material, aumentando a eficiência como mostrado nas simulações RF.

Figura 82 – Na primeira figura, simulação em corte com o material rígido, dissipando pouca energia. Na segunda figura, simulação em corte somente com a pele, mostrando uma grande dissipação. Ambas as figuras mostram as tensões de Von mises na superfície da Pele



Fonte: autoria própria

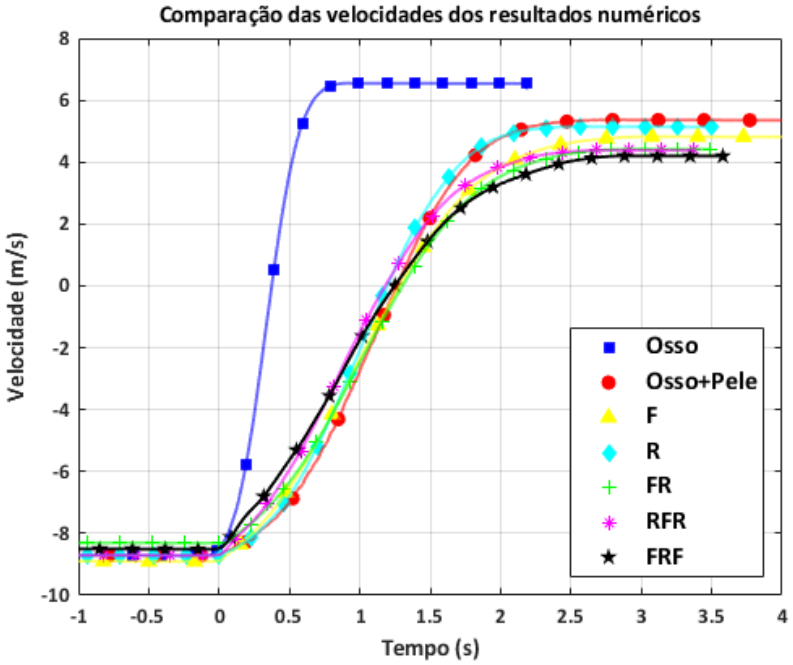
Dessa forma, deseja-se unir as duas características desses materiais a fim de se maximizar a energia dissipada em uma situação de impacto e proteger o máximo possível os ossos faciais. Dessa forma, foi realizada a simulação com uma camada de material flexível e uma camada de material rígido. Os resultados são mostrados nas figuras 61, 65 e 66. Neste caso, os resultados divergiram levemente dos valores encontrados experimentalmente, mas mesmo assim ficaram consistentes. Este fenômeno pode ter sido acarretado pela simplificação do contato entre os corpos de prova. Isso porque durante os ensaios experimentais, para se manter os corpos de provas em posição, foi utilizado um pedaço de fita adesiva em um dos lados dos corpos de prova, o que acaba contaminando um pouco os resultados do teste. Como essa interação não foi modelizada, a diferença entre simulação e ensaio é justificada.

De toda forma, para o ensaio RF em questão, a velocidade da esfera após o impacto

foi de aproximadamente 4.4 m/s e o pico da força transmitida foi de 450 N. É possível perceber que a combinação das propriedades dos tipos de EVA ajudou na amenização do impacto, uma vez que o material rígido faz o papel de distribuir mais uniformemente a força do impacto sobre toda a superfície do material flexível, o que melhora a capacidade deste último de absorver a energia mecânica.

Este conceito de unir as propriedades de um material rígido com um flexível foi extrapolado com as modelizações dos ensaios RFR e FRF, cujos os resultados podem ser conferidos nas figuras de 67 até 78. Pode-se perceber que houve uma melhora significativa na questão de absorção de energia, passando à uma velocidade final da esfera após o impacto de 4.1 m/s para o caso FRF. Para resumir em questão de velocidades antes e depois do impacto, para todas as simulações realizadas, que pode ser conferido abaixo.

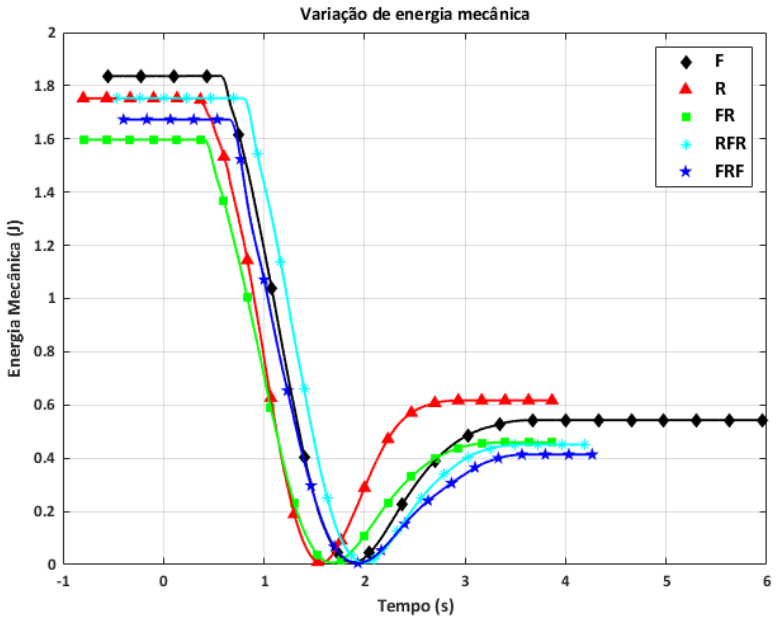
Figura 83 – Resultados de velocidade em todas as simulações realizadas



Fonte: autoria própria

É possível perceber mais facilmente agora a importância da presença do dispositivo de proteção facial em uma situação de impacto, dissipando a energia mecânica e prevenindo que o atleta lesionado se machuque ainda mais. Uma outra análise interessante a se fazer é a respeito das energias mecânicas antes e depois do impacto, que pode ser conferido no gráfico abaixo

Figura 84 – Energia Mecânica de cada uma das simulações com material de proteção

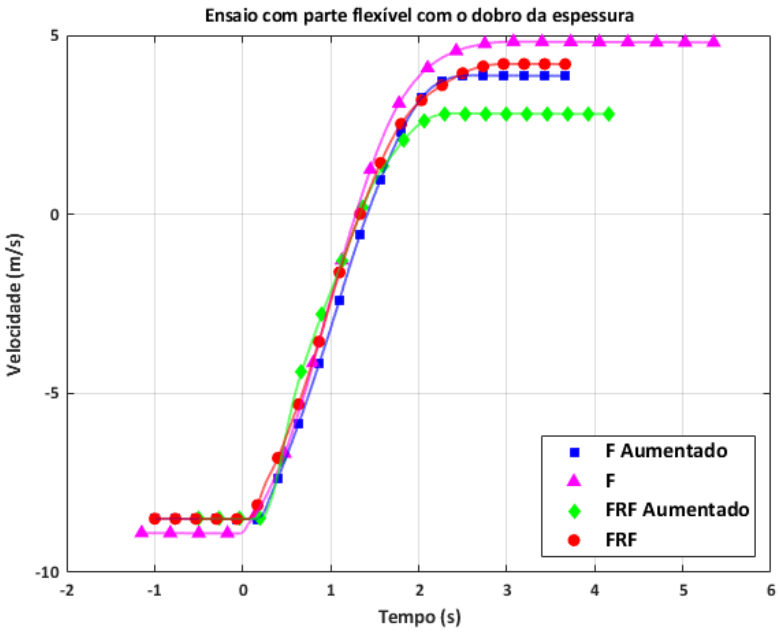


Fonte: autoria própria

Na figura 84 acima, os valores iniciais de energia mecânica variam pois cada ensaio teve uma energia cinética diferente. Devido aos fatores externos (como resistência do ar, trajetória da esfera, etc) ao ensaio de laboratório, nem sempre a massa de impacto chegava aos corpos de prova com a mesma velocidade. Desse modo, a fim de se aproximar o máximo possível da situação real feita experimentalmente, essas diferenças de velocidade inicial foram levadas em conta neste trabalho, o que acarreta em diferenças de energia mecânica antes do impacto. Entretanto, somente as variações de energia serão levadas em conta, de modo a estimar a quantidade de energia que foi absorvida pelo protetor no impacto.

Pode-se perceber a importância do dispositivo de proteção facial quando são comparadas a figuras 81 com 84 e percebe-se que a quantidade de energia absorvida foi maior nos casos em que os materiais de proteção facial estão presentes, notavelmente nas situações F e FRF. Uma grande vantagem de se realizar uma simulação numérica é a capacidade de explorar as possibilidades que normalmente não seriam possíveis nos ensaios experimentais. Por exemplo, não foi possível realizar nos estudos em laboratório simulações de impacto com a espessura da parte flexível aumentada. Entretanto, isto é possível de ser feito com uma simulação numérica, como mostra a figura abaixo, onde os ensaios F e FRF foram repetidos com os valores das espessuras das partes flexíveis com o dobro da espessura

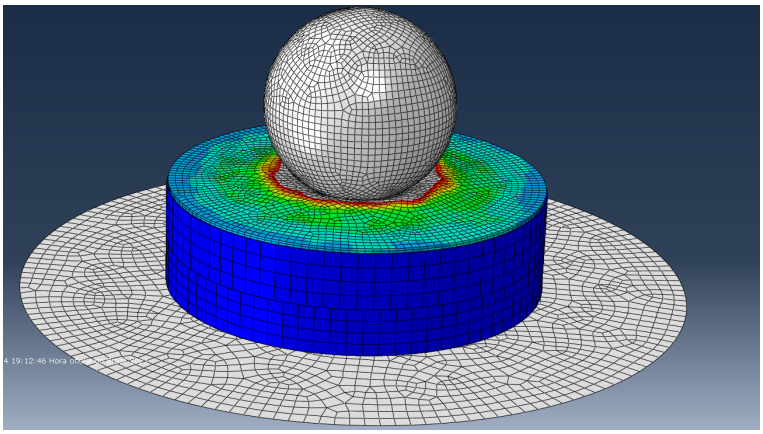
Figura 85 – Simulação de impacto realizada com o dobro da espessura



Fonte: autoria própria

É perceptível a melhora em questão de absorção de energia, com o modelo FRF modificado chegando a uma velocidade de retorno da massa de impacto de aproximadamente 2.8 m/s, o que representa uma redução muito significativa do modelo original com 4.1 m/s, ou seja, o material flexível mais espesso dissipa mais energia do que o rígido, devido à sua capacidade viscosa de dissipação radial. Analogamente, o material rígido transferirá toda a força do impacto diretamente para a camada abaixo dele, podendo provocar provocar prejuízos à saúde dos ossos faciais.

Figura 86 – Impacto com o material flexível - Dissipação radial de energia



Fonte: autoria própria

Assim, a partir dos estudos feitos, uma possível proposta de protetor facial seria com uma mistura de material flexível e rígido, otimizados agora quanto à sua espessura,

uma vez que um protetor espesso demais pode prejudicar a visão do atleta. Dessa forma, a segurança do atleta será preservada enquanto que sua performance não será prejudicada.

7 Conclusão

O presente trabalho rendeu resultados extremamente satisfatório para o objetivo apresentado, que consiste na realização de um modelo numérico que represente bem os experimentos realizados em laboratório. Todos os resultados estavam de acordo com o esperado. A calibração dos materiais foi bem sucedida no quesito de caracterização de um material que atende bem as exigências do projeto e também que se aproxima do comportamento do material usado em ensaios experimentais.

Todos os modelos respeitaram os aspectos teóricos esperados, em relação à viscoelasticidade. A maneira como o modelo foi feito também foi muito bem pensada para se facilitar a mudança de parâmetros se necessário. É possível portanto mudar os parâmetros de dimensão, material usado, tipo de análise e outros apenas mudando uma única linha de código, pois o restante dele é dependente desta linha geral de propriedades. Graças à essa construção, é muito simples realizar uma grande gama de possibilidades de otimização para se chegar em um material e geometria ótimos para os protetores faciais.

A metodologia aplicada foi considerada razoável para os objetivos deste relatório. O software ABAQUS se apresentou como uma ferramenta de grande potencial e capacidade de análise. Da mesma maneira, o modeFrontier se mostrou uma ferramenta excelente para problemas de otimização multiobjetivo. Graças a facilidade de construção dos DOEs e dos métodos de otimização, foi possível se avançar mais fundo no projeto e nas análises dos resultados.

7.1 Trabalhos Futuros

Quanto à modelização proposta, existem melhorias que podem ser feitas para aproximar ainda mais o modelo da realidade. Uma possível otimização seria a utilização de uma sub-rotina para o ABAQUS a fim de se implementar um modelo de material de Bergstrom-Boyce e simular os mesmos ensaios novamente, com o objetivo de se comparar os resultados. Além disso, pode-se utilizar o modeFrontier para otimizar as espessuras das camadas de material flexível e rígido, a fim de se melhorar ainda mais a absorção de energia e atenuação da força transmitida.

É preciso também se realizar ensaios de compressão com os materiais do Osso e da Pele utilizados nos ensaios experimentais. Neste trabalho, foram utilizados os ensaios de impacto para a caracterização destes dois materiais, o que não é aconselhável. Assim, para se garantir maior confiabilidade nos modelos de material, seria interessante a realização de ensaios cíclicos e de relaxamento nos materiais do Osso e da Pele.

Por fim, visa-se simular diversas geometrias de máscaras para se encontrar a geometria ideal de protetor facial em questão de absorção de energia mecânica. Dessa forma, deve-se testar diversos modelos de material em diversas geometrias diferentes de protetores, e comparar-se os resultados de força x tempo. Dessa forma, podem ser obtidos resultados mais próximos da realidade, que podem ser confirmados prototipando-se a melhor geometria de protetor com o melhor material e realizando-se ensaios experimentais de impacto.

Referências

ALMEIDA, N. H. d. *Reconstrução facial: mensuração da espessura dos tecidos moles que recobrem a face*. Tese (Doutorado) — Universidade de São Paulo, 2012. Citado na página 28.

BARRA, G. *APOSTILA DE PROCESSOS 4: PARTE 1: FUNDAMENTOS DE REOLOGIA DE MATERIAIS POLIMÉRICOS*. [S.l.]: Curso de EMC 5744. Citado na página 51.

BLACK, A. M. et al. Epidemiology of facial injuries in sport. *Clinics in sports medicine*, Elsevier, v. 36, n. 2, p. 237–255, 2017. Citado na página 28.

BRASIL, M. d. S. Lesões por esforços repetitivos. 2009. Citado na página 27.

BRAUNSTEIN, P. W. Medical aspects of automotive crash injury research. *Journal of the American Medical Association*, American Medical Association, v. 163, n. 4, p. 249–255, 1957. Citado 2 vezes nas páginas 33 e 34.

BUCHOLZ, R. W. et al. Occult cervical spine injuries in fatal traffic accidents. *Journal of Trauma and Acute Care Surgery*, LWW, v. 19, n. 10, p. 768–771, 1979. Citado na página 27.

CALLISTER, W. *Ciência E Engenharia de Materiais: Uma Introdução* . [S.l.]: Grupo Gen-LTC, 2000. Citado 3 vezes nas páginas 38, 39 e 40.

CARROLL, S. M. et al. One hundred and ten sports related facial fractures. *British journal of sports medicine*, British Association of Sport and Excercise Medicine, v. 29, n. 3, p. 194–195, 1995. Citado na página 34.

COLEMAN, J. N. et al. Small but strong: a review of the mechanical properties of carbon nanotube–polymer composites. *Carbon*, Elsevier, v. 44, n. 9, p. 1624–1652, 2006. Citado na página 48.

COSTA, L. Pires da. Metodologia de análise e otimização de absorvedores deenergia em automóveis. 2019. Citado na página 44.

COTO, N. P. *Avaliação de protetor nasal para atividades desportivas: análise por elementos finitos*. Tese (Doutorado) — Universidade de São Paulo, 2009. Citado 2 vezes nas páginas 29 e 30.

DELILBASI, C. et al. Maxillofacial fractures sustained during sports played with a ball. *Oral Surgery, Oral Medicine, Oral Pathology, Oral Radiology, and Endodontology*, Elsevier, v. 97, n. 1, p. 23–27, 2004. Citado na página 28.

DIAS, R. Brito e. *FOUSP na midia: Protetores faciais para esporte*. Disponível em: <http://www.fo.usp.br/?p=28959>. Citado na página 30.

DYER, P. V. Experimental study of fractures of the upper jaw: a critique of the original papers published by rené le fort. *Trauma*, Sage Publications Sage CA: Thousand Oaks, CA, v. 1, n. 1, p. 81–84, 1999. Citado na página 28.

FORT, R. L. Étude expérimentale sur les fractures de la machoire supérieure. *Rev. Chirurgique de Paris*, Medscape, v. 1, 2 and 3, 1901. Citado na página 35.

FROMMER, S. A.; BHATT, R. A. Sports-related facial trauma. *Plataforma online de saúde Medscape*, Medscape, p. 1–16, 2016. Citado na página 28.

FUNK, G. *Facial Fracture Management Handbook - LeFort Fractures*. Disponível em: <https://medicine.uiowa.edu/iowaprotocols/facial-fracture-management-handbook-lefort-fractures>. Citado 2 vezes nas páginas 35 e 36.

GASSNER, R. et al. Craniomaxillofacial trauma in children: a review of 3,385 cases with 6,060 injuries in 10 years. *Journal of oral and maxillofacial surgery*, Elsevier, v. 62, n. 4, p. 399–407, 2004. Citado na página 27.

GHOREISHY, M. H. R. Determination of the parameters of the prony series in hyper-viscoelastic material models using the finite element method. *Materials & Design*, Elsevier, v. 35, p. 791–797, 2012. Citado na página 52.

JONES, D. The intermaxillary screw: a dedicated bicortical bone screw for temporary intermaxillary fixation. *British journal of oral & maxillofacial surgery*, v. 37, n. 2, p. 115–116, 1999. Citado na página 29.

KALPAKJIAN, S. *Manufacturing processes for engineering materials*. [S.l.]: Pearson Education India, 1984. Citado na página 39.

KIRK, D. et al. Tight placement of erich arch bar while avoiding wire fatigue failure. *Journal of Oral and Maxillofacial Surgery*, Elsevier, v. 74, n. 3, p. 562–568, 2016. Citado na página 29.

MACMILLAN, A. et al. How do le fort–type fractures present in a pediatric cohort? *Journal of Oral and Maxillofacial Surgery*, Elsevier, v. 76, n. 5, p. 1044–1054, 2018. Citado 2 vezes nas páginas 34 e 36.

MALADIÈRE, E. et al. Aetiology and incidence of facial fractures sustained during sports: a prospective study of 140 patients. *International journal of oral and maxillofacial surgery*, Elsevier, v. 30, n. 4, p. 291–295, 2001. Citado na página 28.

MATSUNAGA, S. et al. Biomechanics of jaw bone considering structural properties of trabecular bone. *Journal of Oral Biosciences*, Elsevier, v. 53, n. 2, p. 143–147, 2011. Citado 2 vezes nas páginas 40 e 41.

MONTOVANI, J. C. et al. Etiologia e incidência das fraturas faciais em adultos e crianças: experiência em 513 casos. *Revista brasileira de otorrinolaringologia*, SciELO Brasil, v. 72, n. 2, p. 235–241, 2006. Citado na página 33.

MOREIRA, P.; GENTIL, D.; OLIVEIRA, C. de. Prevalência de lesões na temporada 2002 da seleção brasileira masculina de basquete. *Clínica*, SciELO Brasil, v. 22, p. 22, 2003. Citado na página 28.

MOURA, R. T. *Modelagem e ensaios mecânicos de polímeros termoplásticos sob carregamentos quase-estático e dinâmico*. Tese (Doutorado) — Universidade de São Paulo, 2013. Citado 3 vezes nas páginas 33, 37 e 47.

MOUROUZIS, C.; KOUMOURA, F. Sports-related maxillofacial fractures: a retrospective study of 125 patients. *International journal of oral and maxillofacial surgery*, Elsevier, v. 34, n. 6, p. 635–638, 2005. Citado 2 vezes nas páginas 28 e 34.

OMS. *RELATÓRIO MUNDIAL SOBRE PREVENÇÃO DE LESÕES CAUSADAS PELO TRÂNSITO: RESUMO*. 2012. Citado na página 27.

PICCININNI, P. et al. Dental and orofacial injuries. *Clinics in sports medicine*, Elsevier, v. 36, n. 2, p. 369–405, 2017. Citado 2 vezes nas páginas 34 e 36.

PRAZERES, L. *Ossos do corpo humano – Onde se localizam?* Disponível em: <https://enfermagemcomamor.com.br/index.php/2018/03/27/ossos-do-corpo-humano-onde-se-localizam/>. Citado na página 33.

QING, Y. et al. Microwave-absorbing and mechanical properties of carbonyl-iron/epoxy-silicone resin coatings. *Journal of Magnetism and Magnetic Materials*, Elsevier, v. 321, n. 1, p. 25–28, 2009. Citado na página 30.

QURESHI, A. A. et al. Intermaxillary fixation screws versus erich arch bars in mandibular fractures: A comparative study and review of literature. *Annals of maxillofacial surgery*, Wolters Kluwer–Medknow Publications, v. 6, n. 1, p. 25, 2016. Citado na página 28.

ROLAND, S. *Apostila Introduction aux matériaux polymères thermoplastiques*. [S.l.]: Grupo Gen-LTC, 2018–2019. Citado 3 vezes nas páginas 48, 49 e 50.

SANTOIEMMA, B. P. Impacto em protetores faciais. 2018. Citado 7 vezes nas páginas 31, 53, 54, 55, 59, 62 e 93.

SHAH, A. R.; ROURE, R. M. Le fort fractures imaging. *Plataforma online de saúde Medscape*, Medscape, p. 1–12, 2019. Citado 2 vezes nas páginas 34 e 36.

SIKORA, W.; MICHALCZYK, K.; MACHNIEWICZ, T. Numerical modelling of metal-elastomer spring nonlinear response for low-rate deformations. *acta mechanica et automatica*, v. 12, n. 1, 2018. Citado na página 92.

SILVA, E. C. N. *Apostila Otimização aplicada ao projeto de sistemas mecânicos*. [S.l.]: Curso de PMR 5215, 2010. Citado 2 vezes nas páginas 44 e 45.

SILVA, J. J. d. L. et al. Trauma facial: análise de 194 casos. *Revista Brasileira de cirurgia plástica*, SciELO Brasil, v. 26, n. 1, p. 37–41, 2011. Citado 2 vezes nas páginas 27 e 28.

STEIN, M. Large sample properties of simulations using latin hypercube sampling. *Technometrics*, Taylor & Francis, v. 29, n. 2, p. 143–151, 1987. Citado na página 58.

TIMOSHENKO, S. P. *Mecânica dos sólidos*. [S.l.]: Ltc, 1984. Citado na página 37.

UFES, P. P. M. *Elementos Finitos*. Disponível em: <https://petmecanica.ufes.br/elementos-finitos>. Citado na página 42.

WULKAN, M.; JR, J. G. P.; BOTTER, D. A. Epidemiologia do trauma facial. *Revista da associação médica brasileira*, SciELO Brasil, v. 51, n. 5, p. 290–295, 2005. Citado na página 27.

Apêndices

APÊNDICE A – Código Python da Modelização

Code Listing A.1 – Insert code directly in your document

```
# -*- coding: mbcs -*-
from abaqus import *
from abaqusConstants import *
from caeModules import *

'''
TESTE - ENSAIO DE TRACAO E COMPRESSAO

# Autor: Victor Siqueira Chaim (N USP: 9345467)
# Data de inicio:  10/09/2020
# Data de finalizacao: 14/07/2021
# Arquivo destinado ao projeto de TCC para obtencao do titulo de
                                    Engenheiro Mecatronico
# Universidade: Escola Politecnica da Universidade de Sao Paulo


'''

Mdb() #apaga o modelo anterior e cria um novo

'''
Escolha do tipo de corpo de prova a ser utilizado
Mod = 1 => Corpo de prova para ensaio de tracao
Mod = 2 => Corpo de prova para ensaio de compressao
Mod = 3 => Teste de Impacto
'''

Mod=3


'''
# DADOS EXPERIMENTAIS
Escolha do tipo de corpo de prova a ser utilizado => SOMENTE PARA MOD ==
                                        2!!
taxa_def = 1 => Corpo de prova sujeito a uma taxa de deformacao de 10^-1
taxa_def = 2 => Corpo de prova sujeito a uma taxa de deformacao de 10^-2
taxa_def = 3 => Corpo de prova sujeito a uma taxa de deformacao de 10^-3
```

```
'''
taxa_def = 1

'''
# MATERIAL UTILIZADO
Escolha do tipo de corpo de prova a ser utilizado
MAT = 1 => EVA Flexivel
MAT = 2 => EVA Rigido
'''
MAT = 1

'''
# CAMADAS A SEREM USADAS NO TESTE DE IMPACTO
Escolha quantas camadas e de quais tipos devem ser utilizadas (so faz
                                    sentido com Mod = 3!!!)
Camadas = 'Osso' => Somente impacto no osso
Camadas = 'sEVA' => sem camada de EVA (Pele + Osso)
Camadas = 'F' => Uma camada de material flexivel
Camadas = 'R' => Uma camada de material rigido
Camadas = 'FR' => Uma camada de material rigido e uma de flexivel
Camadas = 'RFR' => Uma camada de material rigido, flexivel e rigido
Camadas = 'FRF' => Uma camada de material flexivel, rigido e flexivel
'''
Camadas = 'F'



#Criacao do vetor de forca e tempo a partir da nossa referencia
Force_Ref = []
TimeF_Ref = []
#Criacao do vetor de deslocamento radial e tempo a partir da nossa
                                    referencia
Disp_Ref = []
TimeD_Ref = []

# Dados dos ensaios de compressao experimentais
if taxa_def == 1:
  if MAT == 1:
    Disp_Ref =  [0.0300,  0.0301,  0.0302,  0.0303,  0.0304,  0.0305,  0
                                    .0306,  0.0307,  0.0309,  0.0310,  0
                                    .0311,  0.0313,  0.0314,  0.0316,  0
                                    .0317,  0.0319,  0.0320,  0.0322,  0
                                    .0324,  0.0326,  0.0328,  0.0330,  0
                                    .0332,  0.0334,  0.0336,  0.0338,  0
                                    .0340,  0.0342,  0.0345,  0.0347,  0
                                    .0350,  0.0352,  0.0355,  0.0357,  0
```

```
                                .0360,  0.0363,  0.0366,  0.0369,  0
                                .0372,  0.0375,  0.0378,  0.0381,  0
                                .0384,  0.0388,  0.0391,  0.0395,  0
                                .0398,  0.0402,  0.0405,  0.0409,  0
                                .0413,  0.0417,  0.0421,  0.0425,  0
                                .0429,  0.0433,  0.0438,  0.0442,  0
                                .0447,  0.0451,  0.0456,  0.0461,  0
                                .0465,  0.0470,  0.0475,  0.0480,  0
                                .0485,  0.0491,  0.0496,  0.0501]
TimeD_Ref = [0,0.104,0.204,0.304,0.404,0.504,0.604,0.704,0.804,0.904
                                ,1.004,1.104,1.204,1.304,1.404,1.504
                                ,1.604,1.704,1.804,1.904,2.004,2.104
                                ,2.204,2.304,2.404,2.504,2.604,2.704
                                ,2.804,2.904,3.004,3.104,3.204,3.304
                                ,3.404,3.504,3.604,3.704,3.804,3.904
                                ,4.004,4.104,4.204,4.304,4.404,4.504
                                ,4.604,4.704,4.804,4.904,5.004,5.104
                                ,5.204,5.304,5.404,5.504,5.604,5.704
                                ,5.804,5.904,6.004,6.104,6.204,6.304
                                ,6.404,6.504,6.604,6.704,6.804,6.904
                                ,7.004,7.104,7.204,7.304,7.404,7.504
                                ,7.604,7.704,7.804,7.904,8.004,8.104
                                ,8.204,8.304,8.404,8.504]
TimeF_Ref = [0,0.104,0.204,0.304,0.404,0.504,0.604,0.704,0.804,0.904
                                ,1.004,1.104,1.204,1.304,1.404,1.504
                                ,1.604,1.704,1.804,1.904,2.004,2.104
                                ,2.204,2.304,2.404,2.504,2.604,2.704
                                ,2.804,2.904,3.004,3.104,3.204,3.304
                                ,3.404,3.504,3.604,3.704,3.804,3.904
                                ,4.004,4.104,4.204,4.304,4.404,4.504
                                ,4.604,4.704,4.804,4.904,5.004,5.104
                                ,5.204,5.304,5.404,5.504,5.604,5.704
                                ,5.804,5.904,6.004,6.104,6.204,6.304
                                ,6.404,6.504,6.604,6.704,6.804,6.904
                                ,7.004,7.104,7.204,7.304,7.404,7.504
                                ,7.604,7.704,7.804,7.904,8.004,8.104
                                ,8.204,8.304,8.404,8.504]
Force_Ref = [6.496,187.7062,499.86928,768.40608,1029.56872,1279.
                                96327,1521.83659,1756.36485,1981.
                                16191,2197.94456,2406.33823,2607.
                                65161,2804.23257,2996.91148,3186.
                                85099,3373.44185,3558.81937,3743.
                                18846,3928.32346,4113.40185,4299.
                                68163,4487.27161,4677.04497,4868.
                                76555,5062.90123,5259.47027,5459.
                                16855,5660.25078,5863.62705,6070.
                                46299,6278.37107,6490.09943,6704.
```

```
                                    35354,6919.81092,7137.19502,7358.
                                    84458,7581.51114,7810.12774,8041.
                                    34533,8276.36495,8518.81206,8768.
                                    22099,9024.59621,9288.83851,9560.
                                    16555,9842.50382,10136.79653,10440.
                                    07987,10760.55691,11095.59834,11445.
                                    53646,11815.41085,12205.26546,12620.
                                    51165,13062.47115,13527.15045,14016.
                                    90543,14531.92234,15074.09811,15647.
                                    80623,16256.70642,16897.03614,17578.
                                    09967,18291.87423,19030.35939,19797.
                                    56206,20549.39866,21259.10223,21919.
                                    07912,22551.06419,23157.5951,23731.
                                    07225,24246.43338,24753.40217,25290.
                                    86769,25837.40056,26441.90788,27053.
                                    2161,27752.91502,28561.96165,29394.
                                    37628,30209.82742,31006.90544,31873.
                                    77751,32875.85974,34015.21146]
    Velocidade_Compressao = 0.00045

if MAT == 2:
    Disp_Ref = [0.0300,  0.0301,  0.0302,  0.0303,  0.0304,  0.0305,  0.
                                    0306,  0.0307,  0.0309,  0.0310,  0.
                                    0311,  0.0313,  0.0314,  0.0316,  0.
                                    0317,  0.0319,  0.0320,  0.0322,  0.
                                    0324,  0.0326,  0.0328,  0.0330,  0.
                                    0332,  0.0334,  0.0336,  0.0338,  0.
                                    0340,  0.0342,  0.0345,  0.0347,  0.
                                    0350,  0.0352,  0.0355,  0.0357,  0.
                                    0360,  0.0363,  0.0366,  0.0369,  0.
                                    0372,  0.0375,  0.0378,  0.0381,  0.
                                    0384,  0.0388,  0.0391,  0.0395,  0.
                                    0398,  0.0402,  0.0405,  0.0409,  0.
                                    0413,  0.0417,  0.0421,  0.0425,  0.
                                    0429,  0.0433,  0.0438,  0.0442,  0.
                                    0447,  0.0451,  0.0456,  0.0461,  0.
                                    0465,  0.0470,  0.0475,  0.0480,  0.
                                    0485,  0.0491,  0.0496,  0.0501]
    TimeD_Ref = [0,   0.10,   0.20,   0.30,   0.40,   0.50,   0.60,   0.
                                    70,   0.80,   0.90,   1.00,   1.10,
                                      1.20,   1.30,   1.40,   1.50,   1.
                                    60,   1.70,   1.80,   1.90,   2.00,
                                      2.10,   2.20,   2.30,   2.40,   2.
                                    50,   2.60,   2.70,   2.80,   2.90,
                                      3.00,   3.10,   3.20,   3.30,   3.
                                    40,   3.50,   3.60,   3.70,   3.80,
                                      3.90,   4.00,   4.10,   4.20,   4.
```

```
                                    30,   4.40,   4.50,   4.60,   4.70,
                                      4.80]
    TimeF_Ref = [0, 0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9,
                                      1.0,  1.1,  1.2,  1.3,  1.4,  1.5]
    Force_Ref = [3810.02203,  4878.92702, 6857.35419, 9315.49594, 12001.
                                    72454,  14789.84803,  17631.33556,
                                    20509.19682,  23379.86082,  26251.
                                    27733,  29106.02987,  31937.17897,
                                    34726.60482,  37474.72465,  40169.
                                    35229,  42774.72496]
    Velocidade_Compressao = 0.00055

if taxa_def == 2:
  if MAT == 1:
    Disp_Ref =  [0.0300,  0.0300,  0.0300,  0.0300,  0.0300,  0.0300,  0
                                    .0300,  0.0300,  0.0300,  0.0300,  0
                                    .0300,  0.0300,  0.0300,  0.0301,  0
                                    .0301,  0.0301,  0.0301,  0.0301,  0
                                    .0302,  0.0302,  0.0302,  0.0302,  0
                                    .0303,  0.0303,  0.0303,  0.0304,  0
                                    .0304,  0.0305,  0.0305,  0.0306,  0
                                    .0306,  0.0306,  0.0307,  0.0308,  0
                                    .0308,  0.0309,  0.0309,  0.0310,  0
                                    .0310,  0.0311,  0.0312,  0.0312,  0
                                    .0313,  0.0314,  0.0314,  0.0315,  0
                                    .0316,  0.0317,  0.0318,  0.0318,  0
                                    .0319,  0.0320,  0.0321,  0.0322,  0
                                    .0323,  0.0323,  0.0324,  0.0325,  0
                                    .0326,  0.0327,  0.0328,  0.0329,  0
                                    .0330,  0.0331,  0.0332,  0.0333,  0
                                    .0334,  0.0335,  0.0336,  0.0337,  0
                                    .0338,  0.0339,  0.0341,  0.0342,  0
                                    .0343,  0.0344,  0.0345,  0.0346,  0
                                    .0347,  0.0349,  0.0350,  0.0351,  0
                                    .0352,  0.0354,  0.0355,  0.0356,  0
                                    .0357,  0.0359,  0.0360,  0.0361,  0
                                    .0362,  0.0364,  0.0365,  0.0366,  0
                                    .0368,  0.0369,  0.0370,  0.0372,  0
                                    .0373,  0.0374,  0.0376,  0.0377,  0
                                    .0379,  0.0380,  0.0381,  0.0383,  0
                                    .0384,  0.0386,  0.0387,  0.0389,  0
                                    .0390,  0.0391,  0.0393,  0.0394,  0
                                    .0396,  0.0397,  0.0399,  0.0400,  0
                                    .0402,  0.0403,  0.0405,  0.0406,  0
                                    .0408,  0.0409,  0.0411,  0.0412,  0
                                    .0414,  0.0415,  0.0417,  0.0418,  0
                                    .0420,  0.0421,  0.0423,  0.0424,  0
```

```
                                    .0426,  0.0427,  0.0429,  0.0431]
    TimeD_Ref = [0.0000,  0.5000,  1.0000,  1.5000,  2.0000,  2.5000,  3
                                    .0000,  3.5000,  4.0000,  4.5000,  5
                                    .0000,  5.5000,  6.0000,  6.5000,  7
                                    .0000,  7.5000,  8.0000,  8.5000,  9
                                    .0000,  9.5000, 10.0000, 10.5000, 11
                                    .0000, 11.5000, 12.0000, 12.5000, 13
                                    .0000, 13.5000, 14.0000, 14.5000, 15
                                    .0000, 15.5000, 16.0000, 16.5000, 17
                                    .0000, 17.5000, 18.0000, 18.5000, 19
                                    .0000, 19.5000, 20.0000, 20.5000, 21
                                    .0000, 21.5000, 22.0000, 22.5000, 23
                                    .0000, 23.5000, 24.0000, 24.5000, 25
                                    .0000, 25.5000, 26.0000, 26.5000, 27
                                    .0000, 27.5000, 28.0000, 28.5000, 29
                                    .0000, 29.5000, 30.0000, 30.5000, 31
                                    .0000, 31.5000, 32.0000, 32.5000, 33
                                    .0000, 33.5000, 34.0000, 34.5000, 35
                                    .0000, 35.5000, 36.0000, 36.5000, 37
                                    .0000, 37.5000, 38.0000, 38.5000, 39
                                    .0000, 39.5000, 40.0000, 40.5000, 41
                                    .0000, 41.5000, 42.0000, 42.5000, 43
                                    .0000, 43.5000, 44.0000, 44.5000, 45
                                    .0000, 45.5000, 46.0000, 46.5000, 47
                                    .0000, 47.5000, 48.0000, 48.5000, 49
                                    .0000, 49.5000, 50.0000, 50.5000, 51
                                    .0000, 51.5000, 52.0000, 52.5000, 53
                                    .0000, 53.5000, 54.0000, 54.5000, 55
                                    .0000, 55.5000, 56.0000, 56.5000, 57
                                    .0000, 57.5000, 58.0000, 58.5000, 59
                                    .0000, 59.5000, 60.0000, 60.5000, 61
                                    .0000, 61.5000, 62.0000, 62.5000, 63
                                    .0000, 63.5000, 64.0000, 64.5000, 65
                                    .0000, 65.5000, 66.0000, 66.5000, 67
                                    .0000, 67.5000, 68.0000, 68.5000]
    TimeF_Ref = [0.0000,  0.5000,  1.0000,  1.5000,  2.0000,  2.5000,  3
                                    .0000,  3.5000,  4.0000,  4.5000,  5
                                    .0000,  5.5000,  6.0000,  6.5000,  7
                                    .0000,  7.5000,  8.0000,  8.5000,  9
                                    .0000,  9.5000, 10.0000, 10.5000, 11
                                    .0000, 11.5000, 12.0000, 12.5000, 13
                                    .0000, 13.5000, 14.0000, 14.5000, 15
                                    .0000, 15.5000, 16.0000, 16.5000, 17
                                    .0000, 17.5000, 18.0000, 18.5000, 19
                                    .0000, 19.5000, 20.0000, 20.5000, 21
                                    .0000, 21.5000, 22.0000, 22.5000, 23
                                    .0000, 23.5000, 24.0000, 24.5000, 25
```

```
                                    .0000, 25.5000, 26.0000, 26.5000, 27
                                    .0000, 27.5000, 28.0000, 28.5000, 29
                                    .0000, 29.5000, 30.0000, 30.5000, 31
                                    .0000, 31.5000, 32.0000, 32.5000, 33
                                    .0000, 33.5000, 34.0000, 34.5000, 35
                                    .0000, 35.5000, 36.0000, 36.5000, 37
                                    .0000, 37.5000, 38.0000, 38.5000, 39
                                    .0000, 39.5000, 40.0000, 40.5000, 41
                                    .0000, 41.5000, 42.0000, 42.5000, 43
                                    .0000, 43.5000, 44.0000, 44.5000, 45
                                    .0000, 45.5000, 46.0000, 46.5000, 47
                                    .0000, 47.5000, 48.0000, 48.5000, 49
                                    .0000, 49.5000, 50.0000, 50.5000, 51
                                    .0000, 51.5000, 52.0000, 52.5000, 53
                                    .0000, 53.5000, 54.0000, 54.5000, 55
                                    .0000, 55.5000, 56.0000, 56.5000, 57
                                    .0000, 57.5000, 58.0000, 58.5000, 59
                                    .0000, 59.5000, 60.0000, 60.5000, 61
                                    .0000, 61.5000, 62.0000, 62.5000, 63
                                    .0000, 63.5000, 64.0000, 64.5000, 65
                                    .0000, 65.5000, 66.0000, 66.5000, 67
                                    .0000, 67.5000, 68.0000, 68.5000]
Force_Ref = [0.1296, 49.2527,121.1517,211.2313,310.8289,421.8663,540
                                    .8468,663.9882,788.5486,916.9311,
                                    1044.1379,1170.0112,1293.1821,1415.
                                    3152,1536.3389,1654.8490,1770.7275,
                                    1882.7349,1994.3094,2104.1658,2212.
                                    3296,2317.4811,2420.6845,2523.3904,
                                    2624.3649,2723.7073,2819.1235,2915.
                                    8072,3010.0990,3104.5513,3195.8912,
                                    3284.8649,3376.0503,3465.4472,3554.
                                    5718,3640.5377,3727.3534,3813.5592,
                                    3900.5637,3984.7430,4068.6022,4154.
                                    0112,4240.1820,4325.5642,4408.3636,
                                    4494.1395,4579.6540,4667.3331,4752.
                                    4668,4837.0909,4924.4121,5012.0253,
                                    5100.9767,5187.8698,5275.4089,5365.
                                    6831,5456.4953,5547.2869,5635.8039,
                                    5728.2575,5821.0224,5914.2143,6004.
                                    9117,6097.9873,6191.8218,6287.7156,
                                    6382.5443,6475.9880,6571.4605,6668.
                                    1042,6767.1351,6862.5405,6959.3571,
                                    7058.1220,7158.8464,7258.7028,7353.
                                    9704,7454.5540,7556.3669,7660.6616,
                                    7761.8174,7862.4360,7966.8537,8070.
                                    8727,8177.8750,8278.7193,8383.8150,
                                    8492.1859,8604.7038,8714.6997,8823.
```

```
                                    8135,8939.6767,9057.4481,9177.4017,
                                    9293.5145,9415.3337,9540.7523,9672.
                                    2618,9802.0859,9929.3992,10065.8789,
                                    10204.2019,10350.5574,10491.6863,
                                    10639.4030,10796.4471,10960.2772,
                                    11125.0438,11286.1641,11460.6768,
                                    11639.5257,11825.5504,12006.5048,
                                    12197.6867,12399.5714,12610.5860,
                                    12823.5117,13036.2183,13259.9816,
                                    13490.4772,13734.8101,13972.9202,
                                    14218.7715]
    Velocidade_Compressao = 9.e-5

if MAT == 2:
    Disp_Ref =  [0.0300,  0.0300,  0.0300,  0.0300,  0.0300,  0.0300,  0
                                    .0300,  0.0300,  0.0300,  0.0300,  0
                                    .0300,  0.0300,  0.0300,  0.0301,  0
                                    .0301,  0.0301,  0.0301,  0.0301,  0
                                    .0302,  0.0302,  0.0302,  0.0302,  0
                                    .0303,  0.0303,  0.0303,  0.0304,  0
                                    .0304,  0.0305,  0.0305,  0.0306,  0
                                    .0306,  0.0306,  0.0307,  0.0308,  0
                                    .0308,  0.0309,  0.0309,  0.0310,  0
                                    .0310,  0.0311,  0.0312,  0.0312,  0
                                    .0313,  0.0314,  0.0314,  0.0315,  0
                                    .0316,  0.0317,  0.0318,  0.0318,  0
                                    .0319,  0.0320,  0.0321,  0.0322,  0
                                    .0323,  0.0323,  0.0324,  0.0325,  0
                                    .0326,  0.0327,  0.0328,  0.0329,  0
                                    .0330,  0.0331,  0.0332,  0.0333,  0
                                    .0334,  0.0335,  0.0336,  0.0337,  0
                                    .0338,  0.0339,  0.0341,  0.0342,  0
                                    .0343,  0.0344,  0.0345,  0.0346,  0
                                    .0347,  0.0349,  0.0350,  0.0351,  0
                                    .0352,  0.0354,  0.0355,  0.0356,  0
                                    .0357,  0.0359,  0.0360,  0.0361,  0
                                    .0362,  0.0364,  0.0365,  0.0366,  0
                                    .0368,  0.0369,  0.0370,  0.0372,  0
                                    .0373,  0.0374,  0.0376,  0.0377,  0
                                    .0379,  0.0380,  0.0381,  0.0383,  0
                                    .0384,  0.0386,  0.0387,  0.0389,  0
                                    .0390,  0.0391,  0.0393,  0.0394,  0
                                    .0396,  0.0397,  0.0399,  0.0400,  0
                                    .0402,  0.0403,  0.0405,  0.0406,  0
                                    .0408,  0.0409,  0.0411,  0.0412,  0
                                    .0414,  0.0415,  0.0417,  0.0418,  0
                                    .0420,  0.0421,  0.0423,  0.0424,  0
```

```
                                .0426,  0.0427,  0.0429,  0.0431]
TimeD_Ref = [0, 0.5,  1,  1.5,  2,  2.5,  3,  3.5,  4,  4.5,  5,  5.
                                5,  6,  6.5,  7,  7.5,  8,  8.5,  9,
                                  9.5,  10, 10.5, 11, 11.5, 12, 12.5
                                , 13, 13.5, 14, 14.5, 15, 15.5, 16,
                                16.5, 17, 17.5, 18, 18.5, 19, 19.5,
                                20, 20.5, 21, 21.5, 22, 22.5, 23, 23
                                .5, 24, 24.5, 25, 25.5, 26, 26.5, 27
                                , 27.5, 28, 28.5, 29, 29.5, 30, 30.5
                                , 31, 31.5, 32, 32.5, 33, 33.5, 34,
                                34.5, 35, 35.5, 36, 36.5, 37, 37.5,
                                38, 38.5, 39, 39.5, 40, 40.5, 41, 41
                                .5, 42, 42.5, 43, 43.5, 44, 44.5, 45
                                , 45.5, 46, 46.5, 47, 47.5, 48, 48.5
                                ]
TimeF_Ref = [0, 0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,1.1,1.2,1.3,
                                1.4,1.5,1.6,1.7,1.8,1.9,2.0,2.1,2.2,
                                2.3,2.4,2.5,2.6,2.7,2.8,2.9,3.,3.1,3
                                .2,3.3,3.4,3.5,3.6,3.7,3.8,3.9,4.,4.
                                1,4.2,4.3,4.4,4.5,4.6,4.7,4.8,4.9,5
                                .,5.1,5.2,5.3,5.4,5.5,5.6,5.7,5.8,5.
                                9,6.,6.1,6.2,6.3,6.4,6.5,6.6,6.7,6.8
                                ,6.9,7.,7.1,7.2,7.3,7.4,7.5,7.6,7.7,
                                7.8,7.9,8.,8.1,8.2,8.3,8.4,8.5,8.6,8
                                .7,8.8,8.9,9.,9.1,9.2,9.3,9.4,9.5,9.
                                6,9.7,9.8,9.9,10.,10.1,10.2,10.3,10.
                                4,10.5,10.6,10.7,10.8,10.9,11.,11.1,
                                11.2,11.3,11.4,11.5,11.6,11.7,11.8,
                                11.9,12.,12.1,12.2,12.3,12.4,12.5,12
                                .6,12.7,12.8,12.9,13.,13.1,13.2,13.3
                                ,13.4,13.5,13.6,13.7,13.8,13.9,14.,
                                14.1,14.2,14.3,14.4,14.5,14.6,14.7,
                                14.8,14.9,15.]
Force_Ref = [3116.60711,  3141.90127, 3175.72355, 3201.79909, 3237.
                                25678, 3264.79226, 3306.25214, 3339.
                                74473, 3388.74944, 3422.12133, 3473.
                                43981, 3515.84628, 3571.40563, 3620.
                                49378, 3685.53065, 3737.75996, 3811.
                                3188,  3876.26104, 3962.87255, 4037.
                                82241, 4136.19652, 4224.84614, 4341.
                                97634, 4453.27088, 4594.92542, 4721.
                                36475, 4886.77733, 5035.92305, 5217.
                                82301, 5384.25595, 5580.90247, 5756.
                                82074, 5973.94221, 6159.69151, 6388.
                                44818, 6585.71273, 6816.85284, 7024.
                                96767, 7276.75334, 7488.89521, 7742.
                                2604,  7965.27565, 8227.59569, 8451.
```

```
                                    84326, 8724.80571, 8954.53468, 9225.
                                    74401, 9461.69272, 9743.41556, 9987.
                                    03241, 10272.24958,  10513.69682,
                                    10800.96364,  11048.54494,  11337.
                                    68484,  11589.09798,  11879.71309,
                                    12130.43705,  12421.22352,  12680.
                                    65721,  12974.43956,  13230.37744,
                                    13524.12105,  13787.2681, 14077.
                                    89439,  14345.73233,  14637.2512,
                                    14900.69926,  15191.34641,  15462.
                                    44025,  15756.33734,  16028.62775,
                                    16320.33736,  16592.43256,  16881.
                                    07401,  17157.99272,  17447.0529,
                                    17720.88259,  18011.99615,  18287.
                                    72277,  18585.28256,  18863.88958,
                                    19147.10402,  19427.48874,  19720.
                                    70336,  19996.86956,  20290.41052,
                                    20573.43423,  20858.45172,  21137.
                                    99155,  21426.88781,  21712.704,
                                    21995.37903,  22283.94002,  22574.
                                    56034,  22852.1958, 23144.95891,
                                    23426.4791, 23705.48695,  23992.
                                    71131,  24277.04632,  24563.27975,
                                    24848.68318,  25132.54285,  25408.
                                    95939,  25696.37895,  25977.36418,
                                    26266.3573, 26546.37098,  26833.
                                    51636,  27104.27642,  27399.11675,
                                    27674.64221,  27958.81331,  28230.
                                    24094,  28529.13439,  28793.3588,
                                    29092.14497,  29361.36425,  29648.
                                    18776,  29910.66277,  30212.10134,
                                    30475.75057,  30774.90926,  31038.
                                    21278,  31336.68602,  31589.09082,
                                    31891.71851,  32142.11762,  32436.
                                    06091,  32695.40668,  32987.74362,
                                    33244.30585,  33550.77505,  33794.
                                    08419,  34088.85598,  34342.04757,
                                    34634.75108,  34887.12013,  35190.
                                    624,  35432.60098,  35724.84255,
                                    35971.55511,  36266.69347,  36504.
                                    23586,  36801.97001,  37043.82479,
                                    37328.42207,  37569.02218,  37862.
                                    30385,  38091.40921]
    Velocidade_Compressao = 6.e-5

if taxa_def == 3:
  if MAT == 1:
```

```
Disp_Ref = [0.0300,  0.0302,  0.0302,  0.0302,  0.0302,  0.0301,  0
                     .0303,  0.0304,  0.0304,  0.0305,  0
                     .0306,  0.0308,  0.0308,  0.0308,  0
                     .0309,  0.0310,  0.0308,  0.0310,  0
                     .0310,  0.0311,  0.0312,  0.0313,  0
                     .0314,  0.0315,  0.0317,  0.0317,  0
                     .0318,  0.0319,  0.0319,  0.0320,  0
                     .0321,  0.0323,  0.0326,  0.0327,  0
                     .0327,  0.0329,  0.0331,  0.0332,  0
                     .0333,  0.0333,  0.0335,  0.0336,  0
                     .0338,  0.0339,  0.0339,  0.0340,  0
                     .0342,  0.0343,  0.0344,  0.0344,  0
                     .0346,  0.0348,  0.0348,  0.0350,  0
                     .0350,  0.0352,  0.0354,  0.0354,  0
                     .0358,  0.0356,  0.0359,  0.0371,  0
                     .0446,  0.0362,  0.0363,  0.0366,  0
                     .0365,  0.0366,  0.0368,  0.0369,  0
                     .0371,  0.0373,  0.0373,  0.0375,  0
                     .0376,  0.0378,  0.0379,  0.0382,  0
                     .0382,  0.0385,  0.0385,  0.0386,  0
                     .0388,  0.0390,  0.0392,  0.0393,  0
                     .0462,  0.0396,  0.0398,  0.0400,  0
                     .0402,  0.0403,  0.0466,  0.0406,  0
                     .0408,  0.0410,  0.0412,  0.0414,  0
                     .0415,  0.0470,  0.0418,  0.0420,  0
                     .0423,  0.0425,  0.0427,  0.0429,  0
                     .0431,  0.0433,  0.0435,  0.0437,  0
                     .0441,  0.0441,  0.0466,  0.0445]
TimeD_Ref = [0.0000,  4.0000,  8.0000, 12.0000, 16.0000, 20.0000, 24
                     .0000, 28.0000, 32.0000, 36.0000, 40
                     .0000, 44.0000, 48.0000, 52.0000, 56
                     .0000, 60.0000, 64.0000, 68.0000, 72
                     .0000, 76.0000, 80.0000, 84.0000, 88
                     .0000, 92.0000, 96.0000,100.0000,104
                     .0000,108.0000,112.0000,116.0000,120
                     .0000,124.0000,128.0000,132.0000,136
                     .0000,140.0000,144.0000,148.0000,152
                     .0000,156.0000,160.0000,164.0000,168
                     .0000,172.0000,176.0000,180.0000,184
                     .0000,188.0000,192.0000,196.0000,200
                     .0000,204.0000,208.0000,212.0000,216
                     .0000,220.0000,224.0000,228.0000,232
                     .0000,236.0000,240.0000,244.0000,248
                     .0000,252.0000,256.0000,260.0000,264
                     .0000,268.0000,272.0000,276.0000,280
                     .0000,284.0000,288.0000,292.0000,296
                     .0000,300.0000,304.0000,308.0000,312
```

```
                                .0000,316.0000,320.0000,324.0000,328
                                .0000,332.0000,336.0000,340.0000,344
                                .0000,348.0000,352.0000,356.0000,360
                                .0000,364.0000,368.0000,372.0000,376
                                .0000,380.0000,384.0000,388.0000,392
                                .0000,396.0000,400.0000,404.0000,408
                                .0000,412.0000,416.0000,420.0000,424
                                .0000,428.0000,432.0000,436.0000,440
                                .0000,444.0000,448.0000,452.0000]
    TimeF_Ref = [0.0000,  4.0000,  8.0000, 12.0000, 16.0000, 20.0000, 24
                                .0000, 28.0000, 32.0000, 36.0000, 40
                                .0000, 44.0000, 48.0000, 52.0000, 56
                                .0000, 60.0000, 64.0000, 68.0000, 72
                                .0000, 76.0000, 80.0000, 84.0000, 88
                                .0000, 92.0000, 96.0000,100.0000,104
                                .0000,108.0000,112.0000,116.0000,120
                                .0000,124.0000,128.0000,132.0000,136
                                .0000,140.0000,144.0000,148.0000,152
                                .0000,156.0000,160.0000,164.0000,168
                                .0000,172.0000,176.0000,180.0000,184
                                .0000,188.0000,192.0000,196.0000,200
                                .0000,204.0000,208.0000,212.0000,216
                                .0000,220.0000,224.0000,228.0000,232
                                .0000,236.0000,240.0000,244.0000,248
                                .0000,252.0000,256.0000,260.0000,264
                                .0000,268.0000,272.0000,276.0000,280
                                .0000,284.0000,288.0000,292.0000,296
                                .0000,300.0000,304.0000,308.0000,312
                                .0000,316.0000,320.0000,324.0000,328
                                .0000,332.0000,336.0000,340.0000,344
                                .0000,348.0000,352.0000,356.0000,360
                                .0000,364.0000,368.0000,372.0000,376
                                .0000,380.0000,384.0000,388.0000,392
                                .0000,396.0000,400.0000,404.0000,408
                                .0000,412.0000,416.0000,420.0000,424
                                .0000,428.0000,432.0000,436.0000,440
                                .0000,444.0000,448.0000,452.0000]
    Force_Ref = [0.0506,  5.7285, 14.4888, 29.3059, 56.0543, 93.2047,144
                                .1027,210.2402,292.2751,382.2826,475
                                .0973,569.9774,665.4241,759.0893,850
                                .7050,945.5144,1038.2994,1126.9662,
                                1217.5705,1308.4014,1394.4455,1480.
                                9141,1567.3930,1652.9810,1736.2425,
                                1818.7823,1902.4197,1986.2374,2065.
                                6917,2148.4513,2229.9912,2307.6540,
                                2386.1339,2466.1917,2543.2365,2620.
                                9606,2699.3606,2777.8534,2855.1629,
```

```
                                    2932.1225,3011.4401,3092.2428,3167.
                                    0876,3246.4020,3328.0835,3405.3668,
                                    3482.9199,3563.7882,3644.9440,3726.
                                    0778,3806.5068,3889.7250,3974.9175,
                                    4055.3633,4142.7311,4230.2981,4312.
                                    7213,4397.9473,4487.6188,4573.7252,
                                    4661.4520,4750.4429,4842.5108,4933.
                                    1453,5023.1803,5119.4809,5216.1727,
                                    5306.4447,5403.4069,5503.9220,5598.
                                    5045,5694.9951,5794.0189,5894.9243,
                                    5995.4081,6095.9775,6201.5601,6308.
                                    4222,6409.4074,6520.1715,6633.2161,
                                    6737.4662,6846.8437,6961.7517,7071.
                                    8996,7184.7044,7300.0759,7420.1889,
                                    7539.4027,7658.2290,7787.9749,7921.
                                    0803,8043.5909,8178.3824,8318.2923,
                                    8449.9173,8586.8053,8729.9600,8877.
                                    6737,9024.5925,9173.6056,9331.6115,
                                    9492.8712,9645.5656,9817.6524,9995.
                                    0261,10158.7579,10333.2616,10518.
                                    0509,10697.4341,10885.0844,11075.
                                    6308,11277.0356,11475.8588,11677.
                                    2644,11895.6596,12121.1954,12332.
                                    2740,12562.1557,12803.5888,13035.
                                    0933,13272.3153,13521.8948,13782.
                                    5206,14042.2881,14306.0133,14589.
                                    2173,14878.4161,15157.3688,15467.
                                    9492,15787.0606,16088.2860,16406.
                                    0667,16743.2606,17075.3136,17417.
                                    7244,17770.5422,18140.5798,18510.
                                    4877]
  Velocidade_Compressao = 7.1e-5

if MAT == 2:
  Disp_Ref = [0.0300,  0.0302,  0.0302,  0.0302,  0.0302,  0.0301,  0.
                                    0303,  0.0304,  0.0304,  0.0305,  0.
                                    0306,  0.0308,  0.0308,  0.0308,  0.
                                    0309,  0.0310,  0.0308,  0.0310,  0.
                                    0310,  0.0311,  0.0312,  0.0313,  0.
                                    0314,  0.0315,  0.0317,  0.0317,  0.
                                    0318,  0.0319,  0.0319,  0.0320,  0.
                                    0321,  0.0323,  0.0326,  0.0327,  0.
                                    0327,  0.0329,  0.0331,  0.0332,  0.
                                    0333,  0.0333,  0.0335,  0.0336,  0.
                                    0338,  0.0339,  0.0339,  0.0340,  0.
                                    0342,  0.0343,  0.0344,  0.0344,  0.
                                    0346,  0.0348,  0.0348,  0.0350,  0.
```

```
                                0350,  0.0352,  0.0354,  0.0354,  0.
                                0358,  0.0356,  0.0359,  0.0371,  0.
                                0446,  0.0362,  0.0363,  0.0366,  0.
                                0365,  0.0366,  0.0368,  0.0369,  0.
                                0371,  0.0373,  0.0373,  0.0375,  0.
                                0376,  0.0378,  0.0379,  0.0382,  0.
                                0382,  0.0385,  0.0385,  0.0386,  0.
                                0388,  0.0390,  0.0392,  0.0393,  0.
                                0462,  0.0396,  0.0398,  0.0400,  0.
                                0402,  0.0403,  0.0466,  0.0406,  0.
                                0408,  0.0410,  0.0412,  0.0414,  0.
                                0415,  0.0470,  0.0418,  0.0420,  0.
                                0423,  0.0425,  0.0427,  0.0429,  0.
                                0431,  0.0433,  0.0435,  0.0437,  0.
                                0441,  0.0441,  0.0466,  0.0445]
    TimeD_Ref = [0,   4,  8,  12,   16,   20,   24,   28,   32,   36,
                                40,   44,   48,   52,   56,   60,
                                64,   68,   72,   76,   80,   84,
                                88,   92,   96,   100,  104,  108.
                                0000,   112.0000,   116.0000,   120.
                                0000,   124.0000,   128.0000,   132.
                                00,   136.00,   140.00,   144.00,
                                148.00,   152.00,   156.00,   160.00
                                ,   164.00,   168.0000,   172.00,
                                176.00,   180.00,   184.00,   188.00
                                ,   192.00,   196.00,   200.00,
                                204.00,   208.00,   212.00,   216.00
                                ,   220.00,   224.00,   228.00,
                                232.00,   236.00,   240.00,   244.00
                                ,   248.00,   252.00,   256.00,
                                260.00,   264.00,   268.00,   272.00
                                ,   276.00,   280.00,   284.00]
    TimeF_Ref = [0.,  4.0,  8.0,  12.0, 16.0, 20.0, 24.0, 28.0, 32.0, 36
                                .0, 40.0, 44.0, 48.0, 52.0, 56.0, 60
                                .0, 64.0, 68.0, 72.0, 76.0, 80.0, 84
                                .0, 88.0, 92.0, 96.0, 100.0,  104.0,
                                  108.0,  112.0,  116.0,  120.0,
                                124.0,  128.0,  132.0,  136.0,  140.
                                0,  144.0,  148.0,  152.0,]
    Force_Ref = [3525.58009,  4327.81428, 5302.66203, 6365.95935, 7475.
                                01478, 8598.838, 9724.68778, 10816.
                                54876,  11908.98865,  12916.66031,
                                14056.0627, 15168.81734,  16326.
                                20245,  17465.44242,  18609.41201,
                                19758.61341,  20916.35764,  22071.
                                10673,  23209.72383,  24330.96915,
                                25449.17464,  26569.80753,  27666.
```

```
                                    70883,  28747.07282,  29750.34714,
                                    30858.98757,  31913.08081,  33000.
                                    82088,  34002.3458, 35092.0409,
                                    36177.91831,  37229.75552,  38272.
                                    13049,  39278.30458,  40267.4675,
                                    41208.14204,  42101.5054, 42918.
                                    87879,  43658.21183]
    Velocidade_Compressao = 6.e-6

#-----------------------------
#VARIABLES
#-----------------------------
#Caracteristicas dos corpos de prova

#name, dimensions of the sample [big diameter, small diameter, height,
                                    height of head, lenght of arc],
#linear/quadratic interpolation (False/True), Reduced integration,
                                    middle
param_trac=dict([('name_Sup','Superior'),('name_Inf','Inferior'),('
                                    name_Meio', 'Meio'), ('dim',[0.0125,
                                     0.0064, 0.127, 0.034, 0.013, 0.0001
                                    ]),
  ('quad',False), ('Reduced',True), ('middle', [0.003923, 0.02318]), ('
                                    prolongacao', 0.0135), ('espessura',
                                     0.003), ('teste_mesh', True)])
#dim = diametro, altura
if MAT == 1:
  param_comp=dict([('name','compressao'), ('dim',[0.03,0.009]), ('quad',
                                    False), ('Reduced',True),('
                                    dia_support', 0.05)])
if MAT == 2:
  param_comp=dict([('name','compressao'), ('dim',[0.03,0.0055]), ('quad'
                                    ,False), ('Reduced',True),('
                                    dia_support', 0.05)])

#Material - Aco 1020 standard
#name, density, Young modulus, Poisson ratio, initial yield stress,
                                    linear hardening coefficient
mat=dict([('name','steel'),('dens',7850.),('E',2.1e11),('nu',0.3),('Re',
                                    200.e6),('E1',1.0),('K', 530.e6),('n
                                    ',0.26)])

if MAT == 1:

                                                #1179907.97, 3.
                                    54804, 0.0
```

```
  # mat_EVA=dict([('name', 'EVA_Flex'), ('dens', 2000), ('E', 30.e6), ('
                                    Re', 5.e6), ('nu', 0.48), ('E1', 1.0
                                    ),('g_i', [7.50E-1, 0.3, 0.3134, 0.
                                    786]),('k_i',[0.0, 0.0, 0.0, 0.0])
                                    ,('tau',[5.000E0, 0.0647, 0.0001163,
                                     7.321e-7]), ('Arruda_Boyce', [1.
                                    255E6, 3.478E1, 0.0]),
  #          ('Ogden1', [7.e7, 0.8, 0.0]), ('Ogden2', [2.6e6, 2.6, 0.0])
                                    ])
    mat_EVA=dict([('name', 'EVA_Flex'), ('dens', 2500), ('E', 30.e6), ('
                                        Re', 5.e6), ('nu', 0.48), ('E1',
                                         1.0),('g_i', [2.80E-1, 0.20, 0.
                                        186, 0.1]),('k_i',[0.0, 0.0, 0.0
                                        , 0.0]),('tau',[5.000E-5, 0.
                                        000647, 0.01163, 7.321]), ('
                                        Arruda_Boyce', [1.755E6, 3.478E1
                                        , 0.0]),
                    ('Ogden1', [7.e7, 0.8, 0.0]), ('Ogden2', [2.6e6, 2.6
                                                        , 0.0])])
if MAT == 2:                                                          ##
                                        A      B     n     m    Melt   Tra
                                       C    Epsilonp
  mat_EVA=dict([('name', 'EVA_Rigid'), ('dens', 1200), ('E', 65.e6), ('
                                    Re', 21.e6), ('nu', 0.49), ('E1', 1.
                                    0), ('Johnson_Cook', [105.e6, 445.e6
                                    , 0.7, 1.0, 0.0, 0.0, 0.007, 1.0]),
                                    ])

#Paramteros da geometria do problema
param_osso = dict([('name', 'Osso'), ('Mat_Name', 'Mat_Osso'), ('dens',
                                    1283), ('E', 1.5e7), ('nu', 0.49), (
                                    'SigmaY', 115.06e6), ('EpsilonY', 0.
                                    0), ('SigmaU', 153.59e6),('EpsilonU'
                                    , 0.32), ('Diametro', 0.085), ('
                                    Altura', 0.0103), ('quad', False),
                    ('g_i', [0.305, 0,45]),('k_i',[0.01, 0.2]),('tau',[0
                                                        .00001, 0.00005]
                                                        )])

param_pele = dict([('name', 'Pele'), ('Mat_Name', 'Mat_Pele'), ('dens',
                                    1020), ('E', 0.7e6), ('nu', 0.4999),
                                     ('Diametro', 0.085), ('Altura', 0.
                                    0123), ('quad', False),
                    ('Ogden1', [4.6e6, 1.4, 0.0]), ('Ogden2', [2.7e3, 10
                                                        .0, 0.0]), ('g_i
                                                        ', [0.051, 0.10,
                                                         0.120, 0.019])
```

```
                                                    ,('k_i',[0.02, 0
                                                    .150, 0.120, 0.
                                                    019]),('tau',[0.
                                                    00063, 0.0366,
                                                    13.10, 94.56])])
                                                     #('g_i', [0.051
                                                    , 0.150, 0.120,
                                                    0.019]),('k_i',[
                                                    0.051, 0.150, 0.
                                                    120, 0.019]),('
                                                    tau',[0.63, 3.66
                                                    , 13.10, 94.56])

param_bola = dict([('name','Esfera'), ('Diametro', 0.04267), ('Massa',
                                    45.93e-3), ('Velocidade', 8.5), ('
                                    quad', False)])

param_FLEX = dict([('name', 'FLEX'), ('Mat_Name','EVA_Flex'), ('Diametro
                                    ', 0.085), ('Altura', 0.003), ('quad
                                    ', False)])
param_RIG = dict([('name', 'RIG'), ('Mat_Name','EVA_Rigid'), ('Diametro'
                                    , 0.085), ('Altura', 0.001), ('quad'
                                    , False)])
param_impacto=dict([('name','Suporte'), ('dim',[0.03,0.009]), ('quad',
                                    False), ('Reduced',True),('
                                    dia_support', 0.15)])

#Definicoes dos materiais do problema
mat_EVA_FLEX=dict([('name', 'EVA_Flex'), ('dens', 2500), ('E', 30.e6), (
                                    'Re', 5.e6), ('nu', 0.48), ('E1', 1.
                                    0),('g_i', [2.8E-1, 0.2, 0.186, 0.1]
                                    ),('k_i',[0.0, 0.0, 0.0, 0.0]),('tau
                                    ',[5.000E-5, 0.000647, 0.01163, 7.
                                    321]), ('Arruda_Boyce', [1.755E6, 3.
                                    478E1, 0.0]),
                    ('Ogden1', [7.e7, 0.8, 0.0]), ('Ogden2', [2.6e6, 2.6
                                                    , 0.0])])
mat_EVA_RIG=dict([('name', 'EVA_Rigid'), ('dens', 1200), ('E', 65.e6), (
                                    'Re', 21.e6), ('nu', 0.49), ('E1', 1
                                    .0), ('Johnson_Cook', [105.e6, 445.
                                    e6, 0.7, 1.0, 0.0, 0.0, 0.007, 1.0])
                                    , ])
#Parametros da simulacao
#Velocidade do ensaio de tracao, Velocidade do ensaio de compressao,
                                    numero de passos maximo, tamanho dos
                                     elementos de malha.
```

```
simu=dict([('Vel_tra', 0.0125), ('Vel_comp', Velocidade_Compressao), ('
                                    npas',1000), ('big_number', 40), ('
                                    small_number', 10),('medium_number',
                                     30),('sup_number', 20),
 ('raio_cilindro', 0.005), ('ratio', 5.0), ('global_meio', 0.0005), ('
                                     global_extremidades', 0.0045), ('
                                     time_Implicit', 100), ('
                                     Mesh_Protetor_Flex', param_FLEX['
                                     Altura']/1.8), ('Mesh_Protetor_Rig'
                                     , param_RIG['Altura']), ('Mesh_Osso
                                     ', param_osso['Altura']/3.), ('
                                     Mesh_Pele', param_pele['Altura']/3.
                                     5) ])



#                                   ------------------------------------------------

### MODELING
#                                   ------------------------------------------------

def sketch_tra(param_trac):
  ''' Sketch de tracao '''

  s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__1',
  sheetSize=200.0)
  g1, v1, d1, c1 = s1.geometry, s1.vertices, s1.dimensions, s1.
                                    constraints
  s1.setPrimaryObject(option=STANDALONE)

  s2 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__2',
  sheetSize=200.0)
  g2, v2, d2, c2 = s2.geometry, s2.vertices, s2.dimensions, s2.
                                    constraints
  s2.setPrimaryObject(option=STANDALONE)

  s3 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__3',
  sheetSize=200.0)
  g3, v3, d3, c3 = s3.geometry, s3.vertices, s3.dimensions, s3.
                                    constraints
  s3.setPrimaryObject(option=STANDALONE)

  s1.ConstructionLine(point1=(0.0, -100.0), point2=(0.0, 100.0))
  s2.ConstructionLine(point1=(0.0, -100.0), point2=(0.0, 100.0))
```

```
s3.ConstructionLine(point1=(0.0, -100.0), point2=(0.0, 100.0))


point1 = (0.0, param_trac['dim'][2]/2)
point2 = (param_trac['dim'][0]/2., param_trac['dim'][2]/2.)
point3 = (param_trac['dim'][0]/2., param_trac['dim'][2]/2. -
                                   param_trac['dim'][3])
point4 = (param_trac['dim'][1]/2., param_trac['dim'][2]/2. -
                                   param_trac['dim'][3] - param_trac['
                                   dim'][4])
point5 = (param_trac['middle'][0], param_trac['middle'][1])
point5_5 = (0.0, param_trac['dim'][2]/2. - param_trac['dim'][3] -
                                   param_trac['dim'][4] - param_trac['
                                   prolongacao'])
point4_40 = (param_trac['dim'][1]/2., param_trac['dim'][2]/2. -
                                   param_trac['dim'][3] - param_trac['
                                   dim'][4] - param_trac['prolongacao']
                                   )
point4_41 = (0.0, param_trac['dim'][2]/2. - param_trac['dim'][3] -
                                   param_trac['dim'][4] - param_trac['
                                   prolongacao'])

point11 = (0.0, param_trac['dim'][2]/2. - param_trac['dim'][3] -
                                   param_trac['dim'][4] - param_trac['
                                   prolongacao'])
point12 = (param_trac['dim'][1]/2., param_trac['dim'][2]/2. -
                                   param_trac['dim'][3] - param_trac['
                                   dim'][4] - param_trac['prolongacao']
                                   )
point13 = (param_trac['dim'][1]/2., -(param_trac['dim'][2]/2. -
                                   param_trac['dim'][3] - param_trac['
                                   dim'][4] - param_trac['prolongacao']
                                   ))
point14 = (0.0, -(param_trac['dim'][2]/2. - param_trac['dim'][3] -
                                   param_trac['dim'][4] - param_trac['
                                   prolongacao']))
point15 = (param_trac['dim'][1]/2., 10.0*param_trac['dim'][5])
point16 = (param_trac['dim'][1]/2. - param_trac['dim'][5]/2., 0.0)
point17 = (param_trac['dim'][1]/2., -10.0*param_trac['dim'][5])

point6 = (param_trac['dim'][1]/2., -(param_trac['dim'][2]/2. -
                                   param_trac['dim'][3] - param_trac['
                                   dim'][4]))
point7 = (param_trac['dim'][0]/2., -(param_trac['dim'][2]/2. -
                                   param_trac['dim'][3]))
point8 = (param_trac['middle'][0], -param_trac['middle'][1])
point9 = (param_trac['dim'][0]/2., -param_trac['dim'][2]/2.)
```

```
  point10 = (0.0, -param_trac['dim'][2]/2.)
  point10_10 = (0.0, -(param_trac['dim'][2]/2. - param_trac['dim'][3] -
                                    param_trac['dim'][4] - param_trac['
                                    prolongacao']))
  point6_60 = (param_trac['dim'][1]/2., -(param_trac['dim'][2]/2. -
                                    param_trac['dim'][3] - param_trac['
                                    dim'][4] - param_trac['prolongacao']
                                    ))
  point6_61 = (0.0, -(param_trac['dim'][2]/2. - param_trac['dim'][3] -
                                    param_trac['dim'][4] - param_trac['
                                    prolongacao']))



  #Definicao das linhas do sketch
  #Superior
  s1.Line(point1, point2)
  s1.Line(point2, point3)
  s1.Arc3Points(point1=(point3), point2=(point4), point3=(point5))
  s1.Line(point4,point4_40)
  s1.Line(point4_40,point4_41)
  s1.Line(point4_41,point5_5)
  s1.Line(point5_5,point1)
  #Inferior
  s2.Line(point10_10,point6_61)
  s2.Line(point6_61,point6_60)
  s2.Line(point6_60,point6)
  s2.Arc3Points(point1=(point6), point2=(point7), point3=(point8))
  s2.Line(point7, point9)
  s2.Line(point9, point10)
  s2.Line(point10, point10_10)
  #Meio
  s3.Line(point11, point12)
  s3.Line(point12,point15)
  s3.Line(point15,point16)
  s3.Line(point16, point17)
  s3.Line(point17,point13)
  s3.Line(point13,point14)
  s3.Line(point14,point11)


  return s1, s2, s3

def sketch_compressao(param_comp):
  ''' Sketch de Compressao '''
  s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__1',
  sheetSize=200.0)
```

```
  g1, v1, d1, c1 = s1.geometry, s1.vertices, s1.dimensions, s1.
                                        constraints
  s1.setPrimaryObject(option=STANDALONE)

  s1.CircleByCenterPerimeter(center=(0.0, 0.0), point1=(0.0, param_comp[
                                        'dim'][0]/2.))

  return s1

def sketch_osso(param_osso):
    ''' Sketch do disco de Osso '''
    s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__1',
    sheetSize=200.0)
    g1, v1, d1, c1 = s1.geometry, s1.vertices, s1.dimensions, s1.
                                            constraints
    s1.setPrimaryObject(option=STANDALONE)

    s1.CircleByCenterPerimeter(center=(0.0, 0.0), point1=(0.0,
                                            param_osso['Diametro']/2.))

    return s1

def sketch_pele(param_pele):
    ''' Sketch do disco de Pele '''
    s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__1',
    sheetSize=200.0)
    g1, v1, d1, c1 = s1.geometry, s1.vertices, s1.dimensions, s1.
                                            constraints
    s1.setPrimaryObject(option=STANDALONE)

    s1.CircleByCenterPerimeter(center=(0.0, 0.0), point1=(0.0,
                                            param_osso['Diametro']/2.))

    return s1

def sketch_FLEX(param_osso):
    ''' Sketch do disco de Pele '''
    s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__1',
    sheetSize=200.0)
    g1, v1, d1, c1 = s1.geometry, s1.vertices, s1.dimensions, s1.
                                            constraints
    s1.setPrimaryObject(option=STANDALONE)

    s1.CircleByCenterPerimeter(center=(0.0, 0.0), point1=(0.0,
                                            param_osso['Diametro']/2.))

    return s1
```

```
def sketch_RIG(param_osso):
    ''' Sketch do disco de Pele '''
    s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__1',
    sheetSize=200.0)
    g1, v1, d1, c1 = s1.geometry, s1.vertices, s1.dimensions, s1.
                                        constraints
    s1.setPrimaryObject(option=STANDALONE)

    s1.CircleByCenterPerimeter(center=(0.0, 0.0), point1=(0.0,
                                        param_osso['Diametro']/2.))

    return s1

def sketch_bola(param_bola):
    ''' Sketch do disco de Pele '''
    s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__1',
    sheetSize=200.0)
    g1, v1, d1, c1 = s1.geometry, s1.vertices, s1.dimensions, s1.
                                        constraints
    s1.setPrimaryObject(option=STANDALONE)

    point1 = (0.0,param_bola['Diametro']/2.)
    point2 = (0.0,-param_bola['Diametro']/2.)
    point3 = (param_bola['Diametro']/2.,0.0)

    s1.Arc3Points(point1=point1, point2=point2, point3=point3)
    s1.Line(point1,point2)

    return s1


def Supports_Compression(param_comp, namec):
  ''' Cria os suportes de compressao '''
  s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
  g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
  s.setPrimaryObject(option=STANDALONE)

  s.CircleByCenterPerimeter(center=(0.0, 0.0), point1=(0.0, param_comp['
                                    dia_support']/2.))
  p = mdb.models['Model-1'].Part(name=namec, dimensionality=THREE_D,
      type=DISCRETE_RIGID_SURFACE)
  p = mdb.models['Model-1'].parts[namec]
  p.BaseShell(sketch=s)
  s.unsetPrimaryObject()
  p = mdb.models['Model-1'].parts[namec]
```

```
  session.viewports['Viewport: 1'].setValues(displayedObject=p)
  del mdb.models['Model-1'].sketches['__profile__']
  #Criacao de um reference point no centro
  p = mdb.models['Model-1'].parts[namec]
  p.DatumPointByCoordinate(coords=(0.0, 0.0, 0.0))
  p = mdb.models['Model-1'].parts[namec]
  p.ReferencePoint(point=(0.0, 0.0, 0.0))

  #Criacao de sets uteis
  p.Surface(side1Faces=p.faces.getByBoundingBox(), name=namec)
  p.Set(edges=p.edges.getByBoundingBox(), name=namec)

  #mesh
  elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=STANDARD)
  elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=STANDARD)
  p.setElementType(regions=(p.surfaces[namec].faces,), elemTypes=(
                                    elemType1, elemType2 ))
  p.seedPart(size=simu['global_extremidades'], deviationFactor=0.1,
                                    minSizeFactor=0.1)
  p.generateMesh()

  return p

def traction_geometry(param_trac, mat):
  ''' Definicao da geometria do ensaio de tracao'''

  #Chama o sketch para ter uma base
  s1, s2, s3=sketch_tra(param_trac)

  #Assimila uma geometria ao sketch
  #Parte Superior
  p = mdb.models['Model-1'].Part(name=param_trac['name_Sup'],
                                    dimensionality=THREE_D, type=
                                    DEFORMABLE_BODY)
  p = mdb.models['Model-1'].parts[param_trac['name_Sup']]
  p.BaseSolidExtrude(sketch=s1, depth=param_trac['espessura'])
  s1.unsetPrimaryObject()
  p = mdb.models['Model-1'].parts[param_trac['name_Sup']]
  session.viewports['Viewport: 1'].setValues(displayedObject=p)
  del mdb.models['Model-1'].sketches['__profile__1']
  p = mdb.models['Model-1'].parts['Superior']
  f1 = p.faces
  p.Mirror(mirrorPlane=f1[5], keepOriginal=ON)
  #Parte Inferior
  p = mdb.models['Model-1'].Part(name=param_trac['name_Inf'],
                                    dimensionality=THREE_D, type=
                                    DEFORMABLE_BODY)
```

```
p = mdb.models['Model-1'].parts[param_trac['name_Inf']]
p.BaseSolidExtrude(sketch=s2, depth=param_trac['espessura'])
s2.unsetPrimaryObject()
p = mdb.models['Model-1'].parts[param_trac['name_Inf']]
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['Model-1'].sketches['__profile__2']
p = mdb.models['Model-1'].parts['Inferior']
f = p.faces
p.Mirror(mirrorPlane=f[5], keepOriginal=ON)
#Parte do Meio
p = mdb.models['Model-1'].Part(name=param_trac['name_Meio'],
                               dimensionality=THREE_D, type=
                               DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts[param_trac['name_Meio']]
p.BaseSolidExtrude(sketch=s3, depth=param_trac['espessura'])
s3.unsetPrimaryObject()
p = mdb.models['Model-1'].parts[param_trac['name_Meio']]
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['Model-1'].sketches['__profile__3']
p = mdb.models['Model-1'].parts['Meio']
f1 = p.faces
p.Mirror(mirrorPlane=f1[6], keepOriginal=ON)
#########Criacao das particoes##############

#Face com z = 0.003
p = mdb.models['Model-1'].parts['Meio']
f, e, d = p.faces, p.edges, p.datums
t = p.MakeSketchTransform(sketchPlane=f[10], sketchUpEdge=e[18],
    sketchPlaneSide=SIDE1, origin=(0.0, 0.0, 0.0))
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=0.0331, gridSpacing=0.0008, transform=t)
g, v, d1, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.sketchOptions.setValues(decimalPlaces=4)
p = mdb.models['Model-1'].parts['Meio']
p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)
point15 = (param_trac['dim'][1]/2., 10.0*param_trac['dim'][5])
point16 = (param_trac['dim'][1]/2. - param_trac['dim'][5]/2., 0.0)
point17 = (param_trac['dim'][1]/2., -10.0*param_trac['dim'][5])
point15_15 = (-param_trac['dim'][1]/2., 10.0*param_trac['dim'][5])
point16_16 = (-(param_trac['dim'][1]/2. - param_trac['dim'][5]/2.), 0.
                                0)
point17_17 = (-param_trac['dim'][1]/2., -10.0*param_trac['dim'][5])
s.Line(point1=(point15_15), point2=(point15))
s.Line(point1=(point16_16), point2=(point16))
s.Line(point1=(point17_17), point2=(point17))
p = mdb.models['Model-1'].parts['Meio']
f = p.faces
```

```
pickedFaces = f.getByBoundingBox(zMin = param_trac['espessura'])
e1, d2 = p.edges, p.datums
p.PartitionFaceBySketch(sketchUpEdge=e1[18], faces=pickedFaces, sketch
                                    =s)
s.unsetPrimaryObject()
del mdb.models['Model-1'].sketches['__profile__']

###Face com z = 0.0
p = mdb.models['Model-1'].parts['Meio']
f, e, d = p.faces, p.edges, p.datums
t = p.MakeSketchTransform(sketchPlane=f[14], sketchUpEdge=e[14],
    sketchPlaneSide=SIDE1, origin=(0.0, 0.0, 0.003))
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=0.0331, gridSpacing=0.0008, transform=t)
g, v, d1, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.sketchOptions.setValues(decimalPlaces=4)
p = mdb.models['Model-1'].parts['Meio']
p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)
point15 = (param_trac['dim'][1]/2., 10.0*param_trac['dim'][5])
point16 = (param_trac['dim'][1]/2. - param_trac['dim'][5]/2., 0.0)
point17 = (param_trac['dim'][1]/2., -10.0*param_trac['dim'][5])
point15_15 = (-param_trac['dim'][1]/2., 10.0*param_trac['dim'][5])
point16_16 = (-(param_trac['dim'][1]/2. - param_trac['dim'][5]/2.), 0.
                                    0)
point17_17 = (-param_trac['dim'][1]/2., -10.0*param_trac['dim'][5])
s.Line(point1=(point15_15), point2=(point15))
s.Line(point1=(point16_16), point2=(point16))
s.Line(point1=(point17_17), point2=(point17))
p = mdb.models['Model-1'].parts['Meio']
f = p.faces
pickedFaces = f.getByBoundingBox(zMax = 0.0)
e1, d2 = p.edges, p.datums
p.PartitionFaceBySketch(sketchUpEdge=e1[14], faces=pickedFaces, sketch
                                    =s)
s.unsetPrimaryObject()
del mdb.models['Model-1'].sketches['__profile__']

########Criacao de Sets uteis
######SUPERIOR
p = mdb.models['Model-1'].parts[param_trac['name_Sup']]
e = p.edges
edges = e.getByBoundingBox(yMax = param_trac['dim'][2]/2. - param_trac
                                    ['dim'][3] - param_trac['dim'][4])
p.Set(edges=edges, name='Edges_verticais_Sup')

p = mdb.models['Model-1'].parts[param_trac['name_Sup']]
e = p.edges
```

```
edges = e.getByBoundingBox(yMax = param_trac['dim'][2]/2. - param_trac
                                  ['dim'][3] - param_trac['dim'][4] -
                                  param_trac['prolongacao'])
p.Set(edges=edges, name='Edges_Horizontais_Sup')

p = mdb.models['Model-1'].parts[param_trac['name_Sup']]
f = p.faces
faces = f.getByBoundingBox()
p.Set(faces=faces, name='superior')

######INFERIOR
p = mdb.models['Model-1'].parts[param_trac['name_Inf']]
f = p.faces
faces = f.getByBoundingBox()
p.Set(faces=faces, name='inferior')

p = mdb.models['Model-1'].parts[param_trac['name_Inf']]
e = p.edges
edges = e.getByBoundingBox(yMin = -(param_trac['dim'][2]/2. -
                                  param_trac['dim'][3] - param_trac['
                                  dim'][4] - param_trac['prolongacao']
                                  ))
p.Set(edges=edges, name='Edges_Horizontais_Inf')

p = mdb.models['Model-1'].parts[param_trac['name_Inf']]
e = p.edges
edges = e.getByBoundingBox(yMin = -(param_trac['dim'][2]/2. -
                                  param_trac['dim'][3] - param_trac['
                                  dim'][4]))
p.Set(edges=edges, name='Edges_verticais_Inf')

#####MEIO
p = mdb.models['Model-1'].parts[param_trac['name_Meio']]
f = p.faces
faces = f.getByBoundingBox()
p.Set(faces=faces, name='Zona_Deformacao')

p = mdb.models['Model-1'].parts[param_trac['name_Meio']]
e = p.edges
edges1 = e.getByBoundingBox(xMin = param_trac['dim'][1]/2. -
                                  param_trac['dim'][5]/2., yMin = 0.0,
                                   yMax = 0.00005)
p.Set(edges=edges1, name='Lateral_1')

p = mdb.models['Model-1'].parts[param_trac['name_Meio']]
e = p.edges
```

```
edges1 = e.getByBoundingBox(xMax = -(param_trac['dim'][1]/2. -
                                    param_trac['dim'][5]/2.), yMin = 0.0
                                    , yMax = 0.00005)
p.Set(edges=edges1, name='Lateral_2')

p = mdb.models['Model-1'].parts[param_trac['name_Meio']]
e = p.edges
edges1 = e.getByBoundingBox(zMin = param_trac['espessura'], yMin = 0.0
                                    , yMax = 0.00005)
p.Set(edges=edges1, name='Frontal_1')

p = mdb.models['Model-1'].parts[param_trac['name_Meio']]
e = p.edges
edges1 = e.getByBoundingBox(zMax = 0.0, yMin = 0.0, yMax = 0.00005)
p.Set(edges=edges1, name='Frontal_2')


#Criacao da secao homogenea
#Superior
mdb.models['Model-1'].HomogeneousSolidSection(name='Superior',
                                    material=mat['name'], thickness=None
                                    )
p = mdb.models['Model-1'].parts[param_trac['name_Sup']]
f = p.cells.getByBoundingBox()
region1 = regionToolset.Region(cells=f)
p.SectionAssignment(region=region1, sectionName='Superior', offset=0.0
                                    ,  offsetType=MIDDLE_SURFACE,
                                    offsetField='', thicknessAssignment=
                                    FROM_SECTION)
#Inferior
mdb.models['Model-1'].HomogeneousSolidSection(name='Inferior',
                                    material=mat['name'], thickness=None
                                    )
p = mdb.models['Model-1'].parts[param_trac['name_Inf']]
f = p.cells.getByBoundingBox()
region1 = regionToolset.Region(cells=f)
p.SectionAssignment(region=region1, sectionName='Inferior', offset=0.0
                                    ,  offsetType=MIDDLE_SURFACE,
                                    offsetField='', thicknessAssignment=
                                    FROM_SECTION)
#Meio
mdb.models['Model-1'].HomogeneousSolidSection(name='Meio', material=
                                    mat['name'], thickness=None)
p = mdb.models['Model-1'].parts[param_trac['name_Meio']]
f = p.cells.getByBoundingBox()
region1 = regionToolset.Region(cells=f)
```

```
  p.SectionAssignment(region=region1, sectionName='Meio', offset=0.0,
                                      offsetType=MIDDLE_SURFACE,
                                      offsetField='', thicknessAssignment=
                                      FROM_SECTION)


  #Parametros para a malhagem
  r1 = p.cells.getByBoundingBox()
  if param_trac['quad']:
   #quadratic
   elemType1 = mesh.ElemType(elemCode=C3D20R, elemLibrary=STANDARD,
                                      elemDeletion=ON)
   elemType2 = mesh.ElemType(elemCode=C3D15, elemLibrary=STANDARD,
                                      elemDeletion=ON)
   elemType3 = mesh.ElemType(elemCode=C3D10, elemLibrary=STANDARD,
                                      elemDeletion=ON)
  else:
   #linear
   elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=EXPLICIT,
        kinematicSplit=AVERAGE_STRAIN, secondOrderAccuracy=OFF,
        hourglassControl=DEFAULT, distortionControl=DEFAULT,
                                          elemDeletion=ON)
   elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT,
                                      elemDeletion=ON)
   elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT,
                                      elemDeletion=ON)

  return p,r1,(elemType1, elemType2, elemType3)

def compression_geometry(param_comp, mat_EVA):
  ''' Definicao da geometria do ensaio de tracao'''

  #Chama o sketch para ter uma base
  s1=sketch_compressao(param_comp)

  #Assimila uma geometria ao sketch
  #Parte Superior
  p = mdb.models['Model-1'].Part(name=param_comp['name'], dimensionality
                                      =THREE_D, type=DEFORMABLE_BODY)
  p = mdb.models['Model-1'].parts[param_comp['name']]
  p.BaseSolidExtrude(sketch=s1, depth=param_comp['dim'][1])
  s1.unsetPrimaryObject()
  p = mdb.models['Model-1'].parts[param_comp['name']]
  session.viewports['Viewport: 1'].setValues(displayedObject=p)
  del mdb.models['Model-1'].sketches['__profile__1']
  #Criacao da secao homogenea
```

```
mdb.models['Model-1'].HomogeneousSolidSection(name=param_comp['name'],
                                    material=mat_EVA['name'], thickness
                                   =None)
p = mdb.models['Model-1'].parts[param_comp['name']]
f = p.cells.getByBoundingBox()
region1 = regionToolset.Region(cells=f)
p.SectionAssignment(region=region1, sectionName=param_comp['name'],
                                   offset=0.0,
  offsetType=MIDDLE_SURFACE, offsetField='', thicknessAssignment=
                                   FROM_SECTION)

#Criacao de superficies uteis

p = mdb.models['Model-1'].parts[param_comp['name']]
s = p.faces
side1Faces = s.getByBoundingBox(zMax = 0.0)
p.Surface(side1Faces=side1Faces, name='Superficie_Superior')

p = mdb.models['Model-1'].parts[param_comp['name']]
s = p.faces
side1Faces = s.getByBoundingBox(zMin = param_comp['dim'][1])
p.Surface(side1Faces=side1Faces, name='Superficie_Inferior')

#Criacao do plano para a particao:
p = mdb.models['Model-1'].parts['compressao']
f = p.faces
p.DatumPlaneByOffset(plane=f[2], flip=SIDE2, offset=param_comp['dim'][
                                   1]/2.)

#Criacao da particao
p = mdb.models['Model-1'].parts['compressao']
c = p.cells
pickedCells = c.getByBoundingBox()
d = p.datums
p.PartitionCellByDatumPlane(datumPlane=d[5], cells=pickedCells)

#Criacao do ponto de referencia para convergencia de malha
p = mdb.models['Model-1'].parts['compressao']
v1, e, d1, n = p.vertices, p.edges, p.datums, p.nodes
p.ReferencePoint(point=p.InterestingPoint(edge=e[0], rule=CENTER))
p = mdb.models['Model-1'].parts['compressao']
r = p.referencePoints
refPoints=(r[7], )
p.Set(referencePoints=refPoints, name='Noued_Analysis')

#parametros de malhagem
r1 = p.cells.getByBoundingBox()
```

```
  if param_comp['quad']:
   #quadratic
   elemType1 = mesh.ElemType(elemCode=C3D20RH, elemLibrary=STANDARD,
                                    elemDeletion=ON)
   elemType2 = mesh.ElemType(elemCode=C3D15H, elemLibrary=STANDARD,
                                    elemDeletion=ON)
   elemType3 = mesh.ElemType(elemCode=C3D10H, elemLibrary=STANDARD,
                                    elemDeletion=ON)
  else:
   #linear
   elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=EXPLICIT,
        kinematicSplit=AVERAGE_STRAIN, secondOrderAccuracy=OFF,
        hourglassControl=DEFAULT, distortionControl=DEFAULT,
                                        elemDeletion=ON)
   elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT,
                                    elemDeletion=ON)
   elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT,
                                    elemDeletion=ON)

  return p,r1,(elemType1, elemType2, elemType3)


def bone_geometry(param_osso):
    ''' Definicao da geometria do osso'''

    #Chama o sketch para ter uma base
    s1=sketch_osso(param_osso)

    #Assimila uma geometria ao sketch
    #Parte Superior
    p = mdb.models['Model-1'].Part(name=param_osso['name'],
                                    dimensionality=THREE_D, type=
                                    DEFORMABLE_BODY)
    p = mdb.models['Model-1'].parts[param_osso['name']]
    p.BaseSolidExtrude(sketch=s1, depth=param_osso['Altura'])
    s1.unsetPrimaryObject()
    p = mdb.models['Model-1'].parts[param_osso['name']]
    session.viewports['Viewport: 1'].setValues(displayedObject=p)
    del mdb.models['Model-1'].sketches['__profile__1']
    #Criacao da secao homogenea
    mdb.models['Model-1'].HomogeneousSolidSection(name=param_osso['name'
                                    ], material=param_osso['Mat_Name
                                    '], thickness=None)
    p = mdb.models['Model-1'].parts[param_osso['name']]
    f = p.cells.getByBoundingBox()
    region1 = regionToolset.Region(cells=f)
```

```
p.SectionAssignment(region=region1, sectionName=param_osso['name'],
                                        offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='', thicknessAssignment=
                                            FROM_SECTION)

#Criacao de superficies uteis

p = mdb.models['Model-1'].parts[param_osso['name']]
s = p.faces
side1Faces = s.getByBoundingBox(zMax = 0.0)
p.Surface(side1Faces=side1Faces, name='Superficie_Superior')

p = mdb.models['Model-1'].parts[param_osso['name']]
s = p.faces
side1Faces = s.getByBoundingBox(zMin = param_osso['Altura'])
p.Surface(side1Faces=side1Faces, name='Superficie_Inferior')

#Criacao do plano para a particao:
p = mdb.models['Model-1'].parts[param_osso['name']]
f = p.faces
p.DatumPlaneByOffset(plane=f[2], flip=SIDE2, offset=param_osso['
                                        Altura']/2.)

#Criacao da particao
p = mdb.models['Model-1'].parts[param_osso['name']]
c = p.cells
pickedCells = c.getByBoundingBox()
d = p.datums
p.PartitionCellByDatumPlane(datumPlane=d[5], cells=pickedCells)

#Criacao do ponto de referencia para convergencia de malha
p = mdb.models['Model-1'].parts[param_osso['name']]
v1, e, d1, n = p.vertices, p.edges, p.datums, p.nodes
p.ReferencePoint(point=p.InterestingPoint(edge=e[0], rule=CENTER))
p = mdb.models['Model-1'].parts[param_osso['name']]
r = p.referencePoints
refPoints=(r[7], )
p.Set(referencePoints=refPoints, name='Noued_Analysis')

#parametros de malhagem
r1 = p.cells.getByBoundingBox()
if param_osso['quad']:
 #quadratic
 elemType1 = mesh.ElemType(elemCode=C3D20RH, elemLibrary=STANDARD,
                                        elemDeletion=ON)
 elemType2 = mesh.ElemType(elemCode=C3D15H, elemLibrary=STANDARD,
                                        elemDeletion=ON)
```

```
     elemType3 = mesh.ElemType(elemCode=C3D10H, elemLibrary=STANDARD,
                                          elemDeletion=ON)
    else:
     #linear
     elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=EXPLICIT,
            kinematicSplit=AVERAGE_STRAIN, secondOrderAccuracy=OFF,
            hourglassControl=DEFAULT, distortionControl=DEFAULT,
                                                 elemDeletion=ON)
     elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT,
                                          elemDeletion=ON)
     elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT,
                                          elemDeletion=ON)

    return p,r1,(elemType1, elemType2, elemType3)


def pele_geometry(param_pele):
    ''' Definicao da geometria da pele'''

    #Chama o sketch para ter uma base
    s1=sketch_pele(param_pele)

    #Assimila uma geometria ao sketch
    #Parte Superior
    p = mdb.models['Model-1'].Part(name=param_pele['name'],
                                        dimensionality=THREE_D, type=
                                        DEFORMABLE_BODY)
    p = mdb.models['Model-1'].parts[param_pele['name']]
    p.BaseSolidExtrude(sketch=s1, depth=param_pele['Altura'])
    s1.unsetPrimaryObject()
    p = mdb.models['Model-1'].parts[param_pele['name']]
    session.viewports['Viewport: 1'].setValues(displayedObject=p)
    del mdb.models['Model-1'].sketches['__profile__1']
    #Criacao da secao homogenea
    mdb.models['Model-1'].HomogeneousSolidSection(name=param_pele['name'
                                        ], material=param_pele['Mat_Name
                                        '], thickness=None)
    p = mdb.models['Model-1'].parts[param_pele['name']]
    f = p.cells.getByBoundingBox()
    region1 = regionToolset.Region(cells=f)
    p.SectionAssignment(region=region1, sectionName=param_pele['name'],
                                        offset=0.0,
        offsetType=MIDDLE_SURFACE, offsetField='', thicknessAssignment=
                                            FROM_SECTION)

    #Criacao de superficies uteis
```

```
p = mdb.models['Model-1'].parts[param_pele['name']]
s = p.faces
side1Faces = s.getByBoundingBox(zMax = 0.0)
p.Surface(side1Faces=side1Faces, name='Superficie_Superior')

p = mdb.models['Model-1'].parts[param_pele['name']]
s = p.faces
side1Faces = s.getByBoundingBox(zMin = param_pele['Altura'])
p.Surface(side1Faces=side1Faces, name='Superficie_Inferior')

#Criacao do plano para a particao:
p = mdb.models['Model-1'].parts[param_pele['name']]
f = p.faces
p.DatumPlaneByOffset(plane=f[2], flip=SIDE2, offset=param_pele['
                                    Altura']/2.)

#Criacao da particao
p = mdb.models['Model-1'].parts[param_pele['name']]
c = p.cells
pickedCells = c.getByBoundingBox()
d = p.datums
p.PartitionCellByDatumPlane(datumPlane=d[5], cells=pickedCells)

#Criacao do ponto de referencia para convergencia de malha
p = mdb.models['Model-1'].parts[param_pele['name']]
v1, e, d1, n = p.vertices, p.edges, p.datums, p.nodes
p.ReferencePoint(point=p.InterestingPoint(edge=e[0], rule=CENTER))
p = mdb.models['Model-1'].parts[param_pele['name']]
r = p.referencePoints
refPoints=(r[7], )
p.Set(referencePoints=refPoints, name='Noued_Analysis')

#parametros de malhagem
r1 = p.cells.getByBoundingBox()
if param_pele['quad']:
 #quadratic
 elemType1 = mesh.ElemType(elemCode=C3D20RH, elemLibrary=STANDARD,
                                    elemDeletion=ON)
 elemType2 = mesh.ElemType(elemCode=C3D15H, elemLibrary=STANDARD,
                                    elemDeletion=ON)
 elemType3 = mesh.ElemType(elemCode=C3D10H, elemLibrary=STANDARD,
                                    elemDeletion=ON)
else:
 #linear
 elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=EXPLICIT,
        kinematicSplit=AVERAGE_STRAIN, secondOrderAccuracy=OFF,
```

```
            hourglassControl=DEFAULT, distortionControl=DEFAULT,
                                                  elemDeletion=ON)
     elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT,
                                           elemDeletion=ON)
     elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT,
                                           elemDeletion=ON)

    return p,r1,(elemType1, elemType2, elemType3)

def FLEX_geometry(param_pele):
    ''' Definicao da geometria da pele'''

    #Chama o sketch para ter uma base
    s1=sketch_pele(param_pele)

    #Assimila uma geometria ao sketch
    #Parte Superior
    p = mdb.models['Model-1'].Part(name=param_pele['name'],
                                   dimensionality=THREE_D, type=
                                   DEFORMABLE_BODY)
    p = mdb.models['Model-1'].parts[param_pele['name']]
    p.BaseSolidExtrude(sketch=s1, depth=param_pele['Altura'])
    s1.unsetPrimaryObject()
    p = mdb.models['Model-1'].parts[param_pele['name']]
    session.viewports['Viewport: 1'].setValues(displayedObject=p)
    del mdb.models['Model-1'].sketches['__profile__1']
    #Criacao da secao homogenea
    mdb.models['Model-1'].HomogeneousSolidSection(name=param_pele['name'
                                   ], material=param_pele['Mat_Name
                                   '], thickness=None)
    p = mdb.models['Model-1'].parts[param_pele['name']]
    f = p.cells.getByBoundingBox()
    region1 = regionToolset.Region(cells=f)
    p.SectionAssignment(region=region1, sectionName=param_pele['name'],
                                   offset=0.0,
        offsetType=MIDDLE_SURFACE, offsetField='', thicknessAssignment=
                                       FROM_SECTION)

    #Criacao de superficies uteis

    p = mdb.models['Model-1'].parts[param_pele['name']]
    s = p.faces
    side1Faces = s.getByBoundingBox(zMax = 0.0)
    p.Surface(side1Faces=side1Faces, name='Superficie_Superior')

    p = mdb.models['Model-1'].parts[param_pele['name']]
    s = p.faces
```

```
side1Faces = s.getByBoundingBox(zMin = param_pele['Altura'])
p.Surface(side1Faces=side1Faces, name='Superficie_Inferior')

#Criacao do plano para a particao:
p = mdb.models['Model-1'].parts[param_pele['name']]
f = p.faces
p.DatumPlaneByOffset(plane=f[2], flip=SIDE2, offset=param_pele['
                                    Altura']/2.)

#Criacao da particao
p = mdb.models['Model-1'].parts[param_pele['name']]
c = p.cells
pickedCells = c.getByBoundingBox()
d = p.datums
p.PartitionCellByDatumPlane(datumPlane=d[5], cells=pickedCells)

#Criacao do ponto de referencia para convergencia de malha
p = mdb.models['Model-1'].parts[param_pele['name']]
v1, e, d1, n = p.vertices, p.edges, p.datums, p.nodes
p.ReferencePoint(point=p.InterestingPoint(edge=e[0], rule=CENTER))
p = mdb.models['Model-1'].parts[param_pele['name']]
r = p.referencePoints
refPoints=(r[7], )
p.Set(referencePoints=refPoints, name='Noued_Analysis')

#parametros de malhagem
r1 = p.cells.getByBoundingBox()
if param_FLEX['quad']:
 #quadratic
 elemType1 = mesh.ElemType(elemCode=C3D20RH, elemLibrary=STANDARD,
                                    elemDeletion=ON)
 elemType2 = mesh.ElemType(elemCode=C3D15H, elemLibrary=STANDARD,
                                    elemDeletion=ON)
 elemType3 = mesh.ElemType(elemCode=C3D10H, elemLibrary=STANDARD,
                                    elemDeletion=ON)
else:
 #linear
 elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=EXPLICIT,
        kinematicSplit=AVERAGE_STRAIN, secondOrderAccuracy=OFF,
        hourglassControl=DEFAULT, distortionControl=DEFAULT,
                                        elemDeletion=ON)
 elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT,
                                    elemDeletion=ON)
 elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT,
                                    elemDeletion=ON)

return p,r1,(elemType1, elemType2, elemType3)
```

```
def RIG_geometry(param_pele):
    ''' Definicao da geometria da pele'''

    #Chama o sketch para ter uma base
    s1=sketch_pele(param_pele)

    #Assimila uma geometria ao sketch
    #Parte Superior
    p = mdb.models['Model-1'].Part(name=param_pele['name'],
                                   dimensionality=THREE_D, type=
                                   DEFORMABLE_BODY)
    p = mdb.models['Model-1'].parts[param_pele['name']]
    p.BaseSolidExtrude(sketch=s1, depth=param_pele['Altura'])
    s1.unsetPrimaryObject()
    p = mdb.models['Model-1'].parts[param_pele['name']]
    session.viewports['Viewport: 1'].setValues(displayedObject=p)
    del mdb.models['Model-1'].sketches['__profile__1']
    #Criacao da secao homogenea
    mdb.models['Model-1'].HomogeneousSolidSection(name=param_pele['name'
                                   ], material=param_pele['Mat_Name
                                   '], thickness=None)
    p = mdb.models['Model-1'].parts[param_pele['name']]
    f = p.cells.getByBoundingBox()
    region1 = regionToolset.Region(cells=f)
    p.SectionAssignment(region=region1, sectionName=param_pele['name'],
                                   offset=0.0,
        offsetType=MIDDLE_SURFACE, offsetField='', thicknessAssignment=
                                        FROM_SECTION)

    #Criacao de superficies uteis

    p = mdb.models['Model-1'].parts[param_pele['name']]
    s = p.faces
    side1Faces = s.getByBoundingBox(zMax = 0.0)
    p.Surface(side1Faces=side1Faces, name='Superficie_Superior')

    p = mdb.models['Model-1'].parts[param_pele['name']]
    s = p.faces
    side1Faces = s.getByBoundingBox(zMin = param_pele['Altura'])
    p.Surface(side1Faces=side1Faces, name='Superficie_Inferior')

    #Criacao do plano para a particao:
    p = mdb.models['Model-1'].parts[param_pele['name']]
    f = p.faces
    p.DatumPlaneByOffset(plane=f[2], flip=SIDE2, offset=param_pele['
                                   Altura']/2.)
```

```
    #Criacao da particao
    p = mdb.models['Model-1'].parts[param_pele['name']]
    c = p.cells
    pickedCells = c.getByBoundingBox()
    d = p.datums
    p.PartitionCellByDatumPlane(datumPlane=d[5], cells=pickedCells)

    #Criacao do ponto de referencia para convergencia de malha
    p = mdb.models['Model-1'].parts[param_pele['name']]
    v1, e, d1, n = p.vertices, p.edges, p.datums, p.nodes
    p.ReferencePoint(point=p.InterestingPoint(edge=e[0], rule=CENTER))
    p = mdb.models['Model-1'].parts[param_pele['name']]
    r = p.referencePoints
    refPoints=(r[7], )
    p.Set(referencePoints=refPoints, name='Noued_Analysis')

    #parametros de malhagem
    r1 = p.cells.getByBoundingBox()
    if param_RIG['quad']:
     #quadratic
     elemType1 = mesh.ElemType(elemCode=C3D20RH, elemLibrary=STANDARD,
                                          elemDeletion=ON)
     elemType2 = mesh.ElemType(elemCode=C3D15H, elemLibrary=STANDARD,
                                          elemDeletion=ON)
     elemType3 = mesh.ElemType(elemCode=C3D10H, elemLibrary=STANDARD,
                                          elemDeletion=ON)
    else:
     #linear
     elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=EXPLICIT,
            kinematicSplit=AVERAGE_STRAIN, secondOrderAccuracy=OFF,
            hourglassControl=DEFAULT, distortionControl=DEFAULT,
                                                elemDeletion=ON)
     elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT,
                                          elemDeletion=ON)
     elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT,
                                          elemDeletion=ON)

    return p,r1,(elemType1, elemType2, elemType3)

def Impact_Ball(param_bola, namec):
    ''' Cria a bola de impacto '''
    s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__1',
    sheetSize=200.0)
    g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
    s.setPrimaryObject(option=STANDALONE)
```

```
    point1 = (0.0,param_bola['Diametro']/2.)
    point2 = (0.0,-param_bola['Diametro']/2.)
    point3 = (param_bola['Diametro']/2.,0.0)

    s.ConstructionLine(point1=(0.0, -100.0), point2=(0.0, 100.0))
    s.Arc3Points(point1=point1, point2=point2, point3=point3)
    s.Line(point1=point1, point2= point2)

    p = mdb.models['Model-1'].Part(name=namec, dimensionality=THREE_D,
        type=DISCRETE_RIGID_SURFACE)
    p = mdb.models['Model-1'].parts[namec]
    p.BaseShellRevolve(sketch=s, angle=360.0, flipRevolveDirection=OFF)
    s.unsetPrimaryObject()
    p = mdb.models['Model-1'].parts[namec]
    session.viewports['Viewport: 1'].setValues(displayedObject=p)
    del mdb.models['Model-1'].sketches['__profile__1']


    #Criacao de um reference point no centro
    p = mdb.models['Model-1'].parts[namec]
    p.DatumPointByCoordinate(coords=(0.0, 0.0, 0.0))
    p = mdb.models['Model-1'].parts[namec]
    p.ReferencePoint(point=(0.0, 0.0, 0.0))

    #Criacao de sets uteis
    p.Surface(side1Faces=p.faces.getByBoundingBox(), name=namec)
    p.Set(edges=p.edges.getByBoundingBox(), name=namec)

    #mesh
    elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=STANDARD)
    elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=STANDARD)
    p.setElementType(regions=(p.surfaces[namec].faces,), elemTypes=(
                                        elemType1, elemType2 ))
    p.seedPart(size=simu['global_extremidades']/2., deviationFactor=0.1,
                                         minSizeFactor=0.1)
    p.generateMesh()

    return p

def Support_Impact(param_comp, namec):
    ''' Cria os suportes de compressao '''
    s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
    g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
    s.setPrimaryObject(option=STANDALONE)
```

```
    s.CircleByCenterPerimeter(center=(0.0, 0.0), point1=(0.0, param_comp
                                        ['dia_support']/2.))
    p = mdb.models['Model-1'].Part(name=namec, dimensionality=THREE_D,
        type=DISCRETE_RIGID_SURFACE)
    p = mdb.models['Model-1'].parts[namec]
    p.BaseShell(sketch=s)
    s.unsetPrimaryObject()
    p = mdb.models['Model-1'].parts[namec]
    session.viewports['Viewport: 1'].setValues(displayedObject=p)
    del mdb.models['Model-1'].sketches['__profile__']
    #Criacao de um reference point no centro
    p = mdb.models['Model-1'].parts[namec]
    p.DatumPointByCoordinate(coords=(0.0, 0.0, 0.0))
    p = mdb.models['Model-1'].parts[namec]
    p.ReferencePoint(point=(0.0, 0.0, 0.0))

    #Criacao de sets uteis
    p.Surface(side1Faces=p.faces.getByBoundingBox(), name=namec)
    p.Set(edges=p.edges.getByBoundingBox(), name=namec)

    #mesh
    elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=STANDARD)
    elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=STANDARD)
    p.setElementType(regions=(p.surfaces[namec].faces,), elemTypes=(
                                        elemType1, elemType2 ))
    p.seedPart(size=simu['global_extremidades'], deviationFactor=0.1,
                                        minSizeFactor=0.1)
    p.generateMesh()

    return p



##########################################################
# Geometry
##########################################################
if Mod==1:
  p,r1,elements = traction_geometry(param_trac,mat)

if Mod==2:
  p,r1,elements = compression_geometry(param_comp,mat_EVA)
  sup = Supports_Compression(param_comp, 'Superior')
  inf = Supports_Compression(param_comp, 'Inferior')
if Mod == 3:
    if Camadas == 'Osso':
        p1,r1,elements1 = bone_geometry(param_osso)
```

```
        impact_ball = Impact_Ball(param_bola,'Esfera')
        Suport = Support_Impact(param_impacto, param_impacto['name'])
    if Camadas == 'F':
        p1,r1,elements1 = bone_geometry(param_osso)
        p2,r2,elements2 = pele_geometry(param_pele)
        p3,r3,elements3 = FLEX_geometry(param_FLEX)
        impact_ball = Impact_Ball(param_bola,'Esfera')
        Suport = Support_Impact(param_impacto, param_impacto['name'])
    if Camadas == 'sEVA':
        p1,r1,elements1 = bone_geometry(param_osso)
        p2,r2,elements2 = pele_geometry(param_pele)
        impact_ball = Impact_Ball(param_bola,'Esfera')
        Suport = Support_Impact(param_impacto, param_impacto['name'])
    if Camadas == 'R':
        p1,r1,elements1 = bone_geometry(param_osso)
        p2,r2,elements2 = pele_geometry(param_pele)
        p3,r3,elements3 = RIG_geometry(param_RIG)
        impact_ball = Impact_Ball(param_bola,'Esfera')
        Suport = Support_Impact(param_impacto, param_impacto['name'])
    if Camadas == 'FR':
        p1,r1,elements1 = bone_geometry(param_osso)
        p2,r2,elements2 = pele_geometry(param_pele)
        p3,r3,elements3 = RIG_geometry(param_RIG)
        p4,r4,elements4 = FLEX_geometry(param_FLEX)
        impact_ball = Impact_Ball(param_bola,'Esfera')
        Suport = Support_Impact(param_impacto, param_impacto['name'])
    if Camadas == 'RFR':
        p1,r1,elements1 = bone_geometry(param_osso)
        p2,r2,elements2 = pele_geometry(param_pele)
        p3,r3,elements3 = RIG_geometry(param_RIG)
        mdb.models['Model-1'].parts.changeKey(fromName='RIG', toName='
                                              RIG2')
        p4,r4,elements4 = FLEX_geometry(param_FLEX)
        p5,r5,elements5 = RIG_geometry(param_RIG)
        impact_ball = Impact_Ball(param_bola,'Esfera')
        Suport = Support_Impact(param_impacto, param_impacto['name'])
    if Camadas == 'FRF':
        p1,r1,elements1 = bone_geometry(param_osso)
        p2,r2,elements2 = pele_geometry(param_pele)
        p3,r3,elements3 = RIG_geometry(param_RIG)
        p4,r4,elements4 = FLEX_geometry(param_FLEX)
        mdb.models['Model-1'].parts.changeKey(fromName='FLEX', toName='
                                              FLEX2')
        p5,r5,elements5 = FLEX_geometry(param_FLEX)
        impact_ball = Impact_Ball(param_bola,'Esfera')
        Suport = Support_Impact(param_impacto, param_impacto['name'])
```

```
#########################################################
# Material - steel and EVA
#########################################################


if Mod == 1:
  mdb.models['Model-1'].Material(name=mat['name'])
  mdb.models['Model-1'].materials[mat['name']].Density(table=((mat['dens
                                    '], ), ))
  mdb.models['Model-1'].materials[mat['name']].Elastic(table=((mat['E'],
                                     mat['nu']), ))
  mdb.models['Model-1'].materials[mat['name']].Plastic(table=((mat['Re']
                                    , 0.0), ))
  mdb.models['Model-1'].materials['steel'].plastic.RateDependent(table=
                                    ((mat['K'], mat['n']), ))
  mdb.models['Model-1'].materials['steel'].DuctileDamageInitiation(table
                                    =((0.1, 0.0, 0.0), ))
  mdb.models['Model-1'].materials['steel'].ductileDamageInitiation.
                                    DamageEvolution(
      type=DISPLACEMENT, table=((0.0012, ), ))
  mdb.models['Model-1'].materials['steel'].plastic.setValues(table=((
                                    310000000.0,
      0.0), (315000000.0, 0.0035), (325000000.0, 0.0086), (335000000.0,
                                    0.0163),
      (350000000.0, 0.028), (365000000.0, 0.045), (385000000.0, 0.07), (
      400000000.0, 0.1), (418000000.0, 0.17), (430000000.0, 0.25)))
if Mod == 2:
  if MAT == 1:
    mdb.models['Model-1'].Material(name=mat_EVA['name'])
    mdb.models['Model-1'].materials[mat_EVA['name']].Density(table=((
                                    mat_EVA['dens'], ), ))
    mdb.models['Model-1'].materials[mat_EVA['name']].Hyperelastic(
                                    materialType=ISOTROPIC, testData=OFF
                                    , type=ARRUDA_BOYCE,
                                    volumetricResponse=VOLUMETRIC_DATA,
                                    table=((mat_EVA['Arruda_Boyce'][0],
                                    mat_EVA['Arruda_Boyce'][1], mat_EVA[
                                    'Arruda_Boyce'][2]), ))
    mdb.models['Model-1'].materials[mat_EVA['name']].Viscoelastic(domain
                                    =TIME, time=PRONY, table=((mat_EVA['
                                    g_i'][0],mat_EVA['k_i'][0], mat_EVA[
                                    'tau'][0]), ))
```

```
  if MAT == 2:
    mdb.models['Model-1'].Material(name=mat_EVA['name'])
    mdb.models['Model-1'].materials[mat_EVA['name']].Density(table=((
                                    mat_EVA['dens'], ), ))
    mdb.models['Model-1'].materials['EVA_Rigid'].Elastic(table=((mat_EVA
                                    ['E'], mat_EVA['nu']), ))

if Mod == 3:
    #Caracteristicas do Osso
    mdb.models['Model-1'].Material(name=param_osso['Mat_Name'])
    mdb.models['Model-1'].materials[param_osso['Mat_Name']].Density(
                                        table=((param_osso['dens'], ), )
                                        )
    mdb.models['Model-1'].materials[param_osso['Mat_Name']].Elastic(
                                        table=((param_osso['E'],
                                        param_osso['nu']), ))
    mdb.models['Model-1'].materials[param_osso['Mat_Name']].Viscoelastic
                                        (domain=TIME, time=PRONY, table=
                                        ((param_osso['g_i'][0],
                                        param_pele['k_i'][0], param_pele
                                        ['tau'][0]), ))

    #Caracteristicas da Pele
    mdb.models['Model-1'].Material(name=param_pele['Mat_Name'])
    mdb.models['Model-1'].materials[param_pele['Mat_Name']].Density(
                                        table=((param_pele['dens'], ), )
                                        )
    mdb.models['Model-1'].materials[param_pele['Mat_Name']].Elastic(
                                        table=((param_pele['E'],
                                        param_pele['nu']), ))
    mdb.models['Model-1'].materials[param_pele['Mat_Name']].Viscoelastic
                                        (domain=TIME, time=PRONY, table=
                                        ((param_pele['g_i'][0],
                                        param_pele['k_i'][0], param_pele
                                        ['tau'][0]), (param_pele['g_i'][
                                        1], param_pele['k_i'][1],
                                        param_pele['tau'][1]), (
                                        param_pele['g_i'][2], param_pele
                                        ['k_i'][2], param_pele['tau'][2]
                                        ), (param_pele['g_i'][3],
                                        param_pele['k_i'][3], param_pele
                                        ['tau'][3]) ))


    #Caracteristicas do EVA Flexivel
    mdb.models['Model-1'].Material(name=mat_EVA_FLEX['name'])
```

```
    mdb.models['Model-1'].materials[mat_EVA_FLEX['name']].Density(table=
                                        ((mat_EVA_FLEX['dens'], ), ))
    mdb.models['Model-1'].materials[mat_EVA_FLEX['name']].Hyperelastic(
                                        materialType=ISOTROPIC, testData
                                        =OFF, type=ARRUDA_BOYCE,
                                        volumetricResponse=
                                        VOLUMETRIC_DATA, table=((
                                        mat_EVA_FLEX['Arruda_Boyce'][0],
                                         mat_EVA_FLEX['Arruda_Boyce'][1]
                                        , mat_EVA_FLEX['Arruda_Boyce'][2
                                        ]), ))
    mdb.models['Model-1'].materials[mat_EVA_FLEX['name']].Viscoelastic(
                                        domain=TIME, time=PRONY, table=
                                        ((mat_EVA_FLEX['g_i'][0],
                                        mat_EVA_FLEX['k_i'][0],
                                        mat_EVA_FLEX['tau'][0]), (
                                        mat_EVA_FLEX['g_i'][1],
                                        mat_EVA_FLEX['k_i'][1],
                                        mat_EVA_FLEX['tau'][1]), (
                                        mat_EVA_FLEX['g_i'][2],
                                        mat_EVA_FLEX['k_i'][2],
                                        mat_EVA_FLEX['tau'][2]), (
                                        mat_EVA_FLEX['g_i'][3],
                                        mat_EVA_FLEX['k_i'][3],
                                        mat_EVA_FLEX['tau'][3]), ))

    #Caracteristicas do eva Rigido
    mdb.models['Model-1'].Material(name=mat_EVA_RIG['name'])
    mdb.models['Model-1'].materials[mat_EVA_RIG['name']].Density(table=
                                        ((mat_EVA_RIG['dens'], ), ))
    mdb.models['Model-1'].materials[mat_EVA_RIG['name']].Elastic(table=
                                        ((mat_EVA_RIG['E'], mat_EVA['nu'
                                        ]), ))




##########################################################
# Assembly
##########################################################
if Mod == 1:
  #Import de todas as tres partes
  a = mdb.models['Model-1'].rootAssembly
  a.DatumCsysByDefault(CARTESIAN)
  p = mdb.models['Model-1'].parts[param_trac['name_Sup']]
```

```
a.Instance(name='Assembly_Superior', part=p, dependent=ON)
p = mdb.models['Model-1'].parts[param_trac['name_Inf']]
a.Instance(name='Assembly_Inferior', part=p, dependent=ON)
p = mdb.models['Model-1'].parts[param_trac['name_Meio']]
a.Instance(name='Assembly_Meio', part=p, dependent=ON)
session.viewports['Viewport: 1'].view.fitView()
#Creacao de Sets uteis
f1 = a.instances['Assembly_Superior'].faces
faces1 = f1.getByBoundingBox(yMin = param_trac['dim'][2]/2.)
region = a.Set(faces=faces1, name='Face_Sup')

f1 = a.instances['Assembly_Inferior'].faces
faces1 = f1.getByBoundingBox(yMax = -param_trac['dim'][2]/2.)
region = a.Set(faces=faces1, name='Face_Inf')

f1 = a.instances['Assembly_Inferior'].faces
faces1 = f1.getByBoundingBox(yMax = param_trac['dim'][3] - param_trac[
                                    'dim'][2]/2. + 0.001)
region = a.Set(faces=faces1, name='Head_Inf')

f1 = a.instances['Assembly_Superior'].faces
faces1 = f1.getByBoundingBox(yMin = (param_trac['dim'][2]/2. -
                                    param_trac['dim'][3]))
region = a.Set(faces=faces1, name='Head_Sup')

#Criacao de superficies  teis
#Superficie inferiora da parte Superior
a = mdb.models['Model-1'].rootAssembly
s1 = a.instances['Assembly_Superior'].faces
side1Faces1 = s1.getByBoundingBox(yMax = param_trac['dim'][2]/2. -
                                    param_trac['dim'][3] - param_trac['
                                    dim'][4] - param_trac['prolongacao']
                                    )
region1=a.Surface(side1Faces=side1Faces1, name='Face_Inf-
                                    Parte_Superior')
#Superficie Superiora da zona do meio
a = mdb.models['Model-1'].rootAssembly
s1 = a.instances['Assembly_Meio'].faces
side1Faces1 = s1.getByBoundingBox(yMin = param_trac['dim'][2]/2. -
                                    param_trac['dim'][3] - param_trac['
                                    dim'][4] - param_trac['prolongacao']
                                    )
region1=a.Surface(side1Faces=side1Faces1, name='Face_Sup-Parte_Meio')
#Superficie Inferior da parte do meio
a = mdb.models['Model-1'].rootAssembly
s1 = a.instances['Assembly_Meio'].faces
```

```
  side1Faces1 = s1.getByBoundingBox(yMax = -(param_trac['dim'][2]/2. -
                                    param_trac['dim'][3] - param_trac['
                                    dim'][4] - param_trac['prolongacao']
                                    ))
  region1=a.Surface(side1Faces=side1Faces1, name='Face_Inf-Parte_Meio')
  #Superficie superiora da pare Inferior
  a = mdb.models['Model-1'].rootAssembly
  s1 = a.instances['Assembly_Inferior'].faces
  side1Faces1 = s1.getByBoundingBox(yMin = -(param_trac['dim'][2]/2. -
                                    param_trac['dim'][3] - param_trac['
                                    dim'][4] - param_trac['prolongacao']
                                    ))
  region1=a.Surface(side1Faces=side1Faces1, name='Face_Sup-Parte_Inf')


if Mod == 2:
  #Import de todas as tres partes
  a = mdb.models['Model-1'].rootAssembly
  a.DatumCsysByDefault(CARTESIAN)
  p = mdb.models['Model-1'].parts[param_comp['name']]
  a.Instance(name='Assembly_Corpo_de_Prova', part=p, dependent=ON)
  p = mdb.models['Model-1'].parts['Superior']
  a.Instance(name='Assembly_Superior', part=p, dependent=ON)
  p = mdb.models['Model-1'].parts['Inferior']
  a.Instance(name='Assembly_Inferior', part=p, dependent=ON)
  session.viewports['Viewport: 1'].view.fitView()

  #Posicionamento das pecas
  a.translate(instanceList=('Assembly_Inferior', ), vector=(0.0, 0.0,
                                    param_comp['dim'][1]))

  #Criacao de Sets uteis
  #Reference point do suporte de cima
  a = mdb.models['Model-1'].rootAssembly
  r1 = a.instances['Assembly_Superior'].referencePoints
  refPoints1=(r1[3], )
  a.Set(referencePoints=refPoints1, name='Reference_Point_Superior')
  region=a.sets['Reference_Point_Superior']
  mdb.models['Model-1'].parts['Superior'].engineeringFeatures.
                                    PointMassInertia(
      name='Inertia_Sup', region=region, mass=1.0, i11=1.0, i22=1.0, i33
                                        =1.0, alpha=0.0, composite=0.0)
  #Reference Point Inferior
  a = mdb.models['Model-1'].rootAssembly
  r1 = a.instances['Assembly_Inferior'].referencePoints
  refPoints1=(r1[3], )
  a.Set(referencePoints=refPoints1, name='Reference_Point_Inferior')
```

```
  region=a.sets['Reference_Point_Inferior']
  mdb.models['Model-1'].parts['Inferior'].engineeringFeatures.
                                    PointMassInertia(
      name='Inertia_Inf', region=region, mass=1.0, i11=1.0, i22=1.0, i33
                                        =1.0, alpha=0.0, composite=0.0)

if Mod == 3:

    if Camadas == 'F':
        #Import de todas as tres partes
        a = mdb.models['Model-1'].rootAssembly
        a.DatumCsysByDefault(CARTESIAN)
        p = mdb.models['Model-1'].parts[param_osso['name']]
        a.Instance(name=param_osso['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_pele['name']]
        a.Instance(name=param_pele['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_FLEX['name']]
        a.Instance(name=param_FLEX['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_bola['name']]
        a.Instance(name=param_bola['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_impacto['name']]
        a.Instance(name=param_impacto['name'], part=p, dependent=ON)
        session.viewports['Viewport: 1'].view.fitView()

        #Posicionamento das pecas
        a.translate(instanceList=(param_pele['name'], ), vector=(0.0, 0.
                                        0, -param_pele['Altura']))
        a.translate(instanceList=(param_FLEX['name'], ), vector=(0.0, 0.
                                        0, -param_pele['Altura'] -
                                        param_FLEX['Altura']))
        a.translate(instanceList=(param_bola['name'], ), vector=(0.0, 0.
                                        0, -param_pele['Altura'] -
                                        param_FLEX['Altura'] -
                                        param_bola['Diametro']/2. -
                                        0.01))
        a.translate(instanceList=(param_impacto['name'], ), vector=(0.0,
                                         0.0, param_osso['Altura']))

        #Criacao de Sets uteis
        #Criacao do RP do Suporte
        a = mdb.models['Model-1'].rootAssembly
        r1 = a.instances['Suporte'].referencePoints
        refPoints1=(r1[3], )
        a.Set(referencePoints=refPoints1, name='RP_Suporte')

        region=a.sets['RP_Suporte']
```

```
        mdb.models['Model-1'].parts[param_impacto['name']].
                                            engineeringFeatures.
                                            PointMassInertia(
            name='Inertia_Inf', region=region, mass=1.0, i11=1.0, i22=1.
                                                0, i33=1.0, alpha=0.0,
                                                composite=0.0)


        a = mdb.models['Model-1'].rootAssembly
        r1 = a.instances['Esfera'].referencePoints
        refPoints1=(r1[3], )
        a.Set(referencePoints=refPoints1, name='RF_Ball')

        p = mdb.models['Model-1'].parts['Esfera']
        r = p.referencePoints
        refPoints=(r[3], )
        p.Set(referencePoints=refPoints, name='RP_Ball')

        #Criacao da massa da esfera
        p = mdb.models['Model-1'].parts['Esfera']
        region=p.sets['RP_Ball']
        mdb.models['Model-1'].parts[param_bola['name']].
                                            engineeringFeatures.
                                            PointMassInertia(
            name='Massa_Bola', region=region, mass=param_bola['Massa'],
                                                i11=1.0, i22=1.0, i33=1.
                                                0, alpha=0.0, composite=
                                                0.0)

        #Criacao da superficie do suporte
        a = mdb.models['Model-1'].rootAssembly
        s1 = a.instances['Suporte'].faces
        side2Faces1 = s1.getByBoundingBox(zMin = param_osso['Altura'])
        a.Surface(side2Faces=side2Faces1, name='Superficie_Suporte')


    if Camadas == 'sEVA':
        #Import de todas as tres partes
        a = mdb.models['Model-1'].rootAssembly
        a.DatumCsysByDefault(CARTESIAN)
        p = mdb.models['Model-1'].parts[param_osso['name']]
        a.Instance(name=param_osso['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_pele['name']]
        a.Instance(name=param_pele['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_bola['name']]
        a.Instance(name=param_bola['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_impacto['name']]
```

```
a.Instance(name=param_impacto['name'], part=p, dependent=ON)
session.viewports['Viewport: 1'].view.fitView()

#Posicionamento das pecas
a.translate(instanceList=(param_pele['name'], ), vector=(0.0, 0.
                                    0, -param_pele['Altura']))
a.translate(instanceList=(param_bola['name'], ), vector=(0.0, 0.
                                    0, -param_pele['Altura'] -
                                    param_bola['Diametro']/2. -
                                    0.015))
a.translate(instanceList=(param_impacto['name'], ), vector=(0.0,
                                     0.0, param_osso['Altura']))

#Criacao de Sets uteis
#Criacao do RP do Suporte
a = mdb.models['Model-1'].rootAssembly
r1 = a.instances['Suporte'].referencePoints
refPoints1=(r1[3], )
a.Set(referencePoints=refPoints1, name='RP_Suporte')

region=a.sets['RP_Suporte']
mdb.models['Model-1'].parts[param_impacto['name']].
                                    engineeringFeatures.
                                    PointMassInertia(
    name='Inertia_Inf', region=region, mass=1.0, i11=1.0, i22=1.
                                        0, i33=1.0, alpha=0.0,
                                        composite=0.0)

a = mdb.models['Model-1'].rootAssembly
r1 = a.instances['Esfera'].referencePoints
refPoints1=(r1[3], )
a.Set(referencePoints=refPoints1, name='RF_Ball')

p = mdb.models['Model-1'].parts['Esfera']
r = p.referencePoints
refPoints=(r[3], )
p.Set(referencePoints=refPoints, name='RP_Ball')

#Criacao da massa da esfera
p = mdb.models['Model-1'].parts['Esfera']
region=p.sets['RP_Ball']
mdb.models['Model-1'].parts[param_bola['name']].
                                    engineeringFeatures.
                                    PointMassInertia(
    name='Massa_Bola', region=region, mass=param_bola['Massa'],
                                        i11=1.0, i22=1.0, i33=1.
                                        0, alpha=0.0, composite=
```

```
                                                0.0)
        #Criacao da superficie do suporte
        a = mdb.models['Model-1'].rootAssembly
        s1 = a.instances['Suporte'].faces
        side2Faces1 = s1.getByBoundingBox(zMin = param_osso['Altura'])
        a.Surface(side2Faces=side2Faces1, name='Superficie_Suporte')

    if Camadas == 'Osso':
        #Import de todas as tres partes
        a = mdb.models['Model-1'].rootAssembly
        a.DatumCsysByDefault(CARTESIAN)
        p = mdb.models['Model-1'].parts[param_osso['name']]
        a.Instance(name=param_osso['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_bola['name']]
        a.Instance(name=param_bola['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_impacto['name']]
        a.Instance(name=param_impacto['name'], part=p, dependent=ON)
        session.viewports['Viewport: 1'].view.fitView()

        #Posicionamento das pecas
        a.translate(instanceList=(param_bola['name'], ), vector=(0.0, 0.
                                            0, - param_bola['Diametro']/
                                            2. - 0.015))
        a.translate(instanceList=(param_impacto['name'], ), vector=(0.0,
                                             0.0, param_osso['Altura']))

        #Criacao de Sets uteis
        #Criacao do Set do Suporte
        a = mdb.models['Model-1'].rootAssembly
        r1 = a.instances['Suporte'].referencePoints
        refPoints1=(r1[3], )
        a.Set(referencePoints=refPoints1, name='RP_Suporte')

        region=a.sets['RP_Suporte']
        mdb.models['Model-1'].parts[param_impacto['name']].
                                            engineeringFeatures.
                                            PointMassInertia(
            name='Inertia_Inf', region=region, mass=1.0, i11=1.0, i22=1.
                                                0, i33=1.0, alpha=0.0,
                                                composite=0.0)

        a = mdb.models['Model-1'].rootAssembly
        r1 = a.instances['Esfera'].referencePoints
        refPoints1=(r1[3], )
        a.Set(referencePoints=refPoints1, name='RF_Ball')

        p = mdb.models['Model-1'].parts['Esfera']
```

```
        r = p.referencePoints
        refPoints=(r[3], )
        p.Set(referencePoints=refPoints, name='RP_Ball')

        #Criacao da massa da esfera
        p = mdb.models['Model-1'].parts['Esfera']
        region=p.sets['RP_Ball']
        mdb.models['Model-1'].parts[param_bola['name']].
                                            engineeringFeatures.
                                            PointMassInertia(
            name='Massa_Bola', region=region, mass=param_bola['Massa'],
                                                i11=1.0, i22=1.0, i33=1.
                                                0, alpha=0.0, composite=
                                                0.0)

        #Criacao da superficie do suporte
        a = mdb.models['Model-1'].rootAssembly
        s1 = a.instances['Suporte'].faces
        side2Faces1 = s1.getByBoundingBox(zMin = param_osso['Altura'])
        a.Surface(side2Faces=side2Faces1, name='Superficie_Suporte')


    if Camadas == 'R':
        #Import de todas as tres partes
        a = mdb.models['Model-1'].rootAssembly
        a.DatumCsysByDefault(CARTESIAN)
        p = mdb.models['Model-1'].parts[param_osso['name']]
        a.Instance(name=param_osso['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_pele['name']]
        a.Instance(name=param_pele['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_RIG['name']]
        a.Instance(name=param_RIG['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_bola['name']]
        a.Instance(name=param_bola['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_impacto['name']]
        a.Instance(name=param_impacto['name'], part=p, dependent=ON)
        session.viewports['Viewport: 1'].view.fitView()

        #Posicionamento das pecas
        a.translate(instanceList=(param_pele['name'], ), vector=(0.0, 0.
                                            0, -param_pele['Altura']))
        a.translate(instanceList=(param_RIG['name'], ), vector=(0.0, 0.0
                                            , -param_pele['Altura'] -
                                            param_RIG['Altura']))
        a.translate(instanceList=(param_bola['name'], ), vector=(0.0, 0.
                                            0, -param_pele['Altura'] -
                                            param_RIG['Altura'] -
```

```
                                        param_bola['Diametro']/2. -
                                        0.01))
a.translate(instanceList=(param_impacto['name'], ), vector=(0.0,
                                         0.0, param_osso['Altura']))

#Criacao de Sets uteis
#Criacao do RP do Suporte
a = mdb.models['Model-1'].rootAssembly
r1 = a.instances['Suporte'].referencePoints
refPoints1=(r1[3], )
a.Set(referencePoints=refPoints1, name='RP_Suporte')

region=a.sets['RP_Suporte']
mdb.models['Model-1'].parts[param_impacto['name']].
                                        engineeringFeatures.
                                        PointMassInertia(
    name='Inertia_Inf', region=region, mass=1.0, i11=1.0, i22=1.
                                            0, i33=1.0, alpha=0.0,
                                            composite=0.0)


a = mdb.models['Model-1'].rootAssembly
r1 = a.instances['Esfera'].referencePoints
refPoints1=(r1[3], )
a.Set(referencePoints=refPoints1, name='RF_Ball')

p = mdb.models['Model-1'].parts['Esfera']
r = p.referencePoints
refPoints=(r[3], )
p.Set(referencePoints=refPoints, name='RP_Ball')

#Criacao da massa da esfera
p = mdb.models['Model-1'].parts['Esfera']
region=p.sets['RP_Ball']
mdb.models['Model-1'].parts[param_bola['name']].
                                        engineeringFeatures.
                                        PointMassInertia(
    name='Massa_Bola', region=region, mass=param_bola['Massa'],
                                            i11=1.0, i22=1.0, i33=1.
                                            0, alpha=0.0, composite=
                                            0.0)

#Criacao da superficie do suporte
a = mdb.models['Model-1'].rootAssembly
s1 = a.instances['Suporte'].faces
side2Faces1 = s1.getByBoundingBox(zMin = param_osso['Altura'])
a.Surface(side2Faces=side2Faces1, name='Superficie_Suporte')
```

```
if Camadas == 'FR':
    #Import de todas as tres partes
    a = mdb.models['Model-1'].rootAssembly
    a.DatumCsysByDefault(CARTESIAN)
    p = mdb.models['Model-1'].parts[param_osso['name']]
    a.Instance(name=param_osso['name'], part=p, dependent=ON)
    p = mdb.models['Model-1'].parts[param_pele['name']]
    a.Instance(name=param_pele['name'], part=p, dependent=ON)
    p = mdb.models['Model-1'].parts[param_RIG['name']]
    a.Instance(name=param_RIG['name'], part=p, dependent=ON)
    p = mdb.models['Model-1'].parts[param_FLEX['name']]
    a.Instance(name=param_FLEX['name'], part=p, dependent=ON)
    p = mdb.models['Model-1'].parts[param_bola['name']]
    a.Instance(name=param_bola['name'], part=p, dependent=ON)
    p = mdb.models['Model-1'].parts[param_impacto['name']]
    a.Instance(name=param_impacto['name'], part=p, dependent=ON)
    session.viewports['Viewport: 1'].view.fitView()

    #Posicionamento das pecas
    a.translate(instanceList=(param_pele['name'], ), vector=(0.0, 0.
                                        0, -param_pele['Altura']))
    a.translate(instanceList=(param_FLEX['name'], ), vector=(0.0, 0.
                                        0, -param_pele['Altura'] -
                                        param_FLEX['Altura']))
    a.translate(instanceList=(param_bola['name'], ), vector=(0.0, 0.
                                        0, -param_pele['Altura'] -
                                        param_RIG['Altura'] -
                                        param_FLEX['Altura'] -
                                        param_bola['Diametro']/2. -
                                        0.01))
    a.translate(instanceList=(param_RIG['name'], ), vector=(0.0, 0.0
                                        , -param_pele['Altura'] -
                                        param_FLEX['Altura'] -
                                        param_RIG['Altura'] ))
    a.translate(instanceList=(param_impacto['name'], ), vector=(0.0,
                                         0.0, param_osso['Altura']))

    #Criacao de Sets uteis
    #Criacao do RP do Suporte
    a = mdb.models['Model-1'].rootAssembly
    r1 = a.instances['Suporte'].referencePoints
    refPoints1=(r1[3], )
    a.Set(referencePoints=refPoints1, name='RP_Suporte')

    region=a.sets['RP_Suporte']
```

```
        mdb.models['Model-1'].parts[param_impacto['name']].
                                            engineeringFeatures.
                                            PointMassInertia(
            name='Inertia_Inf', region=region, mass=1.0, i11=1.0, i22=1.
                                                0, i33=1.0, alpha=0.0,
                                                composite=0.0)


        a = mdb.models['Model-1'].rootAssembly
        r1 = a.instances['Esfera'].referencePoints
        refPoints1=(r1[3], )
        a.Set(referencePoints=refPoints1, name='RF_Ball')

        p = mdb.models['Model-1'].parts['Esfera']
        r = p.referencePoints
        refPoints=(r[3], )
        p.Set(referencePoints=refPoints, name='RP_Ball')

        #Criacao da massa da esfera
        p = mdb.models['Model-1'].parts['Esfera']
        region=p.sets['RP_Ball']
        mdb.models['Model-1'].parts[param_bola['name']].
                                            engineeringFeatures.
                                            PointMassInertia(
            name='Massa_Bola', region=region, mass=param_bola['Massa'],
                                                i11=1.0, i22=1.0, i33=1.
                                                0, alpha=0.0, composite=
                                                0.0)

        #Criacao da superficie do suporte
        a = mdb.models['Model-1'].rootAssembly
        s1 = a.instances['Suporte'].faces
        side2Faces1 = s1.getByBoundingBox(zMin = param_osso['Altura'])
        a.Surface(side2Faces=side2Faces1, name='Superficie_Suporte')


    if Camadas == 'RFR':
        #Import de todas as tres partes
        a = mdb.models['Model-1'].rootAssembly
        a.DatumCsysByDefault(CARTESIAN)
        p = mdb.models['Model-1'].parts[param_osso['name']]
        a.Instance(name=param_osso['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_pele['name']]
        a.Instance(name=param_pele['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_RIG['name']]
        a.Instance(name=param_RIG['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_FLEX['name']]
```

```
a.Instance(name=param_FLEX['name'], part=p, dependent=ON)
p = mdb.models['Model-1'].parts[param_bola['name']]
a.Instance(name=param_bola['name'], part=p, dependent=ON)
p = mdb.models['Model-1'].parts[param_impacto['name']]
a.Instance(name=param_impacto['name'], part=p, dependent=ON)
p = mdb.models['Model-1'].parts['RIG2']
a.Instance(name='RIG2', part=p, dependent=ON)
session.viewports['Viewport: 1'].view.fitView()

#Posicionamento das pecas
a.translate(instanceList=(param_pele['name'], ), vector=(0.0, 0.
                                    0, -param_pele['Altura']))
a.translate(instanceList=(param_RIG['name'], ), vector=(0.0, 0.0
                                    , -param_pele['Altura'] -
                                    param_RIG['Altura']))
a.translate(instanceList=(param_bola['name'], ), vector=(0.0, 0.
                                    0, -param_pele['Altura'] -
                                    param_RIG['Altura'] -
                                    param_FLEX['Altura'] -
                                    param_bola['Diametro']/2. -
                                    0.012))
a.translate(instanceList=(param_FLEX['name'], ), vector=(0.0, 0.
                                    0, -param_pele['Altura'] -
                                    param_FLEX['Altura'] -
                                    param_RIG['Altura'] ))
a.translate(instanceList=('RIG2', ), vector=(0.0, 0.0, -
                                    param_pele['Altura'] -
                                    param_FLEX['Altura'] - 2*
                                    param_RIG['Altura'] ))
a.translate(instanceList=(param_impacto['name'], ), vector=(0.0,
                                     0.0, param_osso['Altura']))

#Criacao de Sets uteis
#Criacao do RP do Suporte
a = mdb.models['Model-1'].rootAssembly
r1 = a.instances['Suporte'].referencePoints
refPoints1=(r1[3], )
a.Set(referencePoints=refPoints1, name='RP_Suporte')

region=a.sets['RP_Suporte']
mdb.models['Model-1'].parts[param_impacto['name']].
                                    engineeringFeatures.
                                    PointMassInertia(
    name='Inertia_Inf', region=region, mass=1.0, i11=1.0, i22=1.
                                        0, i33=1.0, alpha=0.0,
                                        composite=0.0)
```

```
        a = mdb.models['Model-1'].rootAssembly
        r1 = a.instances['Esfera'].referencePoints
        refPoints1=(r1[3], )
        a.Set(referencePoints=refPoints1, name='RF_Ball')

        p = mdb.models['Model-1'].parts['Esfera']
        r = p.referencePoints
        refPoints=(r[3], )
        p.Set(referencePoints=refPoints, name='RP_Ball')

        #Criacao da massa da esfera
        p = mdb.models['Model-1'].parts['Esfera']
        region=p.sets['RP_Ball']
        mdb.models['Model-1'].parts[param_bola['name']].
                                            engineeringFeatures.
                                            PointMassInertia(
            name='Massa_Bola', region=region, mass=param_bola['Massa'],
                                                i11=1.0, i22=1.0, i33=1.
                                                0, alpha=0.0, composite=
                                                0.0)

        #Criacao da superficie do suporte
        a = mdb.models['Model-1'].rootAssembly
        s1 = a.instances['Suporte'].faces
        side2Faces1 = s1.getByBoundingBox(zMin = param_osso['Altura'])
        a.Surface(side2Faces=side2Faces1, name='Superficie_Suporte')


    if Camadas == 'FRF':
        #Import de todas as tres partes
        a = mdb.models['Model-1'].rootAssembly
        a.DatumCsysByDefault(CARTESIAN)
        p = mdb.models['Model-1'].parts[param_osso['name']]
        a.Instance(name=param_osso['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_pele['name']]
        a.Instance(name=param_pele['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_RIG['name']]
        a.Instance(name=param_RIG['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_FLEX['name']]
        a.Instance(name=param_FLEX['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_bola['name']]
        a.Instance(name=param_bola['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts[param_impacto['name']]
        a.Instance(name=param_impacto['name'], part=p, dependent=ON)
        p = mdb.models['Model-1'].parts['FLEX2']
        a.Instance(name='FLEX2', part=p, dependent=ON)
```

```
session.viewports['Viewport: 1'].view.fitView()

#Posicionamento das pecas
a.translate(instanceList=(param_pele['name'], ), vector=(0.0, 0.
                                    0, -param_pele['Altura']))
a.translate(instanceList=(param_FLEX['name'], ), vector=(0.0, 0.
                                    0, -param_pele['Altura'] -
                                    param_FLEX['Altura']))
a.translate(instanceList=(param_bola['name'], ), vector=(0.0, 0.
                                    0, -param_pele['Altura'] -
                                    param_RIG['Altura'] -
                                    param_FLEX['Altura'] -
                                    param_bola['Diametro']/2. -
                                    0.012))
a.translate(instanceList=(param_RIG['name'], ), vector=(0.0, 0.0
                                    , -param_pele['Altura'] -
                                    param_FLEX['Altura'] -
                                    param_RIG['Altura'] ))
a.translate(instanceList=('FLEX2', ), vector=(0.0, 0.0, -
                                    param_pele['Altura'] - 2*
                                    param_FLEX['Altura'] -
                                    param_RIG['Altura'] ))
a.translate(instanceList=(param_impacto['name'], ), vector=(0.0,
                                     0.0, param_osso['Altura']))

#Criacao de Sets uteis
#Criacao do RP do Suporte
a = mdb.models['Model-1'].rootAssembly
r1 = a.instances['Suporte'].referencePoints
refPoints1=(r1[3], )
a.Set(referencePoints=refPoints1, name='RP_Suporte')

region=a.sets['RP_Suporte']
mdb.models['Model-1'].parts[param_impacto['name']].
                                    engineeringFeatures.
                                    PointMassInertia(
    name='Inertia_Inf', region=region, mass=1.0, i11=1.0, i22=1.
                                        0, i33=1.0, alpha=0.0,
                                        composite=0.0)


a = mdb.models['Model-1'].rootAssembly
r1 = a.instances['Esfera'].referencePoints
refPoints1=(r1[3], )
a.Set(referencePoints=refPoints1, name='RF_Ball')

p = mdb.models['Model-1'].parts['Esfera']
```

```
        r = p.referencePoints
        refPoints=(r[3], )
        p.Set(referencePoints=refPoints, name='RP_Ball')

        #Criacao da massa da esfera
        p = mdb.models['Model-1'].parts['Esfera']
        region=p.sets['RP_Ball']
        mdb.models['Model-1'].parts[param_bola['name']].
                                            engineeringFeatures.
                                            PointMassInertia(
            name='Massa_Bola', region=region, mass=param_bola['Massa'],
                                                i11=1.0, i22=1.0, i33=1.
                                                0, alpha=0.0, composite=
                                                0.0)

        #Criacao da superficie do suporte
        a = mdb.models['Model-1'].rootAssembly
        s1 = a.instances['Suporte'].faces
        side2Faces1 = s1.getByBoundingBox(zMin = param_osso['Altura'])
        a.Surface(side2Faces=side2Faces1, name='Superficie_Suporte')




#########################################################
# Interacao
#########################################################
if Mod==1:
  #Interacao entre a parte superiora e a parte do meio
  region1 = a.surfaces['Face_Inf-Parte_Superior']
  region2 = a.surfaces['Face_Sup-Parte_Meio']
  mdb.models['Model-1'].Tie(name='Ligacao_Sup_Meio', master=region1,
                                    slave=region2,
      positionToleranceMethod=COMPUTED, adjust=ON, tieRotations=ON,
                                    thickness=ON)
  #Interacao entre parte do meio e inferiora
  region1 = a.surfaces['Face_Inf-Parte_Meio']
  region2 = a.surfaces['Face_Sup-Parte_Inf']
  mdb.models['Model-1'].Tie(name='Ligacao_Meio_Inf', master=region2,
                                    slave=region1,
      positionToleranceMethod=SPECIFIED, positionTolerance=0.0, adjust=
                                    ON, tieRotations=ON,
                                    constraintEnforcement=
                                    SURFACE_TO_SURFACE, thickness=ON)
```

```
if Mod == 2:

  if param_comp['quad'] == True:
    #IMPLiCITO
    #Contato Generalizado
    mdb.models['Model-1'].ContactProperty('Contact')
    mdb.models['Model-1'].interactionProperties['Contact'].
                                    TangentialBehavior(
        formulation=PENALTY, directionality=ISOTROPIC,
                                    slipRateDependency=OFF,
        pressureDependency=OFF, temperatureDependency=OFF, dependencies=
                                    0, table=((
        0.05, ), ), shearStressLimit=None, maximumElasticSlip=FRACTION,
        fraction=0.005, elasticSlipStiffness=None)
    mdb.models['Model-1'].ContactStd(name='Contato', createStepName='
                                    Initial')
    mdb.models['Model-1'].interactions['Contato'].includedPairs.
                                    setValuesInStep(
        stepName='Initial', useAllstar=ON)
    mdb.models['Model-1'].interactions['Contato'].
                                    contactPropertyAssignments.
                                    appendInStep(
        stepName='Initial', assignments=((GLOBAL, SELF, 'Contact'), ))

  if param_comp['quad'] == False:
    ## EXPLiCITO
    mdb.models['Model-1'].ContactProperty('Contact')
    mdb.models['Model-1'].interactionProperties['Contact'].
                                    TangentialBehavior(
        formulation=PENALTY, directionality=ISOTROPIC,
                                    slipRateDependency=OFF,
        pressureDependency=OFF, temperatureDependency=OFF, dependencies=
                                    0, table=((
        0.25, ), ), shearStressLimit=None, maximumElasticSlip=FRACTION,
        fraction=0.005, elasticSlipStiffness=None)
    mdb.models['Model-1'].interactionProperties['Contact'].
                                    NormalBehavior(
        pressureOverclosure=HARD, allowSeparation=ON,
                                    constraintEnforcementMethod=DEFAULT)
    mdb.models['Model-1'].ContactExp(name='Contato', createStepName='
                                    Initial')
    mdb.models['Model-1'].interactions['Contato'].includedPairs.
                                    setValuesInStep(
        stepName='Initial', useAllstar=ON)
    mdb.models['Model-1'].interactions['Contato'].
                                    contactPropertyAssignments.
```

```
                                    appendInStep(
        stepName='Initial', assignments=((GLOBAL, SELF, 'Contact'), ))




  #Suporte da Base
  a = mdb.models['Model-1'].rootAssembly
  region1=a.instances['Assembly_Inferior'].surfaces['Inferior']
  a = mdb.models['Model-1'].rootAssembly
  region2=a.instances['Assembly_Corpo_de_Prova'].surfaces['
                                    Superficie_Inferior']
  mdb.models['Model-1'].Tie(name='Base', master=region1, slave=region2,
      positionToleranceMethod=SPECIFIED, positionTolerance=0.0, adjust=
                                    OFF, tieRotations=ON,
      thickness=ON)
  #Suporte do Topo
  a = mdb.models['Model-1'].rootAssembly
  region1=a.instances['Assembly_Superior'].surfaces['Superior']
  a = mdb.models['Model-1'].rootAssembly
  region2=a.instances['Assembly_Corpo_de_Prova'].surfaces['
                                    Superficie_Superior']
  mdb.models['Model-1'].Tie(name='Topo', master=region1, slave=region2,
     positionToleranceMethod=SPECIFIED, positionTolerance=0.0, adjust=
                                    OFF, tieRotations=ON,
      thickness=ON)

if Mod == 3:
    if Camadas == 'F':
        ## EXPLiCITO
        # CONTATO GERAL
        mdb.models['Model-1'].ContactProperty('Contact')
        mdb.models['Model-1'].interactionProperties['Contact'].
                                            TangentialBehavior(
            formulation=PENALTY, directionality=ISOTROPIC,
                                            slipRateDependency=OFF,
            pressureDependency=OFF, temperatureDependency=OFF,
                                            dependencies=0, table=((
            0.95, ), ), shearStressLimit=None, maximumElasticSlip=
                                            FRACTION,
            fraction=0.01, elasticSlipStiffness=None)
        mdb.models['Model-1'].interactionProperties['Contact'].
                                            NormalBehavior(
            pressureOverclosure=HARD, allowSeparation=ON,
                                            constraintEnforcementMethod
                                            =DEFAULT)
```

```
mdb.models['Model-1'].ContactExp(name='Contato', createStepName=
                                    'Initial')
mdb.models['Model-1'].interactions['Contato'].includedPairs.
                                    setValuesInStep(
    stepName='Initial', useAllstar=ON)
mdb.models['Model-1'].interactions['Contato'].
                                    contactPropertyAssignments.
                                    appendInStep(
    stepName='Initial', assignments=((GLOBAL, SELF, 'Contact'),
                                        ))

#CONTATO OSSO COM O SUPORTE
mdb.models['Model-1'].ContactProperty('Contato_Osso_Suporte')
mdb.models['Model-1'].interactionProperties['
                                    Contato_Osso_Suporte'].
                                    TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
    0.5, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['
                                    Contato_Osso_Suporte'].
                                    NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
a = mdb.models['Model-1'].rootAssembly
region1=a.surfaces['Superficie_Suporte']
a = mdb.models['Model-1'].rootAssembly
region2=a.instances['Osso'].surfaces['Superficie_Inferior']
mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Osso-
                                    Suporte',
    createStepName='Initial', master = region1, slave = region2,
    mechanicalConstraint=KINEMATIC, sliding=FINITE,
    interactionProperty='Contato_Osso_Suporte', initialClearance
                                        =OMIT,
    datumAxis=None, clearanceRegion=None)

#CONTATO PELE COM OSSO
mdb.models['Model-1'].ContactProperty('Contato_Osso_Pele')
mdb.models['Model-1'].interactionProperties['Contato_Osso_Pele']
                                    .TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
```

```
    pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
    0.95, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['Contato_Osso_Pele']
                                    .NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
a = mdb.models['Model-1'].rootAssembly
region1=a.instances['Osso'].surfaces['Superficie_Superior']
a = mdb.models['Model-1'].rootAssembly
region2=a.instances['Pele'].surfaces['Superficie_Inferior']
mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Pele-
                                    Osso',
    createStepName='Initial', master = region1, slave = region2,
    mechanicalConstraint=KINEMATIC, sliding=FINITE,
    interactionProperty='Contato_Osso_Suporte', initialClearance
                                        =OMIT,
    datumAxis=None, clearanceRegion=None)

#CONTATO PELE COM FLEX
mdb.models['Model-1'].ContactProperty('Contato_Pele_Flex')
mdb.models['Model-1'].interactionProperties['Contato_Pele_Flex']
                                    .TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
    0.1, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['Contato_Pele_Flex']
                                    .NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
a = mdb.models['Model-1'].rootAssembly
region1=a.instances['Pele'].surfaces['Superficie_Superior']
a = mdb.models['Model-1'].rootAssembly
region2=a.instances['FLEX'].surfaces['Superficie_Inferior']
mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Pele-
                                    Flex',
    createStepName='Initial', master = region1, slave = region2,
    mechanicalConstraint=KINEMATIC, sliding=FINITE,
    interactionProperty='Contato_Pele_Flex', initialClearance=
                                        OMIT,
    datumAxis=None, clearanceRegion=None)
```

```
    a = mdb.models['Model-1'].rootAssembly
    e1 = a.instances['Osso'].edges
    edges1 = e1.getByBoundingBox(zMin = param_osso['Altura'])
    a.Set(edges=edges1, name='Borda_Engaste')

if Camadas == 'sEVA':
    # CONTATO GERAL
    mdb.models['Model-1'].ContactProperty('Contact')
    mdb.models['Model-1'].interactionProperties['Contact'].
                                        TangentialBehavior(
        formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
        pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
        0.95, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
        fraction=0.01, elasticSlipStiffness=None)
    mdb.models['Model-1'].interactionProperties['Contact'].
                                        NormalBehavior(
        pressureOverclosure=HARD, allowSeparation=ON,
                                        constraintEnforcementMethod
                                        =DEFAULT)
    mdb.models['Model-1'].ContactExp(name='Contato', createStepName=
                                        'Initial')
    mdb.models['Model-1'].interactions['Contato'].includedPairs.
                                        setValuesInStep(
        stepName='Initial', useAllstar=ON)
    mdb.models['Model-1'].interactions['Contato'].
                                        contactPropertyAssignments.
                                        appendInStep(
        stepName='Initial', assignments=((GLOBAL, SELF, 'Contact'),
                                        ))

    #CONTATO OSSO COM O SUPORTE
    mdb.models['Model-1'].ContactProperty('Contato_Osso_Suporte')
    mdb.models['Model-1'].interactionProperties['
                                        Contato_Osso_Suporte'].
                                        TangentialBehavior(
        formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
        pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
        0.5, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
```

```
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['
                                    Contato_Osso_Suporte'].
                                    NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
a = mdb.models['Model-1'].rootAssembly
region1=a.surfaces['Superficie_Suporte']
a = mdb.models['Model-1'].rootAssembly
region2=a.instances['Osso'].surfaces['Superficie_Inferior']
mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Osso-
                                    Suporte',
    createStepName='Initial', master = region1, slave = region2,
    mechanicalConstraint=KINEMATIC, sliding=FINITE,
    interactionProperty='Contato_Osso_Suporte', initialClearance
                                        =OMIT,
    datumAxis=None, clearanceRegion=None)

#CONTATO PELE COM OSSO
mdb.models['Model-1'].ContactProperty('Contato_Osso_Pele')
mdb.models['Model-1'].interactionProperties['Contato_Osso_Pele']
                                    .TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
    0.95, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['Contato_Osso_Pele']
                                    .NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
a = mdb.models['Model-1'].rootAssembly
region1=a.instances['Osso'].surfaces['Superficie_Superior']
a = mdb.models['Model-1'].rootAssembly
region2=a.instances['Pele'].surfaces['Superficie_Inferior']
mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Pele-
                                    Osso',
    createStepName='Initial', master = region1, slave = region2,
    mechanicalConstraint=KINEMATIC, sliding=FINITE,
    interactionProperty='Contato_Osso_Suporte', initialClearance
                                        =OMIT,
    datumAxis=None, clearanceRegion=None)


a = mdb.models['Model-1'].rootAssembly
```

```
        e1 = a.instances['Osso'].edges
        edges1 = e1.getByBoundingBox(zMin = param_osso['Altura'])
        a.Set(edges=edges1, name='Borda_Engaste')

    if Camadas == 'Osso':
        # EXPLiCITO
        mdb.models['Model-1'].ContactProperty('Contact')
        mdb.models['Model-1'].interactionProperties['Contact'].
                                            TangentialBehavior(
            formulation=PENALTY, directionality=ISOTROPIC,
                                            slipRateDependency=OFF,
            pressureDependency=OFF, temperatureDependency=OFF,
                                            dependencies=0, table=((
            0.3, ), ), shearStressLimit=None, maximumElasticSlip=
                                            FRACTION,
            fraction=0.01, elasticSlipStiffness=None)
        mdb.models['Model-1'].interactionProperties['Contact'].
                                            NormalBehavior(
            pressureOverclosure=HARD, allowSeparation=ON,
                                            constraintEnforcementMethod
                                            =DEFAULT)
        mdb.models['Model-1'].ContactExp(name='Contato', createStepName=
                                            'Initial')
        mdb.models['Model-1'].interactions['Contato'].includedPairs.
                                            setValuesInStep(
            stepName='Initial', useAllstar=ON)
        mdb.models['Model-1'].interactions['Contato'].
                                            contactPropertyAssignments.
                                            appendInStep(
            stepName='Initial', assignments=((GLOBAL, SELF, 'Contact'),
                                            ))

        #CONTATO OSSO COM O SUPORTE
        mdb.models['Model-1'].ContactProperty('Contato_Osso_Suporte')
        mdb.models['Model-1'].interactionProperties['
                                            Contato_Osso_Suporte'].
                                            TangentialBehavior(
            formulation=PENALTY, directionality=ISOTROPIC,
                                            slipRateDependency=OFF,
            pressureDependency=OFF, temperatureDependency=OFF,
                                            dependencies=0, table=((
            0.5, ), ), shearStressLimit=None, maximumElasticSlip=
                                            FRACTION,
            fraction=0.01, elasticSlipStiffness=None)
        mdb.models['Model-1'].interactionProperties['
                                            Contato_Osso_Suporte'].
                                            NormalBehavior(
```

```
        pressureOverclosure=HARD, allowSeparation=ON,
        constraintEnforcementMethod=DEFAULT)
    a = mdb.models['Model-1'].rootAssembly
    region1=a.surfaces['Superficie_Suporte']
    a = mdb.models['Model-1'].rootAssembly
    region2=a.instances['Osso'].surfaces['Superficie_Inferior']
    mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Osso-
                                        Suporte',
        createStepName='Initial', master = region1, slave = region2,
        mechanicalConstraint=KINEMATIC, sliding=FINITE,
        interactionProperty='Contato_Osso_Suporte', initialClearance
                                            =OMIT,
        datumAxis=None, clearanceRegion=None)


    a = mdb.models['Model-1'].rootAssembly
    e1 = a.instances['Osso'].edges
    edges1 = e1.getByBoundingBox(zMin = param_osso['Altura'])
    a.Set(edges=edges1, name='Borda_Engaste')

if Camadas == 'R':
    ## EXPLiCITO
    # CONTATO GERAL
    mdb.models['Model-1'].ContactProperty('Contact')
    mdb.models['Model-1'].interactionProperties['Contact'].
                                        TangentialBehavior(
        formulation=PENALTY, directionality=ISOTROPIC,
                                            slipRateDependency=OFF,
        pressureDependency=OFF, temperatureDependency=OFF,
                                            dependencies=0, table=((
        0.95, ), ), shearStressLimit=None, maximumElasticSlip=
                                            FRACTION,
        fraction=0.01, elasticSlipStiffness=None)
    mdb.models['Model-1'].interactionProperties['Contact'].
                                        NormalBehavior(
        pressureOverclosure=HARD, allowSeparation=ON,
                                            constraintEnforcementMethod
                                            =DEFAULT)
    mdb.models['Model-1'].ContactExp(name='Contato', createStepName=
                                        'Initial')
    mdb.models['Model-1'].interactions['Contato'].includedPairs.
                                        setValuesInStep(
        stepName='Initial', useAllstar=ON)
    mdb.models['Model-1'].interactions['Contato'].
                                        contactPropertyAssignments.
                                        appendInStep(
```

```
    stepName='Initial', assignments=((GLOBAL, SELF, 'Contact'),
                                        ))

#CONTATO OSSO COM O SUPORTE
mdb.models['Model-1'].ContactProperty('Contato_Osso_Suporte')
mdb.models['Model-1'].interactionProperties['
                                    Contato_Osso_Suporte'].
                                    TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
    0.5, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['
                                    Contato_Osso_Suporte'].
                                    NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
a = mdb.models['Model-1'].rootAssembly
region1=a.surfaces['Superficie_Suporte']
a = mdb.models['Model-1'].rootAssembly
region2=a.instances['Osso'].surfaces['Superficie_Inferior']
mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Osso-
                                    Suporte',
    createStepName='Initial', master = region1, slave = region2,
    mechanicalConstraint=KINEMATIC, sliding=FINITE,
    interactionProperty='Contato_Osso_Suporte', initialClearance
                                        =OMIT,
    datumAxis=None, clearanceRegion=None)

#CONTATO PELE COM OSSO
mdb.models['Model-1'].ContactProperty('Contato_Osso_Pele')
mdb.models['Model-1'].interactionProperties['Contato_Osso_Pele']
                                    .TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
    0.95, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['Contato_Osso_Pele']
                                    .NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
```

```
        a = mdb.models['Model-1'].rootAssembly
        region1=a.instances['Osso'].surfaces['Superficie_Superior']
        a = mdb.models['Model-1'].rootAssembly
        region2=a.instances['Pele'].surfaces['Superficie_Inferior']
        mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Pele-
                                                Osso',
            createStepName='Initial', master = region1, slave = region2,
            mechanicalConstraint=KINEMATIC, sliding=FINITE,
            interactionProperty='Contato_Osso_Suporte', initialClearance
                                                =OMIT,
            datumAxis=None, clearanceRegion=None)

        #CONTATO PELE COM RIG
        mdb.models['Model-1'].ContactProperty('Contato_Pele_Rig')
        mdb.models['Model-1'].interactionProperties['Contato_Pele_Rig'].
                                                TangentialBehavior(
            formulation=PENALTY, directionality=ISOTROPIC,
                                                slipRateDependency=OFF,
            pressureDependency=OFF, temperatureDependency=OFF,
                                                dependencies=0, table=((
            0.4, ), ), shearStressLimit=None, maximumElasticSlip=
                                                FRACTION,
            fraction=0.01, elasticSlipStiffness=None)
        mdb.models['Model-1'].interactionProperties['Contato_Pele_Rig'].
                                                NormalBehavior(
            pressureOverclosure=HARD, allowSeparation=ON,
            constraintEnforcementMethod=DEFAULT)
        a = mdb.models['Model-1'].rootAssembly
        region1=a.instances['Pele'].surfaces['Superficie_Superior']
        a = mdb.models['Model-1'].rootAssembly
        region2=a.instances['RIG'].surfaces['Superficie_Inferior']
        mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Pele-Rig
                                                ',
            createStepName='Initial', master = region1, slave = region2,
            mechanicalConstraint=KINEMATIC, sliding=FINITE,
            interactionProperty='Contato_Pele_Rig', initialClearance=
                                                OMIT,
            datumAxis=None, clearanceRegion=None)


    if Camadas == 'FR':
        ## EXPLiCITO
        # CONTATO GERAL
        mdb.models['Model-1'].ContactProperty('Contact')
        mdb.models['Model-1'].interactionProperties['Contact'].
                                                TangentialBehavior(
```

```
    formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
    0.95, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['Contact'].
                                    NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
                                        constraintEnforcementMethod
                                        =DEFAULT)
mdb.models['Model-1'].ContactExp(name='Contato', createStepName=
                                    'Initial')
mdb.models['Model-1'].interactions['Contato'].includedPairs.
                                    setValuesInStep(
    stepName='Initial', useAllstar=ON)
mdb.models['Model-1'].interactions['Contato'].
                                    contactPropertyAssignments.
                                    appendInStep(
    stepName='Initial', assignments=((GLOBAL, SELF, 'Contact'),
                                        ))

#CONTATO OSSO COM O SUPORTE
mdb.models['Model-1'].ContactProperty('Contato_Osso_Suporte')
mdb.models['Model-1'].interactionProperties['
                                    Contato_Osso_Suporte'].
                                    TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
    0.5, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['
                                    Contato_Osso_Suporte'].
                                    NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
a = mdb.models['Model-1'].rootAssembly
region1=a.surfaces['Superficie_Suporte']
a = mdb.models['Model-1'].rootAssembly
region2=a.instances['Osso'].surfaces['Superficie_Inferior']
mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Osso-
                                    Suporte',
    createStepName='Initial', master = region1, slave = region2,
```

```
    mechanicalConstraint=KINEMATIC, sliding=FINITE,
    interactionProperty='Contato_Osso_Suporte', initialClearance
                                        =OMIT,
    datumAxis=None, clearanceRegion=None)

#CONTATO PELE COM OSSO
mdb.models['Model-1'].ContactProperty('Contato_Osso_Pele')
mdb.models['Model-1'].interactionProperties['Contato_Osso_Pele']
                                    .TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
    0.95, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['Contato_Osso_Pele']
                                    .NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
a = mdb.models['Model-1'].rootAssembly
region1=a.instances['Osso'].surfaces['Superficie_Superior']
a = mdb.models['Model-1'].rootAssembly
region2=a.instances['Pele'].surfaces['Superficie_Inferior']
mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Pele-
                                    Osso',
    createStepName='Initial', master = region1, slave = region2,
    mechanicalConstraint=KINEMATIC, sliding=FINITE,
    interactionProperty='Contato_Osso_Suporte', initialClearance
                                        =OMIT,
    datumAxis=None, clearanceRegion=None)

#CONTATO PELE COM FLEX
mdb.models['Model-1'].ContactProperty('Contato_Pele_Flex')
mdb.models['Model-1'].interactionProperties['Contato_Pele_Flex']
                                    .TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
    0.1, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['Contato_Pele_Flex']
                                    .NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
```

```
        a = mdb.models['Model-1'].rootAssembly
        region1=a.instances['Pele'].surfaces['Superficie_Superior']
        a = mdb.models['Model-1'].rootAssembly
        region2=a.instances['FLEX'].surfaces['Superficie_Inferior']
        mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Pele-
                                                Flex',
            createStepName='Initial', master = region1, slave = region2,
            mechanicalConstraint=KINEMATIC, sliding=FINITE,
            interactionProperty='Contato_Pele_Flex', initialClearance=
                                                OMIT,
            datumAxis=None, clearanceRegion=None)


        #CONTATO FLEX COM RIG
        mdb.models['Model-1'].ContactProperty('Contato_Flex_Rig')
        mdb.models['Model-1'].interactionProperties['Contato_Flex_Rig'].
                                                TangentialBehavior(
            formulation=PENALTY, directionality=ISOTROPIC,
                                                slipRateDependency=OFF,
            pressureDependency=OFF, temperatureDependency=OFF,
                                                dependencies=0, table=((
            0.7, ), ), shearStressLimit=None, maximumElasticSlip=
                                                FRACTION,
            fraction=0.01, elasticSlipStiffness=None)
        mdb.models['Model-1'].interactionProperties['Contato_Flex_Rig'].
                                                NormalBehavior(
            pressureOverclosure=HARD, allowSeparation=ON,
            constraintEnforcementMethod=DEFAULT)
        a = mdb.models['Model-1'].rootAssembly
        region1=a.instances['FLEX'].surfaces['Superficie_Superior']
        a = mdb.models['Model-1'].rootAssembly
        region2=a.instances['RIG'].surfaces['Superficie_Inferior']
        mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Flex-Rig
                                                ',
            createStepName='Initial', master = region1, slave = region2,
            mechanicalConstraint=KINEMATIC, sliding=FINITE,
            interactionProperty='Contato_Flex_Rig', initialClearance=
                                                OMIT,
            datumAxis=None, clearanceRegion=None)

    if Camadas == 'RFR':
        ## EXPLiCITO
        # CONTATO GERAL
        mdb.models['Model-1'].ContactProperty('Contact')
        mdb.models['Model-1'].interactionProperties['Contact'].
                                                TangentialBehavior(
```

```
    formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
    0.95, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['Contact'].
                                    NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
                                        constraintEnforcementMethod
                                        =DEFAULT)
mdb.models['Model-1'].ContactExp(name='Contato', createStepName=
                                    'Initial')
mdb.models['Model-1'].interactions['Contato'].includedPairs.
                                    setValuesInStep(
    stepName='Initial', useAllstar=ON)
mdb.models['Model-1'].interactions['Contato'].
                                    contactPropertyAssignments.
                                    appendInStep(
    stepName='Initial', assignments=((GLOBAL, SELF, 'Contact'),
                                        ))

#CONTATO OSSO COM O SUPORTE
mdb.models['Model-1'].ContactProperty('Contato_Osso_Suporte')
mdb.models['Model-1'].interactionProperties['
                                    Contato_Osso_Suporte'].
                                    TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
    0.5, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['
                                    Contato_Osso_Suporte'].
                                    NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
a = mdb.models['Model-1'].rootAssembly
region1=a.surfaces['Superficie_Suporte']
a = mdb.models['Model-1'].rootAssembly
region2=a.instances['Osso'].surfaces['Superficie_Inferior']
mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Osso-
                                    Suporte',
    createStepName='Initial', master = region1, slave = region2,
```

```
    mechanicalConstraint=KINEMATIC, sliding=FINITE,
    interactionProperty='Contato_Osso_Suporte', initialClearance
                                        =OMIT,
    datumAxis=None, clearanceRegion=None)

#CONTATO PELE COM OSSO
mdb.models['Model-1'].ContactProperty('Contato_Osso_Pele')
mdb.models['Model-1'].interactionProperties['Contato_Osso_Pele']
                                    .TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
    0.95, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['Contato_Osso_Pele']
                                    .NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
a = mdb.models['Model-1'].rootAssembly
region1=a.instances['Osso'].surfaces['Superficie_Superior']
a = mdb.models['Model-1'].rootAssembly
region2=a.instances['Pele'].surfaces['Superficie_Inferior']
mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Pele-
                                    Osso',
    createStepName='Initial', master = region1, slave = region2,
    mechanicalConstraint=KINEMATIC, sliding=FINITE,
    interactionProperty='Contato_Osso_Suporte', initialClearance
                                        =OMIT,
    datumAxis=None, clearanceRegion=None)

#CONTATO PELE COM RIG
mdb.models['Model-1'].ContactProperty('Contato_Pele_Rig')
mdb.models['Model-1'].interactionProperties['Contato_Pele_Rig'].
                                    TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
    0.2, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['Contato_Pele_Rig'].
                                    NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
```

```
a = mdb.models['Model-1'].rootAssembly
region1=a.instances['Pele'].surfaces['Superficie_Superior']
a = mdb.models['Model-1'].rootAssembly
region2=a.instances['RIG'].surfaces['Superficie_Inferior']
mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Pele-Rig
                                        ',
    createStepName='Initial', master = region1, slave = region2,
    mechanicalConstraint=KINEMATIC, sliding=FINITE,
    interactionProperty='Contato_Pele_Rig', initialClearance=
                                            OMIT,
    datumAxis=None, clearanceRegion=None)


#CONTATO RIG COM FLEX
mdb.models['Model-1'].ContactProperty('Contato_Flex_Rig')
mdb.models['Model-1'].interactionProperties['Contato_Flex_Rig'].
                                        TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC,
                                            slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF,
                                            dependencies=0, table=((
    0.5, ), ), shearStressLimit=None, maximumElasticSlip=
                                            FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['Contato_Flex_Rig'].
                                        NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
a = mdb.models['Model-1'].rootAssembly
region1=a.instances['RIG'].surfaces['Superficie_Superior']
a = mdb.models['Model-1'].rootAssembly
region2=a.instances['FLEX'].surfaces['Superficie_Inferior']
mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Rig-Flex
                                        ',
    createStepName='Initial', master = region1, slave = region2,
    mechanicalConstraint=KINEMATIC, sliding=FINITE,
    interactionProperty='Contato_Flex_Rig', initialClearance=
                                            OMIT,
    datumAxis=None, clearanceRegion=None)

#CONTATO FLEX COM RIG
a = mdb.models['Model-1'].rootAssembly
region1=a.instances['FLEX'].surfaces['Superficie_Superior']
a = mdb.models['Model-1'].rootAssembly
region2=a.instances['RIG2'].surfaces['Superficie_Inferior']
mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Flex-Rig
                                        ',
```

```
        createStepName='Initial', master = region1, slave = region2,
        mechanicalConstraint=KINEMATIC, sliding=FINITE,
        interactionProperty='Contato_Flex_Rig', initialClearance=
                                            OMIT,
        datumAxis=None, clearanceRegion=None)

if Camadas == 'FRF':
    ## EXPLiCITO
    # CONTATO GERAL
    mdb.models['Model-1'].ContactProperty('Contact')
    mdb.models['Model-1'].interactionProperties['Contact'].
                                        TangentialBehavior(
        formulation=PENALTY, directionality=ISOTROPIC,
                                            slipRateDependency=OFF,
        pressureDependency=OFF, temperatureDependency=OFF,
                                            dependencies=0, table=((
        0.95, ), ), shearStressLimit=None, maximumElasticSlip=
                                            FRACTION,
        fraction=0.01, elasticSlipStiffness=None)
    mdb.models['Model-1'].interactionProperties['Contact'].
                                        NormalBehavior(
        pressureOverclosure=HARD, allowSeparation=ON,
                                            constraintEnforcementMethod
                                            =DEFAULT)
    mdb.models['Model-1'].ContactExp(name='Contato', createStepName=
                                        'Initial')
    mdb.models['Model-1'].interactions['Contato'].includedPairs.
                                        setValuesInStep(
        stepName='Initial', useAllstar=ON)
    mdb.models['Model-1'].interactions['Contato'].
                                        contactPropertyAssignments.
                                        appendInStep(
        stepName='Initial', assignments=((GLOBAL, SELF, 'Contact'),
                                            ))

    #CONTATO OSSO COM O SUPORTE
    mdb.models['Model-1'].ContactProperty('Contato_Osso_Suporte')
    mdb.models['Model-1'].interactionProperties['
                                        Contato_Osso_Suporte'].
                                        TangentialBehavior(
        formulation=PENALTY, directionality=ISOTROPIC,
                                            slipRateDependency=OFF,
        pressureDependency=OFF, temperatureDependency=OFF,
                                            dependencies=0, table=((
        0.5, ), ), shearStressLimit=None, maximumElasticSlip=
                                            FRACTION,
        fraction=0.01, elasticSlipStiffness=None)
```

```
mdb.models['Model-1'].interactionProperties['
                                    Contato_Osso_Suporte'].
                                    NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
a = mdb.models['Model-1'].rootAssembly
region1=a.surfaces['Superficie_Suporte']
a = mdb.models['Model-1'].rootAssembly
region2=a.instances['Osso'].surfaces['Superficie_Inferior']
mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Osso-
                                    Suporte',
    createStepName='Initial', master = region1, slave = region2,
    mechanicalConstraint=KINEMATIC, sliding=FINITE,
    interactionProperty='Contato_Osso_Suporte', initialClearance
                                        =OMIT,
    datumAxis=None, clearanceRegion=None)

#CONTATO PELE COM OSSO
mdb.models['Model-1'].ContactProperty('Contato_Osso_Pele')
mdb.models['Model-1'].interactionProperties['Contato_Osso_Pele']
                                    .TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
    0.95, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['Contato_Osso_Pele']
                                    .NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
a = mdb.models['Model-1'].rootAssembly
region1=a.instances['Osso'].surfaces['Superficie_Superior']
a = mdb.models['Model-1'].rootAssembly
region2=a.instances['Pele'].surfaces['Superficie_Inferior']
mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Pele-
                                    Osso',
    createStepName='Initial', master = region1, slave = region2,
    mechanicalConstraint=KINEMATIC, sliding=FINITE,
    interactionProperty='Contato_Osso_Suporte', initialClearance
                                        =OMIT,
    datumAxis=None, clearanceRegion=None)

#CONTATO PELE COM FLEX
mdb.models['Model-1'].ContactProperty('Contato_Pele_Flex')
```

```
mdb.models['Model-1'].interactionProperties['Contato_Pele_Flex']
                                        .TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
    0.1, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['Contato_Pele_Flex']
                                        .NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
a = mdb.models['Model-1'].rootAssembly
region1=a.instances['Pele'].surfaces['Superficie_Superior']
a = mdb.models['Model-1'].rootAssembly
region2=a.instances['FLEX'].surfaces['Superficie_Inferior']
mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Pele-
                                        Flex',
    createStepName='Initial', master = region1, slave = region2,
    mechanicalConstraint=KINEMATIC, sliding=FINITE,
    interactionProperty='Contato_Pele_Flex', initialClearance=
                                        OMIT,
    datumAxis=None, clearanceRegion=None)


#CONTATO FLEX COM RIG
mdb.models['Model-1'].ContactProperty('Contato_Flex_Rig')
mdb.models['Model-1'].interactionProperties['Contato_Flex_Rig'].
                                        TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC,
                                        slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF,
                                        dependencies=0, table=((
    0.5, ), ), shearStressLimit=None, maximumElasticSlip=
                                        FRACTION,
    fraction=0.01, elasticSlipStiffness=None)
mdb.models['Model-1'].interactionProperties['Contato_Flex_Rig'].
                                        NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)
a = mdb.models['Model-1'].rootAssembly
region1=a.instances['FLEX'].surfaces['Superficie_Superior']
a = mdb.models['Model-1'].rootAssembly
region2=a.instances['RIG'].surfaces['Superficie_Inferior']
mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Flex-Rig
                                        ',
```

```
            createStepName='Initial', master = region1, slave = region2,
            mechanicalConstraint=KINEMATIC, sliding=FINITE,
            interactionProperty='Contato_Flex_Rig', initialClearance=
                                                    OMIT,
            datumAxis=None, clearanceRegion=None)

        #CONTATO RIG COM FLEX
        a = mdb.models['Model-1'].rootAssembly
        region1=a.instances['RIG'].surfaces['Superficie_Superior']
        a = mdb.models['Model-1'].rootAssembly
        region2=a.instances['FLEX2'].surfaces['Superficie_Inferior']
        mdb.models['Model-1'].SurfaceToSurfaceContactExp(name ='Rig-Flex
                                                ',
            createStepName='Initial', master = region1, slave = region2,
            mechanicalConstraint=KINEMATIC, sliding=FINITE,
            interactionProperty='Contato_Flex_Rig', initialClearance=
                                                    OMIT,
            datumAxis=None, clearanceRegion=None)









########################################################
# Steps
########################################################


if Mod == 1:
  mdb.models['Model-1'].ExplicitDynamicsStep(name='Dynamic', previous='
                                    Initial',
      massScaling=((SEMI_AUTOMATIC, MODEL, AT_BEGINNING, 1000.0, 5e-07,
      BELOW_MIN, 0, 0, 0.0, 0.0, 0, None), ))
if Mod == 2:

  if MAT == 1:
    #Condicoes especificas para cada taxa de deformacao
    if taxa_def == 1:
      timePeriod = 12.5
      timeInterval = 0.1
    if taxa_def == 2:
      timePeriod = 60.0
      timeInterval = 0.5
```

```
    if taxa_def == 3:
      timePeriod = 400.0
      timeInterval = 4.0

  if MAT == 2:
    #Condicoes especificas para cada taxa de deformacao
    if taxa_def == 1:
      timePeriod = 1.5
      timeInterval = 0.1
    if taxa_def == 2:
      timePeriod = 15.0
      timeInterval = 0.5
    if taxa_def == 3:
      timePeriod = 152.0
      timeInterval = 4.0

  if param_comp['quad'] == True:
    #IMPLiCITO
    mdb.models['Model-1'].ImplicitDynamicsStep(name='Dynamic', previous=
                                    'Initial',
         maxNumInc=10000, timePeriod=timePeriod, initialInc=0.1, minInc=
                                    1.e-10, application=QUASI_STATIC,
                                    nohaf=OFF, amplitude=RAMP,
         alpha=DEFAULT, initialConditions=OFF, nlgeom=ON)
    mdb.models['Model-1'].steps['Dynamic'].control.setValues(
                                    allowPropagation=OFF,
        resetDefaultValues=OFF, discontinuous=ON)

  if param_comp['quad'] == False:

    # ##EXPLiCITO
    mdb.models['Model-1'].ExplicitDynamicsStep(name='Dynamic', previous=
                                    'Initial',
        timePeriod=timePeriod, nlgeom=ON)

if Mod == 3:
    # ##EXPLiCITO
    if Camadas == 'F':
        timePeriod = 6.5e-3
        timeInterval = 2e-6
        mdb.models['Model-1'].ExplicitDynamicsStep(name='Dynamic',
                                        previous='Initial',
            timePeriod=timePeriod, nlgeom=ON)

    if Camadas == 'sEVA':
        timePeriod = 6.5e-3
        timeInterval = 2e-6
```

```
        mdb.models['Model-1'].ExplicitDynamicsStep(name='Dynamic',
                                                   previous='Initial',
            timePeriod=timePeriod, nlgeom=ON)

    if Camadas == 'Osso':
        timePeriod = 4.0e-3
        timeInterval = 2e-6
        mdb.models['Model-1'].ExplicitDynamicsStep(name='Dynamic',
                                                   previous='Initial',
            timePeriod=timePeriod, nlgeom=ON)

    if Camadas == 'R':
        timePeriod = 6.5e-3
        timeInterval = 2e-6
        mdb.models['Model-1'].ExplicitDynamicsStep(name='Dynamic',
                                                   previous='Initial',
            timePeriod=timePeriod, nlgeom=ON)

    if Camadas == 'FR':
        timePeriod = 6.5e-3
        timeInterval = 2e-6
        mdb.models['Model-1'].ExplicitDynamicsStep(name='Dynamic',
                                                   previous='Initial',
            timePeriod=timePeriod, nlgeom=ON)

    if Camadas == 'RFR':
        timePeriod = 6.5e-3
        timeInterval = 2e-6
        mdb.models['Model-1'].ExplicitDynamicsStep(name='Dynamic',
                                                   previous='Initial',
            timePeriod=timePeriod, nlgeom=ON)
    if Camadas == 'FRF':
        timePeriod = 6.5e-3
        timeInterval = 2e-6
        mdb.models['Model-1'].ExplicitDynamicsStep(name='Dynamic',
                                                   previous='Initial',
            timePeriod=timePeriod, nlgeom=ON)


session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Dynamic
                                    ')


##########################################################
# Loads and Boundary Conditions
##########################################################
if Mod == 1:
```

```
  #Velocidade de ensaio
  a = mdb.models['Model-1'].rootAssembly
  region1 = a.sets['Head_Sup']
  mdb.models['Model-1'].VelocityBC(name='Velo_Sup', createStepName='
                                    Dynamic',
      region=region1, v1=UNSET, v2=simu['Vel_tra'], v3=UNSET, vr1=UNSET,
                                     vr2=UNSET,
      vr3=UNSET, amplitude=UNSET, localCsys=None, distributionType=
                                    UNIFORM,
      fieldName='')
  #Engastamento da parte inferiora
  a = mdb.models['Model-1'].rootAssembly
  region = a.sets['Head_Inf']
  mdb.models['Model-1'].EncastreBC(name='Garra_Inferiora',
                                    createStepName='Dynamic',
      region=region, localCsys=None)
  #Engastamento da parte Superior
  a = mdb.models['Model-1'].rootAssembly
  region = a.sets['Head_Sup']
  mdb.models['Model-1'].DisplacementBC(name='Garra_Superiora',
                                    createStepName='Dynamic',
      region=region, u1=0.0, u2=UNSET, u3=0.0, ur1=0.0, ur2=0.0, ur3=0.0
                                    ,
      amplitude=UNSET, fixed=OFF, distributionType=UNIFORM, fieldName=''
                                    ,
      localCsys=None)


if Mod == 2:
  #Engastamento da parte inferiora
  a = mdb.models['Model-1'].rootAssembly
  region = a.sets['Reference_Point_Inferior']
  mdb.models['Model-1'].EncastreBC(name='Prato_Inferior', createStepName
                                    ='Dynamic',
      region=region, localCsys=None)
  #Velocidade de ensaio de compressao
  a = mdb.models['Model-1'].rootAssembly
  region1 = a.sets['Reference_Point_Superior']
  mdb.models['Model-1'].VelocityBC(name='Velo_Sup', createStepName='
                                    Dynamic',
      region=region1, v1=UNSET, v2=UNSET, v3=simu['Vel_comp'], vr1=UNSET
                                    , vr2=UNSET,
      vr3=UNSET, amplitude=UNSET, localCsys=None, distributionType=
                                    UNIFORM,
      fieldName='')
  #Restricoes de movimento de corpo rigido do disco superior:
  a = mdb.models['Model-1'].rootAssembly
```

```
  region1 = a.sets['Reference_Point_Superior']
  mdb.models['Model-1'].DisplacementBC(name='Rigid_Body_Sup',
                                       createStepName='Dynamic',
      region=region1, u1=0.0, u2=0.0, u3=UNSET, ur1=0.0, ur2=0.0, ur3=0.
                                           0,
      amplitude=UNSET, fixed=OFF, distributionType=UNIFORM, fieldName=''
                                           ,
      localCsys=None)

if Mod == 3:

    #Engaste
    a = mdb.models['Model-1'].rootAssembly
    region = a.sets['RP_Suporte']
    mdb.models['Model-1'].EncastreBC(name='BC-3', createStepName='
                                           Dynamic',
        region=region, localCsys=None)

    #Velocidade Inicial
    a = mdb.models['Model-1'].rootAssembly
    region = a.sets['RF_Ball']
    mdb.models['Model-1'].Velocity(name='Velocity_Ball', region=region,
                                           field='',
        distributionType=MAGNITUDE, velocity1=0.0, velocity2=0.0,
                                               velocity3=param_bola['
                                               Velocidade'],
        omega=0.0)

    #Restricoes de movimento da esfera de impacto:
    a = mdb.models['Model-1'].rootAssembly
    region1 = a.sets['RF_Ball']
    mdb.models['Model-1'].DisplacementBC(name='Rigid_Body_Ball',
                                           createStepName='Dynamic',
        region=region1, u1=0.0, u2=0.0, u3=UNSET, ur1=0.0, ur2=0.0, ur3=
                                               0.0,
        amplitude=UNSET, fixed=OFF, distributionType=UNIFORM, fieldName=
                                               '',
        localCsys=None)

    # #Gravidade
    mdb.models['Model-1'].Gravity(name='Gravidade', createStepName='
                                           Dynamic',
        comp3=9.8, distributionType=UNIFORM, field='')



##########################################################
```

```
# Mesh
######################################################
if Mod == 1:
  if param_trac['teste_mesh'] == True:
    #Faz uma malha simplificada somente para testes
    #Mesh global so pra teste
    p = mdb.models['Model-1'].parts['Meio']
    p.seedPart(size=simu['global_extremidades']/4., deviationFactor=0.1,
                                        minSizeFactor=0.1)
    p = mdb.models['Model-1'].parts['Meio']
    c = p.cells
    cells = c.getByBoundingBox()
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions, elemTypes=elements)
    p.generateMesh()

    p = mdb.models['Model-1'].parts['Superior']
    p.seedPart(size=simu['global_extremidades'], deviationFactor=0.1,
                                       minSizeFactor=0.1)
    p = mdb.models['Model-1'].parts['Superior']
    c = p.cells
    cells = c.getByBoundingBox()
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions, elemTypes=elements)
    p.generateMesh()

    p = mdb.models['Model-1'].parts['Inferior']
    p.seedPart(size=simu['global_extremidades'], deviationFactor=0.1,
                                       minSizeFactor=0.1)
    p = mdb.models['Model-1'].parts['Inferior']
    c = p.cells
    cells = c.getByBoundingBox()
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions, elemTypes=elements)
    p.generateMesh()




  else:
    #---------------Parte do Meio---------------#
    # SEED
    #Seed global
    p = mdb.models['Model-1'].parts['Meio']
```

```
p.seedPart(size=simu['global_meio'], deviationFactor=0.1,
                                    minSizeFactor=0.1)
#Seed locais
p = mdb.models['Model-1'].parts['Meio']
e = p.edges
pickedEdges = e.getByBoundingBox(zMin = param_trac['espessura'],
                                    yMin = 0.0, yMax = 0.00005)
p.seedEdgeByNumber(edges=pickedEdges, number=simu['big_number'],
                                    constraint=FINER)
p = mdb.models['Model-1'].parts['Meio']
e = p.edges
pickedEdges = e.getByBoundingBox(zMax = 0.0, yMin = 0.0, yMax = 0.
                                    00005)
p.seedEdgeByNumber(edges=pickedEdges, number=simu['big_number'],
                                    constraint=FINER)
p = mdb.models['Model-1'].parts['Meio']
e = p.edges
pickedEdges = e.getByBoundingBox(xMin = param_trac['dim'][1]/2. -
                                    param_trac['dim'][5]/2., yMin = 0.0,
                                     yMax = 0.00005)
p.seedEdgeByNumber(edges=pickedEdges, number=simu['medium_number'],
                                    constraint=FINER)
p = mdb.models['Model-1'].parts['Meio']
e = p.edges
pickedEdges = e.getByBoundingBox(xMax = -(param_trac['dim'][1]/2. -
                                    param_trac['dim'][5]/2.), yMin = 0.0
                                    , yMax = 0.00005)
p.seedEdgeByNumber(edges=pickedEdges, number=simu['medium_number'],
                                    constraint=FINER)
# MALHAGEM
p = mdb.models['Model-1'].parts['Meio']
c = p.cells
cells = c.getByBoundingBox()
pickedRegions =(cells, )
p.setElementType(regions=pickedRegions, elemTypes=elements)
p.generateMesh()

#---------------Parte Superior---------------#
# SEED
#Seed global
p = mdb.models['Model-1'].parts['Superior']
p.seedPart(size=simu['global_extremidades'], deviationFactor=0.1,
                                    minSizeFactor=0.1)
#Seed locais
p = mdb.models['Model-1'].parts['Superior']
e = p.edges
```

```
pickedEdges = e.getByBoundingBox(yMax = param_trac['dim'][2]/2. -
                                 param_trac['dim'][3] - param_trac['
                                 dim'][4])
p.seedEdgeByBias(biasMethod=SINGLE, end2Edges=pickedEdges, ratio=
                                 simu['ratio'],
    number=simu['small_number'], constraint=FINER)
p = mdb.models['Model-1'].parts['Superior']
e = p.edges
pickedEdges = e.getByBoundingBox(yMax = param_trac['dim'][2]/2. -
                                 param_trac['dim'][3] - param_trac['
                                 dim'][4] - param_trac['prolongacao']
                                 )
p.seedEdgeByNumber(edges=pickedEdges, number=simu['small_number']/2,
                                  constraint=FINER)
## MALHAGEM
p = mdb.models['Model-1'].parts['Superior']
c = p.cells
cells = c.getByBoundingBox()
pickedRegions =(cells, )
p.setElementType(regions=pickedRegions, elemTypes=elements)
p.generateMesh()

#---------------Parte Inferior---------------#
# SEED
#Seed global
p = mdb.models['Model-1'].parts['Inferior']
p.seedPart(size=simu['global_extremidades'], deviationFactor=0.1,
                                 minSizeFactor=0.1)
#Seed locais
p = mdb.models['Model-1'].parts['Inferior']
e = p.edges
pickedEdges = e.getByBoundingBox(yMin = -(param_trac['dim'][2]/2. -
                                 param_trac['dim'][3] - param_trac['
                                 dim'][4]))
p.seedEdgeByBias(biasMethod=SINGLE, end1Edges=pickedEdges, ratio=
                                 simu['ratio'],
    number=simu['small_number'], constraint=FINER)
p = mdb.models['Model-1'].parts['Inferior']
e = p.edges
pickedEdges = e.getByBoundingBox(yMin = -(param_trac['dim'][2]/2. -
                                 param_trac['dim'][3] - param_trac['
                                 dim'][4] - param_trac['prolongacao']
                                 ))
p.seedEdgeByNumber(edges=pickedEdges, number=simu['small_number']/2,
                                  constraint=FINER)
## MALHAGEM
p = mdb.models['Model-1'].parts['Inferior']
```

```
    c = p.cells
    cells = c.getByBoundingBox()
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions, elemTypes=elements)
    p.generateMesh()

if Mod == 2:
  p = mdb.models['Model-1'].parts['compressao']
  p.seedPart(size=simu['global_extremidades']/2., deviationFactor=0.1,
                                      minSizeFactor=0.1)
  p = mdb.models['Model-1'].parts['compressao']
  c = p.cells
  cells = c.getByBoundingBox()
  pickedRegions =(cells, )
  p.setElementType(regions=pickedRegions, elemTypes=elements)
  p.generateMesh()

  # #TAMANHO DA MALHA 0.003
  # #Set para o deslocamento radial
  # p = mdb.models['Model-1'].parts['compressao']
  # n = p.nodes
  # nodes = n.getSequenceFromMask(mask=('[#0 #40000 ]', ), )
  # p.Set(nodes=nodes, name='Disp_node')

  # p = mdb.models['Model-1'].parts['compressao']
  # n = p.nodes
  # nodes = n.getSequenceFromMask(mask=('[#40000 ]', ), )
  # p.Set(nodes=nodes, name='Disp_node_Otherside')

  # # #TAMANHO DA MALHA 0.0045
  p = mdb.models['Model-1'].parts['compressao']
  n = p.nodes
  nodes = n.getSequenceFromMask(mask=('[#0 #2 ]', ), )
  #nodes = n.getSequenceFromMask(mask=('[#0:43 #100000 ]', ), )
  p.Set(nodes=nodes, name='Disp_node')

  p = mdb.models['Model-1'].parts['compressao']
  n = p.nodes
  nodes = n.getSequenceFromMask(mask=('[#2000 ]', ), )
  p.Set(nodes=nodes, name='Disp_node_Otherside')




if Mod == 3:
    # Mesh do Osso
```

```
p = mdb.models['Model-1'].parts[param_osso['name']]
p.seedPart(size=simu['Mesh_Osso'], deviationFactor=0.1,
                                    minSizeFactor=0.1)
p = mdb.models['Model-1'].parts[param_osso['name']]
c = p.cells
cells = c.getByBoundingBox()
pickedRegions =(cells, )
p.setElementType(regions=pickedRegions, elemTypes=elements1)
p.generateMesh()

if Camadas == 'sEVA':

    #Mesh da Pele
    p = mdb.models['Model-1'].parts[param_pele['name']]
    p.seedPart(size=simu['Mesh_Pele'], deviationFactor=0.1,
                                        minSizeFactor=0.1)
    p = mdb.models['Model-1'].parts[param_pele['name']]
    c = p.cells
    cells = c.getByBoundingBox()
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions, elemTypes=elements2)
    p.generateMesh()

if Camadas == 'F':

    #Mesh da Pele
    p = mdb.models['Model-1'].parts[param_pele['name']]
    p.seedPart(size=simu['Mesh_Pele'], deviationFactor=0.1,
                                        minSizeFactor=0.1)
    p = mdb.models['Model-1'].parts[param_pele['name']]
    c = p.cells
    cells = c.getByBoundingBox()
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions, elemTypes=elements2)
    p.generateMesh()

    #Mesh da camada Flexivel
    p = mdb.models['Model-1'].parts[param_FLEX['name']]
    p.seedPart(size=simu['Mesh_Protetor_Flex'], deviationFactor=0.1,
                                        minSizeFactor=0.1)
    p = mdb.models['Model-1'].parts[param_FLEX['name']]
    c = p.cells
    cells = c.getByBoundingBox()
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions, elemTypes=elements3)
    p.generateMesh()
```

```
if Camadas == 'R':

    #Mesh da Pele
    p = mdb.models['Model-1'].parts[param_pele['name']]
    p.seedPart(size=simu['Mesh_Pele'], deviationFactor=0.1,
                                        minSizeFactor=0.1)
    p = mdb.models['Model-1'].parts[param_pele['name']]
    c = p.cells
    cells = c.getByBoundingBox()
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions, elemTypes=elements2)
    p.generateMesh()

    #Mesh da camada Rigida
    p = mdb.models['Model-1'].parts[param_RIG['name']]
    p.seedPart(size=simu['Mesh_Protetor_Flex'], deviationFactor=0.1,
                                         minSizeFactor=0.1)
    p = mdb.models['Model-1'].parts[param_RIG['name']]
    c = p.cells
    cells = c.getByBoundingBox()
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions, elemTypes=elements3)
    p.generateMesh()

if Camadas == 'FR':

    #Mesh da Pele
    p = mdb.models['Model-1'].parts[param_pele['name']]
    p.seedPart(size=simu['Mesh_Pele'], deviationFactor=0.1,
                                        minSizeFactor=0.1)
    p = mdb.models['Model-1'].parts[param_pele['name']]
    c = p.cells
    cells = c.getByBoundingBox()
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions, elemTypes=elements2)
    p.generateMesh()

    #Mesh da camada Rigida
    p = mdb.models['Model-1'].parts[param_RIG['name']]
    p.seedPart(size=simu['Mesh_Protetor_Flex'], deviationFactor=0.1,
                                         minSizeFactor=0.1)
    p = mdb.models['Model-1'].parts[param_RIG['name']]
    c = p.cells
    cells = c.getByBoundingBox()
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions, elemTypes=elements3)
    p.generateMesh()
```

```
        #Mesh da camada Flexivel
        p = mdb.models['Model-1'].parts[param_FLEX['name']]
        p.seedPart(size=simu['Mesh_Protetor_Flex'], deviationFactor=0.1,
                                                minSizeFactor=0.1)
        p = mdb.models['Model-1'].parts[param_FLEX['name']]
        c = p.cells
        cells = c.getByBoundingBox()
        pickedRegions =(cells, )
        p.setElementType(regions=pickedRegions, elemTypes=elements3)
        p.generateMesh()

    if Camadas == 'RFR':

        #Mesh da Pele
        p = mdb.models['Model-1'].parts[param_pele['name']]
        p.seedPart(size=simu['Mesh_Pele'], deviationFactor=0.1,
                                                minSizeFactor=0.1)
        p = mdb.models['Model-1'].parts[param_pele['name']]
        c = p.cells
        cells = c.getByBoundingBox()
        pickedRegions =(cells, )
        p.setElementType(regions=pickedRegions, elemTypes=elements2)
        p.generateMesh()

        #Mesh da camada Rigida
        p = mdb.models['Model-1'].parts[param_RIG['name']]
        p.seedPart(size=simu['Mesh_Protetor_Flex'], deviationFactor=0.1,
                                                minSizeFactor=0.1)
        p = mdb.models['Model-1'].parts[param_RIG['name']]
        c = p.cells
        cells = c.getByBoundingBox()
        pickedRegions =(cells, )
        p.setElementType(regions=pickedRegions, elemTypes=elements3)
        p.generateMesh()

        #Mesh da camada Flexivel
        p = mdb.models['Model-1'].parts[param_FLEX['name']]
        p.seedPart(size=simu['Mesh_Protetor_Flex'], deviationFactor=0.1,
                                                minSizeFactor=0.1)
        p = mdb.models['Model-1'].parts[param_FLEX['name']]
        c = p.cells
        cells = c.getByBoundingBox()
        pickedRegions =(cells, )
        p.setElementType(regions=pickedRegions, elemTypes=elements4)
        p.generateMesh()
```

```
        #Mesh da camada Rigida
        p = mdb.models['Model-1'].parts['RIG2']
        p.seedPart(size=simu['Mesh_Protetor_Flex'], deviationFactor=0.1,
                                                minSizeFactor=0.1)
        p = mdb.models['Model-1'].parts['RIG2']
        c = p.cells
        cells = c.getByBoundingBox()
        pickedRegions =(cells, )
        p.setElementType(regions=pickedRegions, elemTypes=elements5)
        p.generateMesh()


    if Camadas == 'FRF':

        #Mesh da Pele
        p = mdb.models['Model-1'].parts[param_pele['name']]
        p.seedPart(size=simu['Mesh_Pele'], deviationFactor=0.1,
                                               minSizeFactor=0.1)
        p = mdb.models['Model-1'].parts[param_pele['name']]
        c = p.cells
        cells = c.getByBoundingBox()
        pickedRegions =(cells, )
        p.setElementType(regions=pickedRegions, elemTypes=elements2)
        p.generateMesh()

        #Mesh da camada Rigida
        p = mdb.models['Model-1'].parts[param_RIG['name']]
        p.seedPart(size=simu['Mesh_Protetor_Flex'], deviationFactor=0.1,
                                                minSizeFactor=0.1)
        p = mdb.models['Model-1'].parts[param_RIG['name']]
        c = p.cells
        cells = c.getByBoundingBox()
        pickedRegions =(cells, )
        p.setElementType(regions=pickedRegions, elemTypes=elements3)
        p.generateMesh()

        #Mesh da camada Flexivel
        p = mdb.models['Model-1'].parts[param_FLEX['name']]
        p.seedPart(size=simu['Mesh_Protetor_Flex'], deviationFactor=0.1,
                                                minSizeFactor=0.1)
        p = mdb.models['Model-1'].parts[param_FLEX['name']]
        c = p.cells
        cells = c.getByBoundingBox()
        pickedRegions =(cells, )
        p.setElementType(regions=pickedRegions, elemTypes=elements4)
        p.generateMesh()
```

```
        #Mesh da camada Flexivel
        p = mdb.models['Model-1'].parts['FLEX2']
        p.seedPart(size=simu['Mesh_Protetor_Flex'], deviationFactor=0.1,
                                                minSizeFactor=0.1)
        p = mdb.models['Model-1'].parts['FLEX2']
        c = p.cells
        cells = c.getByBoundingBox()
        pickedRegions =(cells, )
        p.setElementType(regions=pickedRegions, elemTypes=elements5)
        p.generateMesh()


########################################################
# Outputs
########################################################

if Mod == 1:
  mdb.models['Model-1'].FieldOutputRequest(name='Output',
      createStepName='Dynamic', variables=('S', 'MISES', 'E', 'U', 'UT',
                                    'UR',
      'V', 'VT', 'VR', 'A', 'AT', 'AR', 'RBANG', 'RBROT', 'ENER','RF', '
                                    CSTRESS', 'DAMAGEC', 'DAMAGET', '
                                    DAMAGEFT', 'DAMAGEFC', 'DAMAGEMT',
      'DAMAGEMC', 'DAMAGESHR', 'SDEG', 'EVF', 'STATUS'),
      timeInterval=0.01)
if Mod == 2:

  if param_comp['quad'] == True:
    ## IMPLiCITO
    mdb.models['Model-1'].FieldOutputRequest(name='Output',
        createStepName='Dynamic', variables=('S', 'MISES', 'E', 'VE', 'U
                                    ', 'UT', 'V', 'VT', 'RF', 'RT'),
                                    timeInterval=timeInterval)

  if param_comp['quad'] == False:
    ##EXPLiCITO
    mdb.models['Model-1'].FieldOutputRequest(name='Output',
        createStepName='Dynamic', variables=('S', 'MISES', 'E', 'PE', '
                                    PEEQ',
      'PEEQT', 'PEEQMAX', 'U', 'UT', 'UR', 'V', 'VT', 'VR', 'RF', 'RT',
                                    'RM'),timeInterval=timeInterval)


  # History

  del mdb.models['Model-1'].fieldOutputRequests['F-Output-1']
  del mdb.models['Model-1'].historyOutputRequests['H-Output-1']
```

```
  regionDef=mdb.models['Model-1'].rootAssembly.sets['
                                    Reference_Point_Superior']
  mdb.models['Model-1'].HistoryOutputRequest(name='Output_Force',
      createStepName='Dynamic', variables=('U3', 'RF3'), timeInterval=
                                       timeInterval,
      region=regionDef, sectionPoints=DEFAULT, rebar=EXCLUDE)

  regionDef=mdb.models['Model-1'].rootAssembly.allInstances['
                                    Assembly_Corpo_de_Prova'].sets['
                                    Disp_node']
  mdb.models['Model-1'].HistoryOutputRequest(name='Output_Disp',
      createStepName='Dynamic', variables=('MISES', 'U1', 'U2'),
      timeInterval=timeInterval, region=regionDef, sectionPoints=DEFAULT
                                    , rebar=EXCLUDE)

  regionDef=mdb.models['Model-1'].rootAssembly.allInstances['
                                    Assembly_Corpo_de_Prova'].sets['
                                    Disp_node_Otherside']
  mdb.models['Model-1'].HistoryOutputRequest(name='Output_Disp_Otherside
                                    ',
      createStepName='Dynamic', variables=('MISES', 'U1', 'U2'),
      timeInterval=timeInterval, region=regionDef, sectionPoints=DEFAULT
                                    , rebar=EXCLUDE)

if Mod == 3:

    if param_FLEX['quad'] == True:
        ## IMPLiCITO
        mdb.models['Model-1'].FieldOutputRequest(name='Output',
            createStepName='Dynamic', variables=('S', 'MISES', 'E', 'VE'
                                        , 'U', 'UT', 'V', 'VT',
                                        'RF', 'RT'),timeInterval
                                        =timeInterval)

    if param_FLEX['quad'] == False:
        ##EXPLiCITO
        mdb.models['Model-1'].FieldOutputRequest(name='Output',
            createStepName='Dynamic', variables=('S', 'MISES', 'E', 'PE'
                                        , 'PEEQ',
            'PEEQT', 'PEEQMAX', 'U', 'UT', 'UR', 'V', 'VT', 'A', 'AT', '
                                        VR', 'RF', 'RT', 'RM'),
                                        timeInterval=
                                        timeInterval)
```

```
    # History

    del mdb.models['Model-1'].fieldOutputRequests['F-Output-1']
    del mdb.models['Model-1'].historyOutputRequests['H-Output-1']




########################################
# SIMULATION
########################################
#Criacao do job

jobname='Simul'
mdb.Job(name=jobname, model='Model-1', description='', type=ANALYSIS,
  atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90,
  memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
  explicitPrecision=DOUBLE, nodalOutputPrecision=FULL, echoPrint=OFF,
  modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='',
  scratch='', resultsFormat=ODB, multiprocessingMode=DEFAULT, numCpus=1,
  numGPUs=0)

mdb.jobs[jobname].submit(consistencyChecking=OFF)
mdb.jobs[jobname].waitForCompletion()

#
                                    ----------------------------------------------------------------

#%% RESULTS
#
                                    ----------------------------------------------------------------


o3 = session.openOdb(name=os.getcwd()+'\\Simul.odb')

session.viewports['Viewport: 1'].setValues(displayedObject=o3)
session.viewports['Viewport: 1'].odbDisplay.setFrame(step=0, frame=simu[
                                    'npas'])

if Mod == 2:

    o3 = session.openOdb(name=os.getcwd()+'\\Simul.odb')

    session.viewports['Viewport: 1'].setValues(displayedObject=o3)
```

```
session.viewports['Viewport: 1'].odbDisplay.setFrame(step=0, frame=
                                        simu['npas'])


Force = []
TimeF =[]
Disp = []
TimeD = []

Erro_F = 0
Erro_D = 0

# odb = session.odbs[os.getcwd()+'/Simul.odb']
# xyList = xyPlot.xyDataListFromField(odb=odb, outputPosition=NODAL,
                                        variable=((
#     'RF', NODAL, ((COMPONENT, 'RF3'), )), ), nodeSets=(
#     'REFERENCE_POINT_ASSEMBLY_SUPERIOR      333', ))

odb = session.odbs[os.getcwd()+'/Simul.odb']
xyList = xyPlot.xyDataListFromField(odb=odb, outputPosition=NODAL,
                                        variable=((
    'RF', NODAL, ((COMPONENT, 'RF3'), )), ), nodeSets=(
    'REFERENCE_POINT_ASSEMBLY_SUPERIOR      147', ))

param_F = min(len(xyList[0].data), len(TimeF_Ref), len(Force_Ref))

#Guardar os resultados da simulacao e fazer o erro absoluto
soma_F = 0.0
Media_F = 0.0
for i in range(param_F):
  TimeF.append(xyList[0].data[i][0])
  Force.append(xyList[0].data[i][1])
  Erro_F = Erro_F + abs(Force[i] - Force_Ref[i])
  soma_F = soma_F + Force[i]
Media_F = soma_F/param_F
#Calculo do coeficiente R^2 para a forca
r2_F = 0.0
SQexp = 0.0
SQtot = 0.0
for i in range(param_F):
  SQexp = (Force[i] - Media_F)**2.
  SQtot = (Force_Ref[i] - Media_F)**2.
r2_F = SQexp/SQtot



odb = session.odbs[os.getcwd()+'/Simul.odb']
```

```
xyList = xyPlot.xyDataListFromField(odb=odb, outputPosition=NODAL,
                                    variable=((
    'U', NODAL, ((INVARIANT, 'Magnitude'), )), ), nodeSets=(
    'ASSEMBLY_CORPO_DE_PROVA.DISP_NODE', ))
xyList_Otherside = xyPlot.xyDataListFromField(odb=odb,
                                    outputPosition=NODAL, variable=
                                    ((
    'U', NODAL, ((INVARIANT, 'Magnitude'), )), ), nodeSets=(
    'ASSEMBLY_CORPO_DE_PROVA.DISP_NODE_OTHERSIDE', ))

param_D = min(len(xyList[0].data), len(TimeD_Ref), len(
                                    xyList_Otherside[0].data), len(
                                    Disp_Ref))

#Guardar os resultados da simulacao e fazer o erro absoluto
soma_D = 0.0
Media_D = 0.0
for j in range(param_D):
  TimeD.append(xyList[0].data[j][0])
  aux = abs(xyList[0].data[j][1]) + abs(xyList_Otherside[0].data[j][
                                    1] + 0.03)
  Disp.append(aux)
  Erro_D = Erro_D + abs(Disp[j] - Disp_Ref[j])
  soma_D = soma_D + Disp[j]
Media_D = soma_D/param_D
#Calculo do coeficiente R^2 para a forca
r2_D = 0.0
SQexp = 0.0
SQtot = 0.0
for i in range(param_D):
  SQexp = (Disp[i] - Media_D)**2.
  SQtot = (Disp_Ref[i] - Media_D)**2.
r2_D = SQexp/SQtot


#Criacao do arquivo .txt para salvar os resultados da simulacao
Resultados = open(os.getcwd()+'/Resultados.txt','w')
Resultados.write('Resultados da Otimizacao: \n\n')
Resultados.write('Somatoria do Erro da Forca (Erro_F) : {:8.4f} \n\n
                                    '.format(Erro_F))
Resultados.write('Somatoria do Erro do deslocamento radial (Erro_D)
                                    : {:8.4f} \n\n'.format(Erro_D))
Resultados.write('Coeficiente R2 para a Forca : {:8.4f} \n\n'.format
                                    (r2_F))
Resultados.write('Coeficiente R2 para o Deslocamento radial : {:8.4f
                                    } \n\n'.format(r2_D))
```

```
    Resultados.write('Densidade (rho) : {:8.4f} kg/m^3 \n'.format(
                                        mat_EVA['dens']))
    Resultados.write('Modulo de Young (E) : {:8.4f} Pa \n'.format(
                                        mat_EVA['E']))
    Resultados.write('Coeficiente de Poisson (v) : {:8.4f}  \n'.format(
                                        mat_EVA['nu']))
    Resultados.write('Limite elastico (sigmaY) : {:8.4f} Pa \n'.format(
                                        mat_EVA['Re']))
    Resultados.close()
```