Universidade de Aveiro Departamento de Engenharia Mecânica

2013

André Gomes de Pinho

Modelação e otimização de ferramentas de conformação plástica de chapa

Universidade de Aveiro

Departamento de Engenharia Mecânica

Ano 2013

André Gomes de Pinho

Modelação e otimização de ferramentas de conformação plástica de chapa

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob a orientação científica do Doutor António Gil d'Orey de Andrade Campos, Professor Auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro e do Doutor Robertt Angelo Fontes Valente, Professor Auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro

O júri

Presidente

Prof. Doutor Ricardo José de Sousa
Professor Auxiliar, Universidade de Aveiro

Vogais

Prof. Doutor Nuno Filipe Ferreira Soares Borges Lopes
Professor Auxiliar, Universidade de Aveiro

Prof. Doutor António Gil d'Orey de Andrade Campos
Professor Auxiliar, Universidade de Aveiro

Agradecimentos

Ao Professor Doutor António Gil D'Orey Andrade Campos e ao Professor Doutor Robertt Angelo Fontes Valente pela oportunidade de trabalhar com eles, pela grande orientação e ajuda ao longo deste trabalho.

À minha família, pela paciência e apoio desmedido que me deram em todo este percurso.

À Márcia, pela paciência, apoio e enorme ajuda que me deu em cada momento.

Aos meus amigos, por todos os momentos que me proporcionaram.

Palavras-chave

conformaçao plástica, chapas metálicas, simulaçao numérica, Método dos Elementos Finitos, optimização

Resumo

Atualmente, os programas de simulação pelo Método dos Elementos Finitos atingiram maturidade e confiança por parte das indústrias de conformação plástica de chapas metálicas para a previsão de defeitos durante os processos de deformação. Adicionalmente, para o desenvolvimento de ferramentas de estampagem, usam-se estes programas em metodologias do tipo tentativa-erro, o que se revela longo e moroso. Contudo, devido a utilização interligada do Método dos Elementos Finitos com algoritmos de otimização, é possível a resolução de problemas inversos, os quais obtêm a geometria inicial da ferramenta que leva à forma requerida da peça deformada.

O objetivo principal desta dissertação é o estudo e desenvolvimento de uma metodologia automática para encontrar as ferramentas ideais de conformação plástica de chapas metálicas. Para tal, numa primeira parte estuda-se e apresenta-se um método de criação de uma simulação numérica, utilizando uma linguagem de programação que permitem a criação parametrizada de geometrias em problemas estruturais e tecnológicos. Também nesta fase surge a criação do acoplamento da simulação numérica a um algoritmo de otimização baseado no gradiente conjugado. Numa segunda fase, aplica-se a metodologia a problemas mais complexos. Estes consistem na criação das geometrias iniciais de ferramentas para obter peças com a forma requerida.

Keywords Sheet metal forming, numerical simulacion, finite element method, optimizacion

Abstract

Currently, simulation programs by Finite Element Method (FEM) have reached the maturity and confidence of metal forming sheet metal industries for predicting defects during deformation processes. However, the development of stamping tools is still a trial-and-error methodology, which proves long and tedious. Considering the coupling of Finite Element Method with optimization algorithms, it is possible to solve geometry inverse problems, whose main goal is to obtain the initial geometry of the tool that leads to the required shape of the deformed part.

The main objective of this thesis is the study and development of an automated methodology to develop tools for metal forming sheet in order to compensate phenomena such as springback. To this end, in a first part, a method of creating a numerical simulation using *script* in Python programming language that allows fully parameterized geometries is presented. Subsequently, a numerical optimization algorithm based on conjugate gradient is coupled with the FEM parameterized simulation. In a second phase, the presented methodology is applied to industrial problems. The solutions obtained, in the form of ideal tool shape, validate the presented methodologies

Índice

Lista de Figuras

Lista de Tabelas

I Enquadramento

1. Preâmbulo

1.1. Introdução

No processo tecnológico de conformação plástica produz-se uma peça de geometria desejada através de uma peça de geometria simples. Este processo é executado através da aplicação de forças externas, as quais propiciam a geometria final desejada através da deformação do material inicial durante o contacto material-ferramenta [1, 2].

Nas últimas décadas tem existido investigação e desenvolvimento ativos nestes processos. Estes caracterizam-se pela otimização da matéria-prima, com o objetivo de se obter de forma eficiente componentes com geometrias complexas. A conformação plástica tem um lugar importante nas indústrias de transformação, devido a esta promover elevadas cadências de produção [1, 3]. Algumas destas indústrias são as seguintes [1–5]:

- Indústria automóvel (por exemplo painéis de carroçaria e tanques de combustível);
- Indústria de eletrodomésticos (por exemplo máquinas de lavar e fogões);
- Indústria de elementos domésticos e decorativos (por exemplo lava-loiças e botijas de gás);
- Indústria elétrica e eletrónica (por exemplo elementos de interruptores e de computadores e casquilhos de lâmpadas);
- Indústria de utensílios alimentares (por exemplo panelas e tabuleiros);
- Indústria aeronáutica e aeroespacial;
- Indústria naval;
- Indústria relojoeira.

Estas indústrias têm tendências caracterizadas pela flexibilidade e complexidade dos produtos, que são provocadas pelas exigências do mercado. A redução da vida útil dos produtos e a competição constante necessita do desenvolvimento cada vez mais rápido e económico de produtos de elevada

qualidade e complexidade e, ao mesmo tempo, deve existir flexibilidade na alteração a nível do *design* dos mesmos. Implicando assim, o desenvolvimento de técnicas e ferramentas de forma a obter o produto desejado [4, 6].

Entre as diferentes indústrias, a indústria automóvel destaca-se devido aos grandes volumes de produção e à grande variedade dos componentes embutidos. Esta indústria torna-se num dos principais acionadores do desenvolvimento de diferentes áreas como, por exemplo, a conformação plástica, causada pela forte concorrência dos produtores e a sua grande importância económica nos países desenvolvidos [3–5]. Salienta-se que foi no século XX que se deu o desenvolvimento da indústria e no século XXI continua a ocupar um lugar muito importante [3, 5, 7].

A proteção ambiental em conjunto com a economia de combustível e a segurança é neste momento uma das principais preocupações da indústria ambiental. Para tal, a redução do peso dos automóveis, para um melhor desempenho do veículo, em conjunto com uma redução de combustível e o melhoramento das estruturas proporcionam uma resposta melhor ao choque, e assim, maior segurança. Estas são as principais estratégias a ter em conta. Para atingir este objetivo, uma das medidas atualmente efetuadas é a utilização de materiais mais leves e mais resistentes, nomeadamente aços macios, aços de elevada resistência e ligas de alumínio [3–5].

De acordo com o mencionado anteriormente e o contínuo aumento da complexidade da conformação plástica ao longo dos anos levaram a que houvesse evolução na conformação plástica, a qual provocou a redução dos métodos de tentativa-e-erro baseados no conhecimento e experiência do projetista. Assim, ocorreu o aumento do uso da produção virtual, tal como é o caso da simulação numérica dos processos de conformação plástica e à sua aplicação na indústria automóvel [3–5, 8]; isto levou a uma redução do tempo de desenvolvimento e ajudou à introdução de novos materiais [4].

Para se dar uma boa resposta à crescente utilização da simulação numérica, um grande esforço tem incidido no desenvolvimento de ferramentas numéricas fiáveis e modelos matemáticos para a simulação dos processos de conformação plástica [4]. Com a evolução a este nível, a evolução na conformação plástica, mencionada anteriormente, ainda se tornou mais notória e possibilitou a execução de problemas cada vez mais complexos. A conceção

assistida por computador e a fabricação assistida por computador, das siglas inglesas CAD e CAM, respetivamente, deram origem a uma muito maior interatividade entre a conceção e o projeto de ferramentas e componentes de conformação, provocando assim, uma redução no número de ciclos de tentativa-e-erro através da execução de ciclos virtuais. Este feito permitiu um desenvolvimento acentuado nas áreas nas quais foram utilizadas em conjunto com uma redução de tempos de fabrico [3, 5].

A evolução computacional e dos métodos de resolução de problemas levaram à simulação da conformação plástica com boa precisão e com tempos de execução aceitáveis em peças com elevada complexidade [3, 5, 9]. Este tipo de simulação continua muito atual, o que levou ao aparecimento de inúmeros programas de computador que se diferenciam, entre várias coisas, pelo tempo de utilização, facilidade de utilização e qualidade dos resultados [3, 5, 10].

A simulação da conformação de chapas metálicas é usada em inúmeras situações, entre as quais os problemas em que se pretende conhecer [3, 5, 11]:

- O fluxo do material a ser conformado,
- A análise de tensões e deformações, para determinar forças de conformação,
- A previsão de defeitos ou falhas, para melhorar a qualidade do produto final e redução do custo do mesmo.

Entre todas estas aplicações também se deve ter em conta a crescente utilização de processos de otimização de ferramentas e controlo de processos. Permitindo assim a redução do desperdício do material e ao mesmo tempo obter a qualidade e resistência mecânicas desejadas [3, 5].

As propriedades dos materiais e as suas leis constitutivas, tal como as condições de atrito, são fatores e parâmetros de grande influência no resultado final da simulação. Contudo, as formas geométricas das ferramentas e o tempo computacional têm também que ser considerados no custo efetivo [3].

Com tudo isto, a simulação nos últimos anos da década passada conseguiu reduzir o tempo de produção em 50% prevendo-se nesta década que chegue a uma redução adicional de 30% [3, 5, 10].

Existem muitas aplicações para a simulação na conformação plástica. Entre elas a estampagem, que é uma técnica de conformação de chapas metálicas que consiste na obtenção de uma peça tridimensional através da imposição de deformação plástica numa chapa fina [1, 3, 4, 12]. Pode formar-se através de elevadas cadências de produção uma grande variedade de peças de superfície, estas com boas propriedades mecânicas e uma boa precisão dimensional [1–4].

Para a execução de uma estampagem geralmente necessita-se de 4 elementos principais: chapa, matriz, punção e cerra-chapas tal como se pode observar na Figura 1 [1, 3, 4].

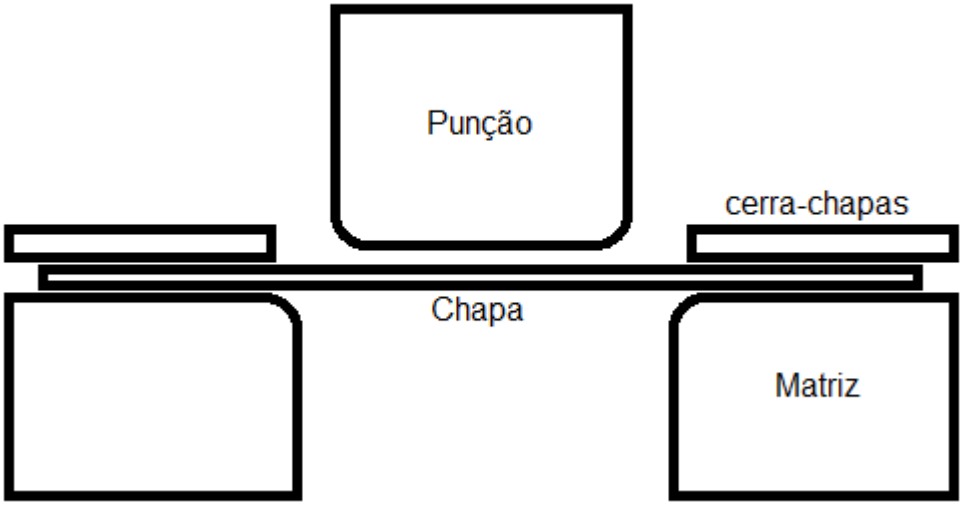

Figura 1 – Representação dos elementos de estampagem.

Pode-se dividir o processo de estampagem em 4 fases principais executadas pela seguinte ordem [3, 4]:

- Colocar e alinhar o esboço (chapa metálica com dimensões previamente definidas);
- Descer o cerra-chapas para impedir que o esboço se mova livremente
- Desce o punção o qual cria uma deformação da chapa no interior da matriz para a obtenção da forma desejada;
- Elevar o punção seguido do cerra-chapas;

Nalguns casos ainda se necessita de uma fase adicional para executar alguns acabamentos. Entre os quais, pode-se referir o caso em que se retira a

gola ou rebordo, os quais são formados por excedentes de chapa que não são necessários para a peça final [3].

Na estampagem existem alguns defeitos que se formam durante este processo. Entre eles, os mais frequentes são originários em falhas no projeto, construção das ferramentas, manutenção das mesmas e em defeitos pré existentes na chapa. Estes podem ser denominados por [1, 3]:

- Retorno elástico – *springback*;
- Formação de rugas/pregas – *wrinkiling*;
- Formação de "orelhas" – *earing*;
- Redução excessiva de espessura – *thinning*;
- Roturas ou fissuras;

Todos estes defeitos nomeados anteriormente podem ser agrupados em 3 classes distintas [1, 3, 4, 13]:

- Defeitos de forma ou dimensionamento;
- Defeitos na peça ou na superfície;
- Propriedades mecânicas finais da peça não satisfatórias;

Para evitar-se o aparecimento de defeitos e para obter-se uma boa qualidade final da peça, tem que se ter em conta certas variáveis tais como [3, 4, 14]:

- Medição: por exemplo método de medição, forças, deslocamento do punção;
- Condições de operação: por exemplo precisão na localização do esboço, força de embutidura, velocidade de conformação, lubrificante, temperatura;
- Prensa: por exemplo tipo de prensa, precisão da prensa;
- Controlo das ferramentas: por exemplo método de transporte, procedimento de manutenção;
- Ferramentas: por exemplo tratamentos superficiais, precisão da ferramenta, freio, rigidez;
- Material do esboço: por envelhecimento, tratamentos superficiais dimensões da chapa, propriedades mecânicas da chapa, diferenças na bobine, método de corte de esboços.

Devido à complexidade e interatividade destas variáveis é difícil tentar estudar o peso de cada uma individualmente. No entanto, se fosse possível obter a informação sobre todos estes parâmetros podia prever-se os defeitos no processo de estampagem, permitindo, assim, diminuir o custo total e reduzir o tempo de desenvolvimento. O problema é que obter esta informação é uma tarefa difícil e morosa [1, 3, 4].

1.2. Objetivos e desafios deste trabalho

O presente trabalho tem como objetivo o desenvolvimento de competências nas áreas da simulação numérica e otimização, com particular foco no desenvolvimento e estudo de uma metodologia automática para o cálculo de ferramentas de conformação plástica de chapas metálicas. Assim, as principais metas deste trabalho são:

- Estudar problemas de simulação que utilizam o Método dos Elementos Finitos na área de conformação plástica em chapas metálicas;

- Estudar e desenvolver uma metodologia baseada em *scripts* utilizando a linguagem Python para a definição do problema de simulação numérica em ABAQUS, tal como a definição e parametrização da geometria deste problema;

- Estudar e desenvolver uma subrotina em Fortran para o acoplamento do problema de simulação numérica com o algoritmo de otimização.

1.3. Guia de Leitura

O presente trabalho está dividido em quatro partes, compostas por sete capítulos.

No enquadramento, a primeira parte, apresenta-se as descrições gerais do processo estudado neste trabalho, as quais são expostas no primeiro capítulo.

Capítulo 1 – Descrição resumida do que é a conformação plástica de chapas metálicas, bem como a apresentação dos objetivos e o guia de leitura.

Na segunda parte, denominada por metodologia, descrevem-se as características gerais da simulação numérica e da otimização aplicadas neste trabalho, expondo-se também um exemplo, como é explicado nos seguintes capítulos:

Capítulo 2 – Apresentação sumária do Método dos Elementos Finitos, tal como o programa utilizado (ABAQUS), e por último, exposição de um exemplo da metodologia aplicada para a simulação numérica.

Capítulo 3 – Descrição resumida dos conceitos de otimização, exposição do método e do programa utilizado, terminando o capítulo com a apresentação de um exemplo da metodologia aplicada para o acoplamento do problema exposto no capítulo anterior ao algoritmo de otimização.

Na terceira parte, intitulada “aplicações numéricas”, apresentam-se os problemas de simulação e otimização estudados, tal como é exposto nos seguintes capítulos:

Capítulo 4 – Descrição das características gerais dos casos de estudo apresentados no presente trabalho.

Capítulo 5 – Explicação dos *scripts* em Python para a definição da simulação numérica em ABAQUS de cada caso de estudo.

Capítulo 6 – Exposição dos *scripts* em Python e da subrotina em Fortran para o acoplamento da simulação numérica no algoritmo de otimização.

Na quarta parte, discussão, apresentam-se de forma resumida as conclusões do presente trabalho:

Capítulo 7 – Descrição das conclusões gerais, bem como a apresentação de algumas sugestões para a execução de trabalhos futuros que possam completar esta dissertação.

II Metodologia

2. Simulação Numérica

2.1. Introdução

O método de elementos finitos é cada vez mais utilizado em toda a engenharia, como por exemplo no estudo de transferência de calor, fluidos, estruturas e sólidos [1, 3, 15].

Nos anos 30 foi quando se desenvolveram os primeiros trabalhos de conformação plástica de chapas, estes realizados por Sachs [1, 3, 16]. Embora só a partir dos anos 80, devido ao grande desenvolvimento dos sistemas informáticos, em conjunto com o desenvolvimento dos métodos numéricos, foi possível a uma simulação de peças com elevada complexidade [1, 9, 17].

As não linearidades dos processos de estampagem relativas aos materiais (por causa do comportamento anisotrópico e a característica inelástica da deformação), a geometria (relativas a nesta existir elevados deslocamentos, rotações, deformações e extensões) e as interações entre corpos (estas referem-se ao atrito e contacto entre as chapas e ferramentas as quais variáveis no decorrer do processo) levam a que a simulação numérica deste processo apresente uma grande complexidade [1].

Atualmente, na indústria de produção de peças embutidas, a simulação numérica pelo Método dos Elementos Finitos tem uma elevada importância, pois tem a capacidade de prever com alguma credibilidade a rotura, o enrugamento, a distribuição de deformações e o retorno elástico [4].

A previsão da geometria das ferramentas e de parâmetros do processo para a diminuição do retorno elástico é difícil e muito estudada. Contudo, para a previsão do retorno elástico é necessário a previsão da distribuição das tensões, a qual é influenciada por um vasto numero de variáveis que podem ser divididas em: propriedades do material, geometria do componente, condições do processo e variáveis numéricas [4, 18].

Para a obtenção de bons resultados na simulação tem que se ter em conta como [1, 3]:

- Definir os elementos finitos a utilizar.

- Como se formula as ferramentas.
- Modelar o comportamento mecânicos da chapa.
- Definir as interações mecânicas entre chapa/ferramenta.
- Caracterizar a formabilidade do material.

A produção de programas de simulação numérica que obtenham resultados realistas e precisos em tempo razoável e com custos computacionais aceitáveis é um dos principais objetivos da investigação nesta área. Com este tipo de programas deve ser possível prever defeitos geométricos (assim evitando erros de projeto poupando custos) e otimizar a geometria da peça e das ferramentas antes de qualquer tipo de investimento físico [1, 3].

2.2. Fundamentos sobre o Método dos Elementos Finitos (MEF)

O Método dos Elementos Finitos é um método numérico utilizado para a obtenção de respostas a problemas demasiado complexos para se obter uma solução exata [3, 19].

A aplicação deste método necessita de alguma disponibilidade computacional. Por esta razão, o desenvolvimento contínuo da tecnologia informática potencia ainda mais o Método dos Elementos Finitos [3, 19]. Nos tempos correntes este conjunto tornou-se ainda mais indispensável devido a proporcionar uma elevada rapidez e baixo custo no desenvolvimento tecnológico e na investigação [1].

O Método dos Elementos Finitos consiste numa metodologia de discretização a qual consiste em dividir o domínio (o qual pode ser em duas dimensões ou em três dimensões) a ser estudado em vários subdomínios mais pequenos adjacentes uns aos outros obtendo os denominados elementos finitos (os quais em duas dimensões geralmente são quadriláteros ou triângulos e em três dimensões são hexaedros, tetraedros ou pentaedros) [1, 19].

A obtenção do estado de tensão e de deformação em componentes sujeitos a carregamentos mecânicos diversos foi um dos primeiros casos onde foi aplicado o Método dos Elementos Finitos [3].

Com base neste caso, a aplicação do Método dos Elementos Finitos é feita da forma como se ira explicar nos seguintes tópicos:

- Obtém-se os campos dos deslocamentos para um número de nós da malha dos elementos finitos (estes são pontos que geralmente se encontram normalmente nos vértices dos elementos finitos, mas também se podem encontrar no meio das arestas ou das faces, e até no interior do elemento), podendo-se calcular o deslocamento de qualquer ponto de um determinado elemento finito em função dos deslocamentos nodais desses elementos. E assim, é possível, no final, o cálculo do deslocamento de qualquer ponto nos domínios [19]. A passagem do elemento finito para o domínio global também pode ser denominada por agrupamento ou assemblagem [1].
- Por conseguinte com o calculado anteriormente calcula-se as deformações correspondentes a cada deslocamento nodal e com estas o estado de tensão [19].
- Uma boa qualidade dos resultados do estado de tensão e de deformação através deste método depende em grande parte da forma como o utilizador implementa a geometria do sistema a estudar, a malha a utilizar, as propriedades dos materiais e definição dos carregamentos e das restrições (também denominadas por condições de fronteira) a que o sistema está sujeito [19].
- No final, deve-se executar a análise cuidada dos resultados de modo a diminuir os erros, os quais são provocados devido a este método obter valores aproximados [1].

2.3. Programa de simulação ABAQUS

No programa ABAQUS[1] as equações de equilíbrio são obtidas pelo princípio dos trabalhos virtuais e cada ponto material é função das suas coordenadas e do tempo. Este programa usa o Método dos Elementos Finitos

[1] ABAQUS deriva da palavra grega Aba-kala-culus, o que significa a memória da solução [1]. Este programa usa o Método dos Elementos Finitos para resolver diversos problemas de engenharia. Atualmente a Dassault Systemes SA mais especificamente uma das suas marcas a SIMULIA tem direitos sobre a comercialização deste programa [1].

do deslocamento, tendo como base a aproximação das condições de equilíbrio de um determinado corpo através do ponto de vista Lagrangiano [3, 15].

O utilizador no pré-processamento pode utilizar com alguma facilidade o módulo gráfico ABAQUS/CAE para introduzir as características da peça a estudar. Este cria através destes dados um ficheiro de entrada na linguagem ASCII e uma representação gráfica dos resultados do pós-processamento a qual pode ser observada no ABAQUS/Viewer [1, 3, 15].

No ABAQUS/CAE o utilizador define as características de entrada do problema em estudo, tais como [1, 3, 15]:

- A geometria da estrutura,
- As características do material,
- Os carregamentos,
- As condições fronteira,
- As interações entre os vários componentes,
- As etapas da análise e natureza (linear ou não-linear)
- A discretização da estrutura em elementos finitos

Estas características também podem ser criadas ou manipuladas através do ficheiro de entrada ASCII [1].

A análise pode ser de dois tipos [1, 3, 15]:

- Utilizando o ABAQUS/Standard, o qual consiste numa análise com integração implícita,
- Utilizando o ABAQUS/Explicit utilizando a integração explícita para a resolução de problemas quasi-estáticos e não-lineares dinâmicos.

Em resumo, através deste programa pode-se analisar os problemas de engenharia mais complexos, com as suas características e obter os resultados com a analise cuidada dos mesmos.

2.4. Exemplo de metodologia utilizando *script*

Utilizam-se *scripts*[2] para inserir os dados relativos ao pré-processamento e pós-processamento no ABAQUS. Este é um meio de definir todas as características de entrada do problema em estudo. A linguagem utilizada para definir *scripts* no presente trabalho é a Python.

Esta linguagem de programação é aconselhada pelo Abaqus Analysis User's Manual e é considerada de fácil aprendizagem e uso. Além disso tem a particularidade de ser fácil de ler e escrever, advindo desta linguagem ser orientada por objetos. Outra grande vantagem desta linguagem é a sua compatibilidade com vários sistemas operativos, sendo unicamente necessário copiar e colar o respetivo ficheiro de um sistema para outro.

Antes do início da criação do *script* deve-se saber todas as características do problema. Irá usar-se, a título de exemplo, um caso simples de uma viga encastrada com a aplicação de uma carga vertical, tal como se pode observar na Figura 2. Algumas das características deste problema podem ser alteradas, como as dimensões e características do material, a intensidade do carregamento.

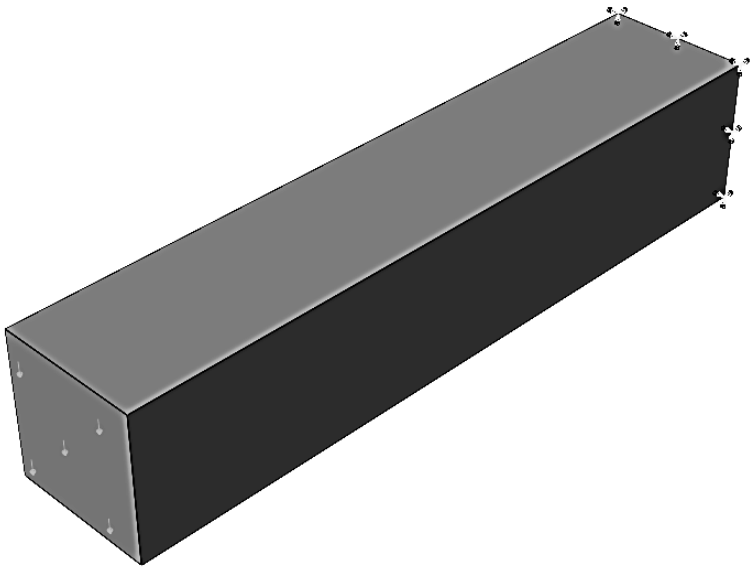

Figura 2 – Representação de uma viga encastrada com uma tensão numa face.

[2] *scripts* são listagem de código de linguagem de programação que podem ser executadas dentro de programas não se restringindo a esse ambiente.

O material utilizado para a viga é um aço cujas dimensões são 10x10x50 mm e a carga utilizada é de 30 N.

Inicialmente, são importadas todas as funções que serão *à posteriori* utlizadas. O código é dado por

```
session.journalOptions.setValues(replayGeometry=COORDINA
TE, recoverGeometry=COORDINATE)

from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
from abaqusConstants import *
```

Pode-se observar seguidamente a definição das variáveis através de código. Estas variáveis podem estar no próprio ficheiro ou noutro separadamente para permitir sua alteração de uma maneira mais fácil. Por simplificação foi inserida no próprio ficheiro.

```
#Variables

C=50.0 #comprimento
L=10.0 #largura
A=10.0 #altura
```

```
M=209.E3 #modulo de young
V=0.3 #coeficiente de poisson
F=30.0 #forca aplicada
P=F/(L*A) #pressao aplicada
l=L/2 #coordenada x
a=A/2 #coordenada y
```

Depois destes pormenores definidos, segue-se para o pré-processamento, iniciado pela criação do modelo.

```
#Model

model=mdb.Model(name='Model')
```

Segue-se a definição da peça a estudar e do seu esboço, o qual consiste num retângulo de dimensões 10x10 mm já anteriormente referidas.

```
#Sketch

sketch=model.Sketch(name='Sketch', sheetSize=250.)
sketch.rectangle(point1=(-l,-a), point2=(l,a))
```

Posteriormente cria-se a peça, definindo que se está a trabalhar em três dimensões e que o tipo da peça em estudo é um objeto deformável. Depois disto, utiliza-se o esboço criado anteriormente e aplica-se uma extrusão.

```
#Part

part=model.Part(name='Part', dimensionality=THREE_D,
type=DEFORMABLE_BODY)
part.BaseSolidExtrude(sketch=sketch, depth=C)
```

Continua-se para a criação do material com as suas características através dos dados referidos anteriormente.

```
#Material

material=model.Material(name='Material')
material.Elastic(table=((M,V),))
```

Segue-se para a criação da secção, com o anexar do material e posterior atribuição desta à parte já definida anteriormente.

```
#Section

section=model.HomogeneousSolidSection(name='Section',
material='Material', thickness=1.0)
sectionregion=(part.cells,)
part.SectionAssignment(region=sectionregion,
sectionName='Section')
```

Após a definição da secção segue-se a assemblagem, na qual se define os eixos de coordenadas e a dependência de cada peça à sua malha. Neste exemplo a malha é dependente para cada parte da geometria da peça.

```
#Assembly

assembly=model.rootAssembly
instance=assembly.Instance(name='Instance', part=part,
dependent=ON)
```

Passa-se para a criação da malha. Na qual, define-se o número de elementos em cada aresta e escolhe-se o formato de cada elemento finito, tal como a técnica de aplicação da malha, o tipo de elemento e o tipo de análise em cada elemento.

```
#Mesh

part.setMeshControls(elemShape=HEX,
regions=instance.cells, technique=SWEEP)
part.seedEdgeByNumber(constraint=FINER,
edges=part.edges.findAt(((l, a, C/2),), ((-l, a, C/2),)
,((l, -a, C/2),), ((-l, -a, C/2), ),),number=50)
part.seedEdgeByNumber(constraint=FINER, edges=
part.edges.findAt(((l, a/2, 0.0),), ((-l, a/2, 0.0),),
((l, a/2, C),), ((-l, a/2, C),) ),number=10)
part.seedEdgeByNumber(constraint=FINER, edges=
part.edges.findAt(((l/2, a, 0.0),), ((l/2, -a, 0.0),),
((l/2, a, C),), ((l/2, -a, C),) ),number=10)
assembly.setElementType(regions=(instance.cells,),
elemTypes=(ElemType(elemCode=C3D8I, elemLibrary=
STANDARD),))
part.generateMesh()
```

Com todas as características iniciais da viga definidas falta criar as etapas do processo. Neste caso só se terá duas etapas, a etapa inicial na qual se define as condições de fronteira e a etapa 1 na qual é aplicada uma tensão equivalente à carga vertical.

```
#Step

model.StaticStep(name='Step', previous='Initial',
timePeriod=1.0, initialInc=0.1, description='Load
Aplication')
```

Com as etapas criadas passa-se para a identificação das condições fronteiras, que neste caso é o encastramento de uma das faces.

```
#Boundary Conditions

encastreregion=(instance.faces.findAt((0.0,0.0,0.0,)),)
model.EncastreBC(name='BC', createStepName='Step',
region=encastreregion)
```

A última coisa a definir é a aplicação da carga, que neste caso é uma tensão na face oposta ao encastramento, definindo a região de aplicação da mesma.

```
#Load

surfface=instance.faces.findAt(((0.0,0.0,C),))
assembly.Surface(name='Surf', side1Faces=surfface)
surftracregion=assembly.surfaces['Surf']
model.SurfaceTraction(name='Load',
createStepName='Step', directionVector=((0,0,0),(0,-
1,0)), region=surftracregion, magnitude=P,
distributionType=UNIFORM, resultant=ON)
```

Para finalizar define-se a tarefa de execução (processamento), responsável pela criação dos resultados que advém dos dados indicados no pré-processamento.

```
#job

job = mdb.Job(name='Bt',
model='Model',description='Test')
job.submit()
job.waitForCompletion()
```

Através deste *script*, no final, obtém-se os ficheiros de resultados do ABAQUS. Na Figura 3, demonstra-se um dos resultados obtidos do ficheiro odb, nomeadamente, a tensão equivalente de von Mises.

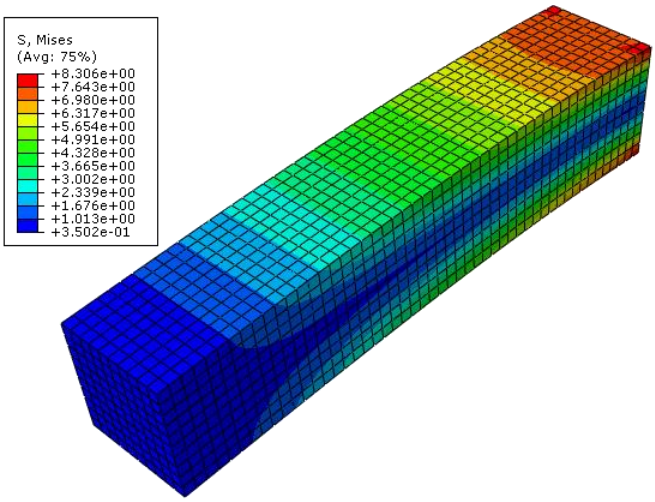

Figura 3 – Distribuição da tensão equivalente de von Mises na peça em estudo.

Este é um exemplo simples que ilustra a metodologia apresentada neste trabalho.

3. Otimização

3.1. Introdução

Ao longo dos anos, o Ser Humano tem tido necessidade de fazer otimização em diversos processos, mas só no início dos anos 50, devido à aparição do computador digital, existiu uma grande evolução na área da otimização. Desde então, com o desenvolvimento do computador, foi possível o desenvolvimento de novas técnicas de otimização e assim foi se tornando viável a resolução de problemas de otimização que até então eram praticamente impossíveis [1, 20].

Para a obtenção de um produto final da conformação plástica com as características desejadas deve-se tentar obter um resultado o mais próximo possível da geometria pretendida, evitando os diferentes defeitos inerentes a este processo. Para chegar a este resultado a tentativa-e-erro e determinadas abordagens empíricas são alguns dos métodos bastante utilizados, mesmo através de programas de simulação. Para evitar custos desnecessários, uma alternativa a estes métodos é a resolução de um problema inverso de forma a otimizar o processo de conformação plástica. Para tal é necessário definir inicialmente a geometria que se pretende em função de vários parâmetros [15].

Para além da obtenção de geometrias complexas, os processos de otimização permitiram a redução do peso e o aumento do desempenho. Por consequência diminuiu o custo dos produtos finais, permitindo assim uma maior competitividade nas indústrias [1].

Um dos fatores a que se dever ter muito cuidado quando se utiliza técnicas de otimização tradicionais é numa boa escolha de parâmetros iniciais do problema. Pois um dos problemas mais comuns destes casos é a obtenção de mínimos locais e não o pretendido mínimo global do problema. Outro fator a ter em conta é o facto da função objetivo do processo não ser suave nem contínua, e se assim for, o utilizador terá de utilizar algoritmos do tipo evolucionários ou genéticos em prole dos algoritmos baseados no gradiente [15, 21].

Na resolução de problemas inversos de identificação de parâmetros e otimização de forma podem ser utilizados um ou mais métodos de otimização. Estes métodos podem ser divididos em três famílias [15]:

- Métodos baseados no gradiente;
- Algoritmos evolucionários e inspirados na natureza;
- Redes neuronais e algoritmos de inteligência artificial.

Dentro destes métodos optou-se, para a resolução dos problemas apresentados no presente trabalho, por um método baseado no gradiente, sendo este um método do gradiente conjugado apresentado por Fletcher-Reeves.

3.2. Formulação de um problema de otimização

Para o processo de otimização de um dado problema, este tem que ser formulado corretamente. As características que definem o sistema a ser optimizador podem ser definidas matematicamente através de uma função objetivo. Através de x_1 a x_n parâmetros é definida a função objetivo F de acordo com [1, 22]:

$$F = f(x_1, x_2, \dots, x_n). \tag{3.1}$$

Na versão compacta em que $\boldsymbol{x}^T = [x_1\ x_2\ \dots\ x_n]$ e E^n é um espaço euclidiano de n dimensões [1, 22]:

$$F = f(\boldsymbol{x}) \text{ com } \boldsymbol{x} \in E^n. \tag{3.2}$$

Um simples problema de otimização pretende determinar todos os parâmetros (x_1 a x_n) contidos num determinado intervalo em que $x_{i,min} \leq x_i \leq x_{i,max}$, com i a variar entre 1 e n. Tendo como finalidade diminuir iterativamente o valor F, o problema define-se por [1, 22]:

$$\text{minimizar } F = f(\boldsymbol{x}). \tag{3.3}$$

Para obter um projeto ótimo existem varias restrições que fazem parte das condições necessárias de modo atingir este objetivo. Estas restrições podem ser divididas em restrições funcionais, as quais indicam limitações de comportamento do sistema, e restrições geométricas, que indicam as limitações físicas de transporte e fabrico [1, 22, 23]. Estas restrições estão representadas por h, para o caso de igualdades, e g, para o caso de desigualdades. A nível matemático pode escrever-se [1, 22]:

$$h(\boldsymbol{x}) = 0, \tag{3.4}$$

$$g(\boldsymbol{x}) \leq 0. \tag{3.5}$$

3.3. Método de Fletcher-Reeves

Neste trabalho foi aplicado o método do gradiente conjugado apresentado por Fletcher-Reeves.

Segue-se a apresentação do método das direções conjugadas, o qual é aplicado a funções quadráticas, e que atinge a solução após n iterações. A função quadrática usada por este é dada por [22, 24]:

$$f(\boldsymbol{x}) = \frac{1}{2} x^T \boldsymbol{Q} x - \boldsymbol{x}^T b, x \in \mathbb{R}^n \; e \; Q = Q^T > 0. \tag{3.6}$$

O método das diferenças conjugadas é baseado computacionalmente na definição de $\boldsymbol{Q}$, que é uma matriz $n \times n$ simétrica em que as direções $d^0, d^1, \ldots, d^m$ são $\boldsymbol{Q}$-conjugadas se para todos os $i \neq j$, $d^{iT} \boldsymbol{Q} d^j = 0$ [20, 22]. Assim, sabendo o ponto inicial x^0 e as direções $\boldsymbol{Q}$-conjugadas $d^0, d^1, \ldots, d^m$ para $k > 0$, a aplicação do algoritmos das direções conjugadas à equação 3.6 resulta em [20, 22]:

$$g^k = \nabla f(\boldsymbol{x}^k) = \boldsymbol{Q}\boldsymbol{x}^k - b, \tag{3.7}$$

$$\alpha_k = -\frac{\boldsymbol{g}^{kT} \boldsymbol{d}^k}{\boldsymbol{d}^{(kT)} \boldsymbol{Q} \boldsymbol{d}^k}, \tag{3.8}$$

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^{k} + \alpha_k \boldsymbol{d}^{k}. \tag{3.9}$$

Outro método das direções conjugadas é o método do gradiente conjugado, que tal como o método explicado anteriormente, quando é aplicado a uma função quadrática de n variáveis atinge o mínimo em n iterações [20, 22].

Por outro lado, no método do gradiente conjugado as direções são calculadas durante a execução do algoritmo. Ou seja, para cada iteração o método do gradiente conjugado calcula as direções usando as direções anteriores e o gradiente atual, fazendo com que todas as direções sejam $\boldsymbol{Q}$-conjugadas [22].

Assim, no algoritmo do gradiente conjugado, conhecendo o ponto inicial x^0, calcula-se o ponto posterior usando a maior direção decrescente, ou seja [22]:

$$\boldsymbol{x}^{1} = \boldsymbol{x}^{0} + \alpha_0 \boldsymbol{d}^{0}, \tag{3.10}$$

Em que

$$\boldsymbol{d}^{0} = -\boldsymbol{g}^{0}, \tag{3.11}$$

$$\alpha_0 = -\frac{\boldsymbol{g}^{0T}\boldsymbol{d}^{0}}{\boldsymbol{d}^{0T}\boldsymbol{Q}\boldsymbol{d}^{0}}. \tag{3.12}$$

O cálculo da direção d^{k+1} é feito pela combinação de g^{k+1} e d^k, sendo k o incremento. Matematicamente descrito por [22]:

$$\boldsymbol{d}^{k+1} = -\boldsymbol{g}^{k+1} + \beta_k \boldsymbol{d}^{k} \quad para \quad k = 1,2,3,\ldots \tag{3.13}$$

Para β_k igual a:

$$\beta_k = \frac{\boldsymbol{g}^{k+1T}\boldsymbol{Q}\boldsymbol{d}^{k}}{\boldsymbol{d}^{kT}\boldsymbol{Q}\boldsymbol{d}^{k}}. \tag{3.14}$$

O algoritmo do gradiente conjugado pode ser aplicado a funções não lineares se a equação 3.6 for aproximada por uma série de Taylor de segunda ordem. Pois, estas séries comportam-se aproximadamente como funções quadráticas perto da solução [22].

No entanto, nas funções não lineares, a aplicação deste método pode-se tornar pesada computacionalmente devido a reavaliação da matriz $\boldsymbol{Q}$ todas as iterações. Denota-se que esta matriz aparece unicamente no calculo de α_k e β_k. Então, para tornar este procedimento mais leve, substitui-se α_k por um procedimento de procura linear e utiliza-se um método que manipula algebricamente β_k de forma a tornar a equação 3.14 independente da matriz $\boldsymbol{Q}$ [22].

O método utilizado neste trabalho é o método de Fletcher-Reeves, o qual através da manipulação algébrica de β_k obtém a seguinte fórmula [20, 22]:

$$\beta_k = \frac{\boldsymbol{g}^{k+1T}\boldsymbol{g}^{k+1}}{\boldsymbol{g}^{kT}\boldsymbol{g}^{k}}. \tag{3.15}$$

3.4. Programa de otimização

Nos dias que correm, vários *softwares* podem ser utilizados para a implementação de metodologias de otimização. As características que distinguem cada um são a sua complexidade, a facilidade de utilização e o preço. Entre estes *softwares*, temos o caso do Microsoft Excel que permite responder a pequenos problemas, pois este possui características de otimização do âmbito geral. Outro é o MATLAB, que tem para a definição de modelos um ambiente interativo e uma diversidade de ferramentas e métodos de otimização [1].

No presente trabalho, utilizam-se *scripts* em Python que utilizam o programa ABAQUS para realizar as simulações numéricas, e obter as variáveis necessárias para o cálculo da função objetivo. Salienta-se que a função objetivo é necessária no problema de otimização. Com a finalidade de

simplificar, interliga-se todos os *scripts* em Python através de um pequeno interface efetuado na mesma linguagem.

O algoritmo de otimização deste problema implementa-se através da linguagem Fortran. Assim, para uma melhor acoplação dos *scripts* ao algoritmo de otimização utiliza-se uma interface implementada na mesma linguagem do algoritmo.

3.5. Aplicação do processo de otimização no exemplo da viga encastrada

A implementação da otimização no exemplo exposto na secção 2.4 é dividida em três secções:

- A obtenção da função objetivo utilizando *scripts*, onde efetua-se o pós-processamento da simulação, em que, utiliza-se os dados obtidos da simulação numérica para calcular-se a função objetivo.
- A interface da simulação utilizando *script*, no qual implementa-se a interligação dos restantes *scripts*, bem como a leitura das variáveis de entrada e escrita das variáveis de saída dos mesmos.
- A função objetivo utilizando uma subrotina Fortran, que consiste na criação do acoplamento da simulação numérica ao programa do algoritmo de otimização.

3.5.1. Exemplo da função objetivo utilizando *scripts*

A função objetivo utilizada neste exemplo consiste na minimização da área da viga. Porem, a função objetivo está sujeita à restrição que a tensão máxima de von Mises não ultrapasse o valor de 6 MPa. São escolhidas como variáveis de entrada do problema a altura e a largura, pois estas são utilizadas para o cálculo da área. O problema é descrito matematicamente por:

$$\text{minimizar } F = f(L, A), \tag{3.16}$$

$$\text{sujeito a: } \sigma_{max} \leq \sigma_{lim}, \text{ com } \sigma_{lim} = 6\ MPa \tag{3.17}$$

É através do programa ABAQUS que se obtém a função objetivo para o problema de otimização. Esta função objetivo é obtida através de um *script* redigido com linguagem Python. De seguida, explica-se um exemplo de como efetuar a obtenção da função objetivo.

Inicialmente tem-se que importar as funções que serão aplicadas no *script* a ser executado.

```
from odbAccess import *
```

Segue-se para o início do programa em si, começando pela definição da base de dados de saída.

```
#odb

odb = openOdb("Bt.odb")
```

Após este pormenor, lê-se o valor máximo da tensão de von Mises, tal como o elemento, *step* e *frame* em que este se encontra.

```
#MaxT

maxMises = -0.1
maxElem = 0
maxStep = "_None_"
maxFrame = -1
Stress = 'S'
isStressPresent = 0
for step in odb.steps.values():
    print 'Processing Step:', step.name
    for frame in step.frames:
        allFields = frame.fieldOutputs
        if (allFields.has_key(Stress)):
            isStressPresent = 1
            stressSet = allFields[Stress]
            for stressValue in stressSet.values:
                if (stressValue.mises > maxMises):
```

```
                        maxMises = stressValue.mises
                        maxElem = stressValue.elementLabel
                        maxStep = step.name
                        maxFrame = frame.incrementNumber

Tmax=str(maxMises)
```

Terminado isto, resta a definição da área da base da viga dada pela largura multiplicada pela altura da viga.

```
#Area

F=str(L*A)
```

No final, obtém-se a tensão máxima de von Mises e a área da base da viga, as quais são as variáveis de saída do exemplo.

3.5.2. Exemplo de interface da subrotina utilizando *script*

No presente trabalho, elabora-se um *script* geral que serve como interface da simulação. Este executa os *scripts* que foram anteriormente apresentados na secção 2.4 e a subsecção 3.5.1, seguindo uma determinada ordem. Além disto, lê os dados de entrada e escreve os dados de saída em ficheiros de texto, permitindo ao interface geral alterar e ler estes dados sem alterar nenhum *script*. Na Figura 4 pode-se observar a organização do Interface.

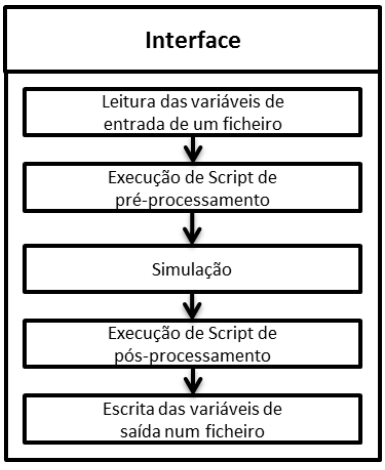

Figura 4 – Esquema do interface utilizado.

Inicialmente, importa-se todas as funções que serão aplicadas neste *script*. Tendo em conta que este *script* abre todos os *scripts* deste caso de estudo, importa-se nesta parte todas as funções necessárias para cada *script*.

```
from odbAccess import *
from abaqusConstants import *
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
from abaqus import *
from abaqusConstants import *
import visualization
import os
import datetime
import shutil
import time
import sys
import subprocess
```

Com isto, passa-se para a leitura dos dados que serão otimizados nesta simulação. É de denotar que no *script* da secção 2.4 estes dados têm de ser eliminados para não sobreporem aos que se inserem neste momento.

```
#Reading

par=open('Bi.txt','r')
L=par.readline()
A=par.readline()
L=abs(float(L))
A=abs(float(A))
par.close()
```

De seguida, executa-se o *script* que efetua o pré-processamento e posteriormente a simulação.

```
#Forming

execfile("B1.py")
mdb.saveAs(pathName='B' + '.cae')
job.submit()
job.waitForCompletion()
```

Com a simulação terminada, executa-se o *script* para a obtenção dos dados relativos a função objetivo.

```
#Post-processing

execfile("B2.py")
```

Por último, escreve-se os dados obtidos pelo *script* anterior num ficheiro de texto, os quais serão analisados posteriormente através do algoritmo de otimização.

```
#Writing

funcob=open('Bo.txt', 'w')
funcob.write(F)
funcob.write('\n')
funcob.write(Tmax)
funcob.close()
```

Como é apresentado, cada *script* relativo a simulação é executado de forma ordenada.

3.5.3. Exemplo da função objetivo utilizando uma subrotina

De forma a calcular a função objetivo, necessita-se da elaboração de uma subrotina que é responsável pela criação da função objetivo e da acoplação da simulação numérica e da otimização. Esta subrotina elabora-se na linguagem Fortran.

Inicialmente, define-se os dados que darão entrada na subrotina, tal como os dados que serão utilizados na mesma.

```
subroutine func(n, x, f,NFV,alfa)
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
DIMENSION X(1)
real*8 :: maxStress, alfa, LimStress,restrPenal
write(*,*)"in func",alfa
LimStress = 6.
```

Após a definição de cada variável a ser usada, escreve-se os valores de entrada para a simulação. Estes valores serão alterados todas as iterações de forma a minimizar a função objetivo.

```
open(unit=34,file="Bi.txt")
write(*,*)"x=",x(1),x(2)
do i=1,n
  write(34,*)x(i)
enddo
close(34)
```

Com as variáveis de entrada da simulação definidas, inicia-se o *script* implementado na subsecção 3.5.2.

```
call system('abaqus cae nogui=B.py')
```

Terminada a simulação, lê-se as variáveis necessárias para o cálculo da função objetivo.

```
open(unit=35,file='Bo.txt')
read(35,*)f
read(35,*) restrPenal
close(35)
```

Agora, calcula-se a função objetivo tendo em conta cada um dos dados lidos anteriormente, e escreve-se uma mensagem para o utilizador monitorizar estes dados no decorrer da otimização.

```
write(*,*)"Valor da funcao:",x(1),x(2)
maxStress = restrPenal
penal = alfa*max(0.0,(maxStress-LimStress))**2.
f = f + penal
write(*,*)"Valor da funcao e penal:",f,penal
write(*,*)"MaxStress=",maxStress," para o limite de",
LimStress
return
end
```

Terminando esta subrotina, insere-se esta no interface geral e é então otimizada a função objetivo.

3.5.4. Resultados da otimização do exemplo da viga encastrada

A interface geral é elaborada em linguagem Fortran, Nesta encontra-se a subrotina elaborada na subsecção anterior e o algoritmo de otimização explicado na secção 3.3. Através disto, obtém-se a otimização da função objetivo num número finito de iterações, o que pode ser observado na Tabela 1, na Figura 5 e na Figura 6.

Tabela 1 – Valores em cada iteração referentes ao exemplo da viga encastrada.

Largura da viga (mm)	Altura da viga (mm)	Função objetivo (mm^2)
10.0000	10.0000	365.8103
10.7114	11.3094	121.9209
10.6959	11.2994	121.9012
10.6909	11.2969	121.8997
10.6802	11.2938	121.8942
10.6659	11.2913	121.8892
10.6352	11.2915	121.8715
10.4659	11.5359	120.8198
5.2784	16.1702	88.1449
4.1354	18.1393	80.6414
4.2576	18.2038	78.2021
4.2485	18.2040	78.1928
4.2474	18.2041	78.1927
4.2460	18.2043	78.1924
4.2458	18.2043	78.1924

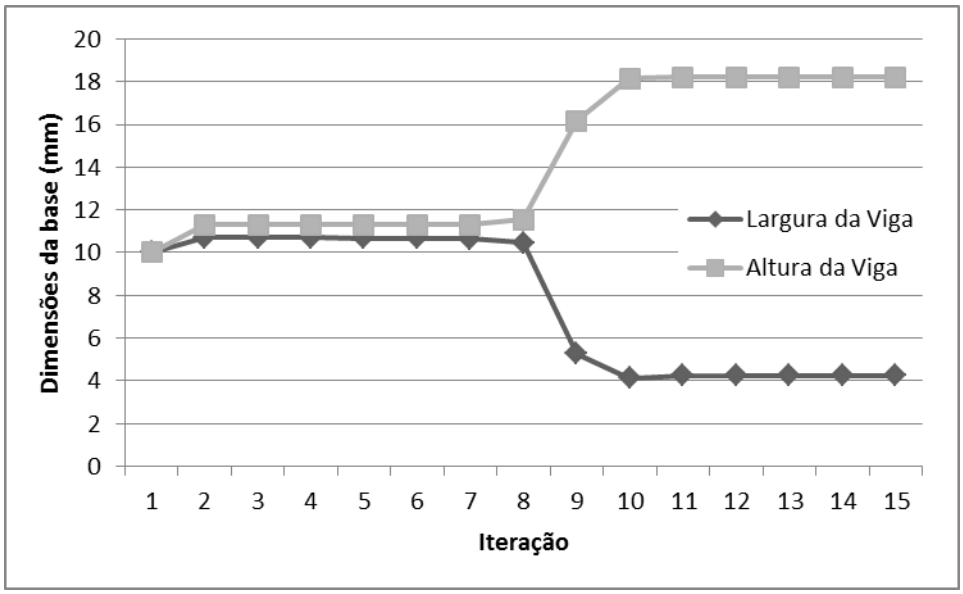

Figura 5 – Evolução das variáveis referentes ao exemplo da viga encastrada.

.

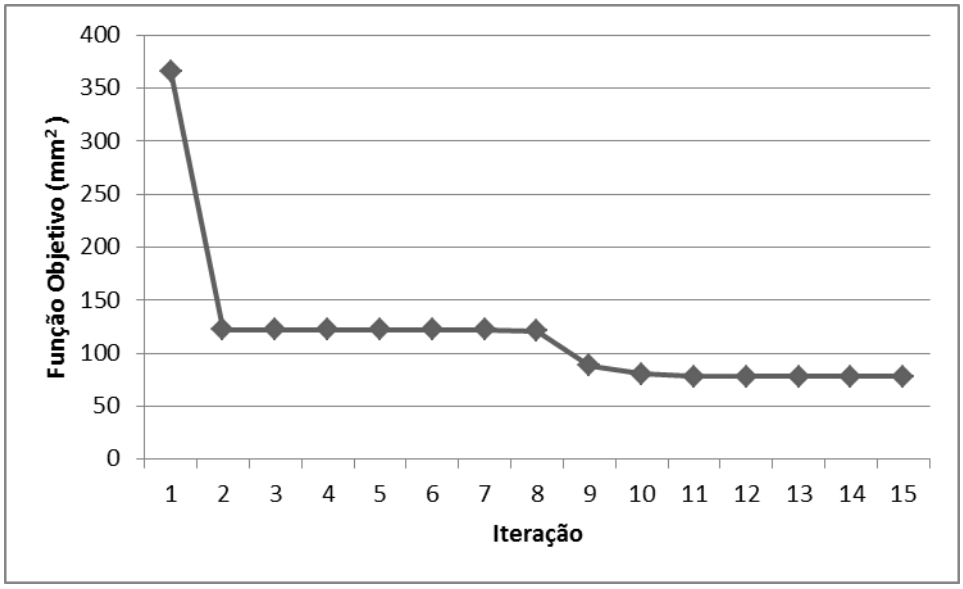

Figura 6 – Evolução da função objetivo referentes ao exemplo da viga encastrada.

Através da análise dos gráficos e da tabela pode-se constatar que ouve uma melhoria da função objetivo de aproximadamente 67% logo na segunda iteração, atingindo uma melhoria de 79% no final da otimização. Também, pode-se afirmar que segundo estes valores, a diminuição da largura e o aumento da altura influenciam positivamente a minimização função objetivo imposta.

Pode-se observar a geometria da viga depois do processo de otimização na Figura 7 e a resposta a restrição deste problema na Figura 8.

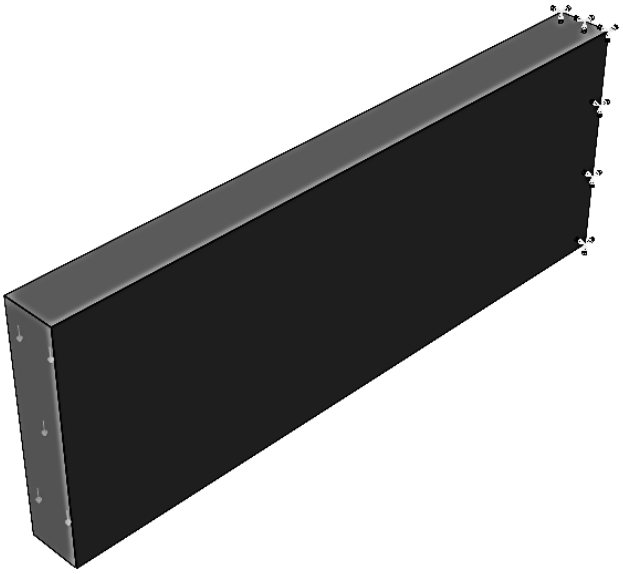

Figura 7 – Representação da geometria da viga depois do processo de otimização.

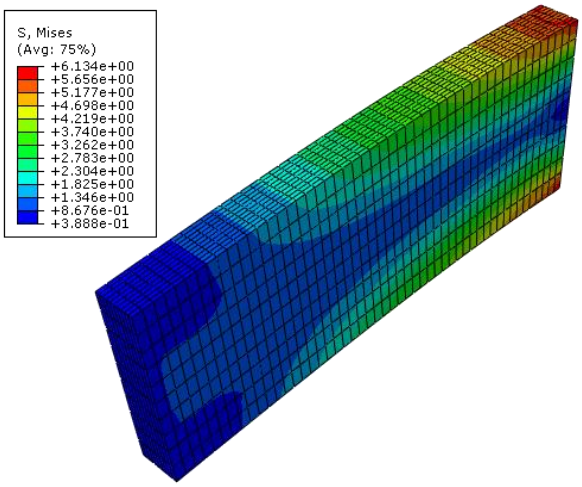

Figura 8 – Distribuição da tensão equivalente de von Mises na peça em estudo no final do processo de otimização.

III Aplicações Numéricas

4. Casos de conformação plástica

4.1. Introdução

Existem inúmeros casos em que se poderia aplicar a simulação numérica e a otimização acoplada. No entanto, neste trabalho optou-se pela escolha de casos de conformação plástica de chapas e a minimização do seu retorno elástico.

Entre os casos de estudo possíveis nesta área optou-se por dois casos que permitem abordar uma maior quantidade de detalhes, promovendo uma maior diversidade. O caso de estudo 1, mais simples e bastante estudado, é a conformação em U de uma chapa. O caso de estudo 2 é mais complexo devido a sua geometria ser mais irregular.

4.2. Caso de estudo 1

O caso de estudo 1 consiste na obtenção de uma chapa com uma geometria em U, como é observada na Figura 9. Para chegar a este resultado a chapa sofre conformação plástica, a qual é efetuada através da deformação do material com ajuda das ferramentas (o punção, a matriz e o cerra chapas). Pode-se observar uma representação do conjunto na Figura 10 no início e na Figura 11 no fim do processo.

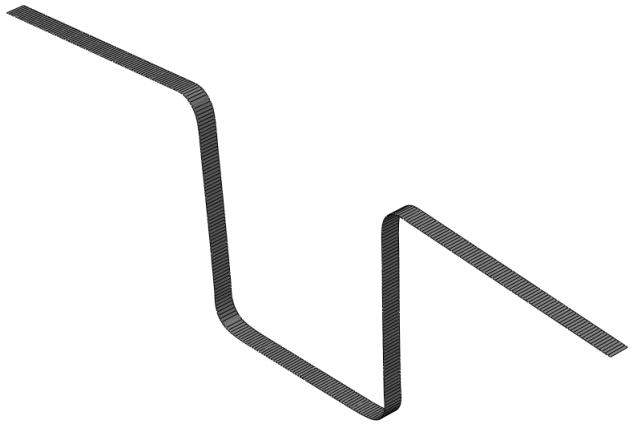

Figura 9 – Representação da chapa em três dimensões no final da conformação.

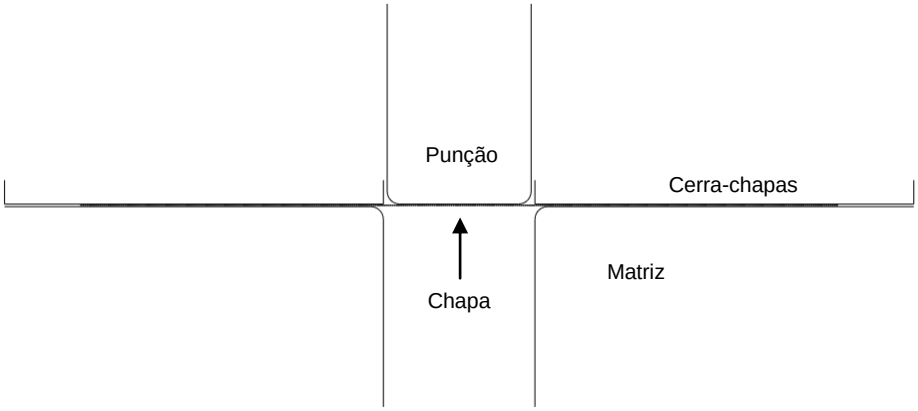

Figura 10 – Esquema do conjunto no início da conformação.

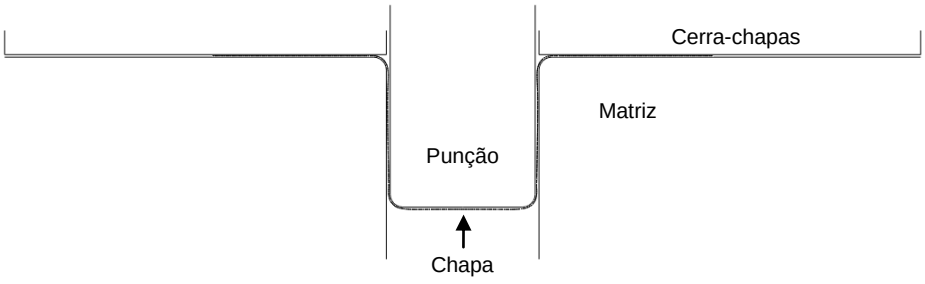

Figura 11 – Esquema do conjunto no final da conformação.

Para o estudo, utiliza-se uma chapa com 300 mm de comprimento, 5 mm de largura e espessura desprezável, que tem como material um aço macio

com o módulo de elasticidade de Young de 206.62 GPa, coeficiente de Poisson de 0.298, tensão de cedência inicial de 175 MPa e a curva tensão/deformação representada na Figura 12.

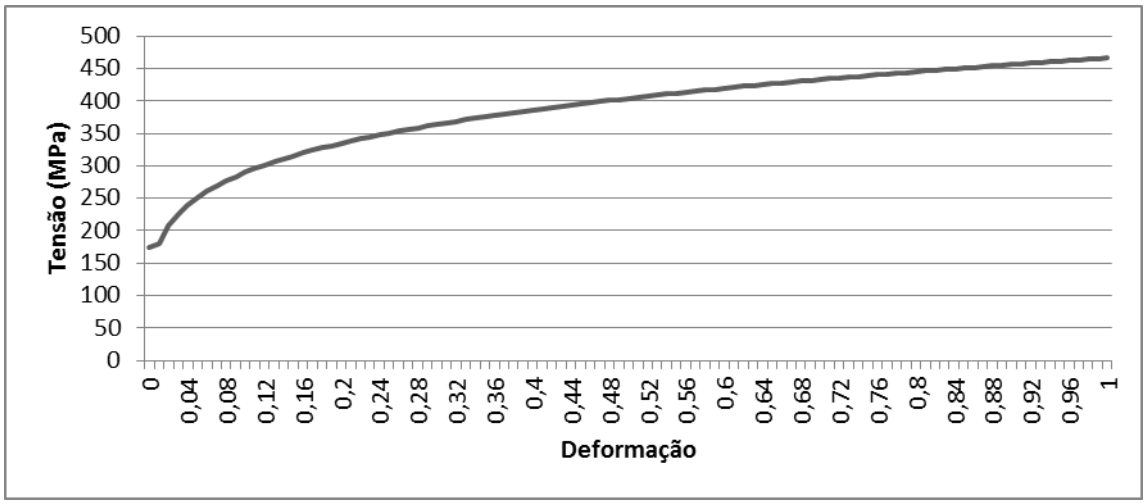

Figura 12 – Curva tensão/deformação do material.

A chapa sofre uma deformação devido a uma descida de 60 mm do punção, o qual tem uma largura de 57 mm e se movimenta a 1000 mm/s. Como resultado final obtém-se representado na Figura 11.

Posteriormente, as ferramentas são levantadas e observa-se o fenómeno de retorno elástico.

No presente caso de estudo tem-se como objetivo a minimização do retorno elástico. Para tal, necessita-se de otimizar uma função objetivo que defina o retorno elástico, e que é calculada a partir de três ângulos ($\theta_1, \theta_2 \; e \; \alpha$) que estão interligados, estes podem ser observados na Figura 13.

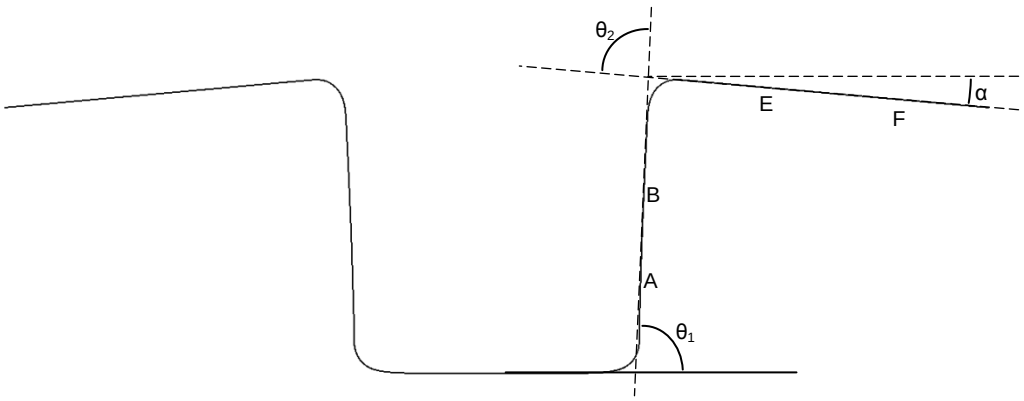

Figura 13 – Representação da chapa em duas dimensões no final da conformação.

Segundo estudos já feitos, o retorno elástico em aços de alta resistência está interligado com o raio do punção, o raio da matriz, e a força do cerra chapas. Sabe-se também que a diminuição do raio da matriz e o aumento do da força do cerra chapas permitem obter melhores resultados em relação ao problema em estudo [25]. Tendo isto em conta, escolhe-se estas como variáveis de entrada do caso de estudo. Matematicamente:

$$\text{minimizar } F = f(r_d, r_p, h_f). \tag{4.1}$$

Em que

$$f(r_d, r_p, h_f) = (90 - \theta_1)^2 + (90 - \theta_2)^2 + \alpha^2. \tag{4.2}$$

4.3. Caso de estudo 2

Para o caso de estudo 2, tem-se como base a geometria da chapa que pode ser observada na Figura 14, Esta é obtida, tal como no caso anterior, pela deformação do material através de ferramentas. O conjunto da chapa com as ferramentas pode ser observado na Figura 15 no início e na Figura 16 no fim do processo.

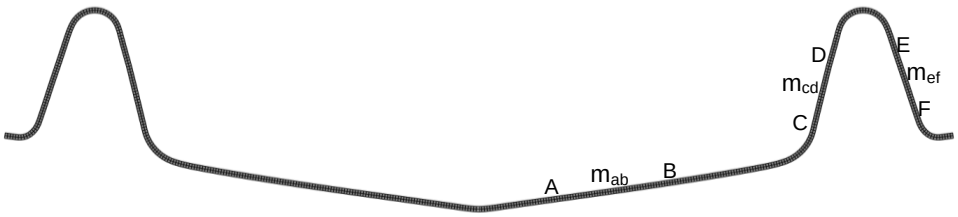

Figura 14 – Representação da chapa no final da conformação.

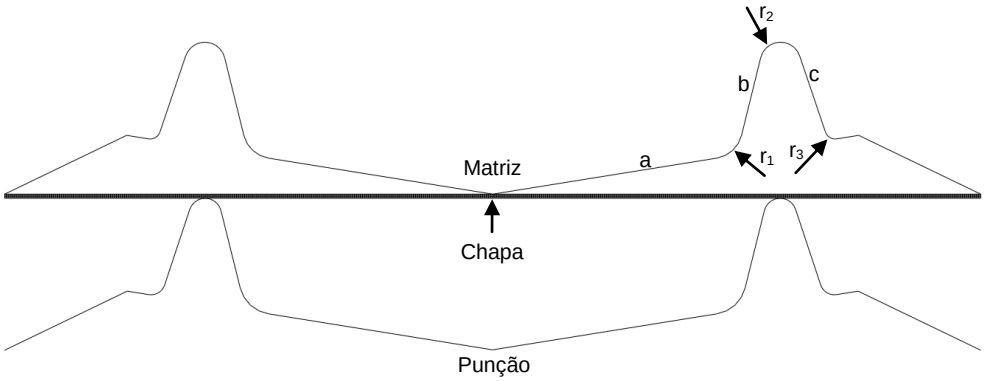

Figura 15 – Esquema do conjunto no início da conformação.

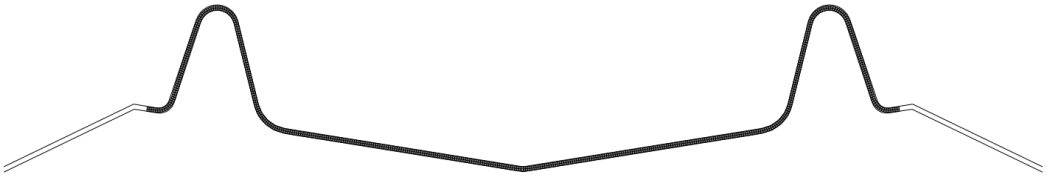

Figura 16 – Esquema do conjunto no final da conformação.

Neste caso, modela-se uma chapa em duas dimensões considerando que tem comprimento de 480 mm, 2 mm de espessura e de largura muito maior que a espessura. O material utilizado é o mesmo que foi utilizado no caso de estudo anterior. A chapa durante a conformação é deformada através da subida do punção com um deslocamento igual à altura da matriz e com uma velocidade de 1000 mm/s.

Para o cálculo da função objetivo, de forma a descrever o retorno elástico, utilizou-se como fatores as diferenças entre os declives que a peça devia ter se não sofresse retorno elástico (a, b, c), que neste caso são iguais

aos declives da matriz, e os observados em cada simulação (m_{ab}, m_{cd}, m_{ef}). Como variáveis de entrada do caso de estudo, escolheu-se três raios (r_1, r_2, r_3), os quais definem três fillets na geometria das ferramentas, estes podem ser observados na geometria da Figura 15. Matematicamente este problema é transcrito por:

$$\text{minimizar } F = f(r_1, r_2, r_3). \tag{4.3}$$

Em que

$$f(r_1, r_2, r_3) = |a - m_{ab}| + |b - m_{cd}| + |c - m_{ef}|. \tag{4.4}$$

5. Simulação numérica em conformação plástica

5.1. Introdução

A simulação numérica em conformação plástica pode ter diferentes metas. De forma a usar os casos de estudo escolhidos foi necessário estabelecer as seguintes:

- Criar uma metodologia para a simulação
- Utilizar *scripts* Python
- Caracterizar diferentes casos de estudo
- Verificar possíveis diferenças nos casos de estudo estudados.

A metodologia criada para a simulação divide-se em varias etapas, em que em cada uma destas cria-se o objeto, o qual é usado nas fases seguintes.

A criação de todos os objetos de uma forma generalizada denomina-se por primeira fase. Segue-se a enumeração das variáveis a ser estudadas e a partir destas, reformula-se o que foi feito na primeira fase de forma que o caso de estudo fique em função das variáveis, esta considera-se a segunda fase. No final da segunda fase, introduz-se no início do *script* a importação de todas as funções utilizadas. Pode-se observar na Figura 17 uma representação generalizada desta metodologia.

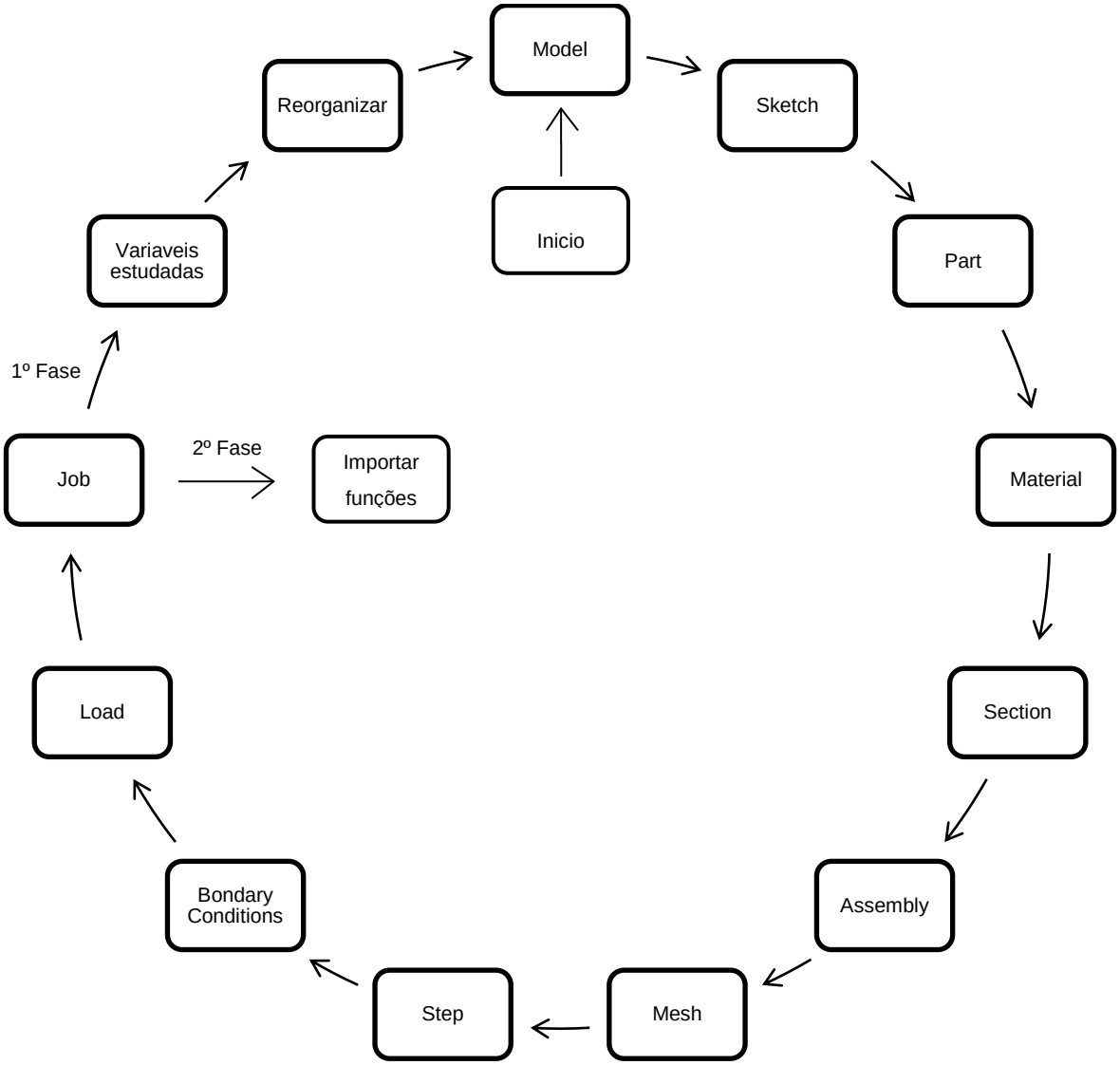

Figura 17 – Esquema da metodologia utilizada.

No caso de estudo 1, explica-se a criação de um *script* para a simulação numérica de uma chapa com perfil em U. Alguns pormenores específicos abordados neste são:

- A aproximação da chapa por uma casca,
- A criação de varias ferramentas,
- A definição de interações entre as ferramentas e a chapa.

No caso de estudo 2, explica-se um *script* para a simulação numérica de uma chapa com um perfil final mais complexo que o caso referido anteriormente. A criação de ferramentas, tal como a definição de interações também é abordada neste caso de estudo. No entanto, este é todo definido em duas dimensões e a chapa é considerada um sólido.

5.2. Caso de estudo 1

Tendo como base o exemplo dado na secção 2.4 e a metodologia apresentada na Figura 17, obtém-se um *script* para a elaboração deste problema.

No início do *script*, importa-se as funções que posteriormente serão utilizadas.

```
session.journalOptions.setValues(replayGeometry=COORDINA
TE, recoverGeometry=COORDINATE)

from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
from abaqusConstants import *
```

Agora, define-se as variáveis de entrada, em que três são lidas num ficheiro de texto.

```
#def var

pv=1000.0 #punch velocity
d=60.0 #distance
mct=d/pv
ct=mct*2 #punch step time
ht=0.01 #holder step time
```

```
par=open('Si.txt','r')
hf=par.readline() #holder force
rp=par.readline() #punch radius
rd=par.readline() #die radius
hf=abs(float(hf))
rp=abs(float(rp))
rd=abs(float(rd))
par.close()
```

De seguida, define-se o modelo.

```
#model

modelo = mdb.Model(name='Model-1')
```

Posteriormente, no *script*, define-se o esboço e de seguida a peça. Neste caso existe quatro peças e, por sua vez, quatro esboços que definem a chapa e 3 ferramentas: o punção, o cerra chapas e a matriz. Para a definição de cada uma destes, implementa-se primeiro o esboço e segue-se a implementação da peça de cada um separadamente.

Para a definição da peça tem-se que ter em conta as características de cada um destes objetos. Neste caso, a chapa é considerada uma casca tridimensional com corpo deformável e as ferramentas são superfícies rígidas analíticas também tridimensionais.

Começa-se pelas ferramentas, nas quais a primeira a definir-se é a matriz. O código do esboço desta é dado por:

```
# Create Die
#Sketch

dieprofile=modelo.ConstrainedSketch(name='dieprofile',
sheetSize=400.0)
dieprofile.Line(point1=(30.0, -80.0), point2=(30.0,
-0.5))
dieprofile.VerticalConstraint(entity=
dieprofile.geometry[2])
```

```
dieprofile.Line(point1=(30.0, -0.5), point2=(180.0,
-0.5))
dieprofile.HorizontalConstraint(entity=
dieprofile.geometry[3])
dieprofile.PerpendicularConstraint(entity1=
dieprofile.geometry[2], entity2=dieprofile.geometry[3])
dieprofile.FilletByRadius(curve1=dieprofile.geometry[3],
curve2=dieprofile.geometry[2], nearPoint1=((30.0+rd),
-0.5), nearPoint2=(30., (-0.5-rd)), radius=rd)
dieprofile.FixedConstraint(entity=
dieprofile.vertices[0])
dieprofile.FixedConstraint(entity=
dieprofile.vertices[2])
```

Com este código chega-se ao que no ABAQUS/CAE corresponderia a Figura 18.

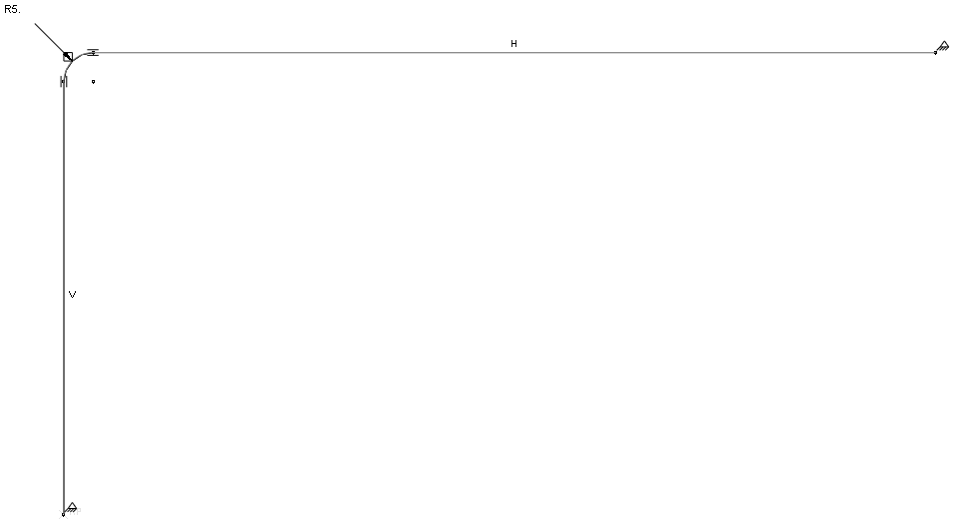

Figura 18 – Representação do esboço da matriz no ABAQUS/CAE.

Após a criação do esboço, cria-se a peça, obtendo-se o correspondente ao que ilustra a Figura 19.

```
#Part

diepart=modelo.Part(dimensionality=THREE_D, name='PART-
DIE', type=ANALYTIC_RIGID_SURFACE)
diepart.AnalyticRigidSurfExtrude(depth=20.0,
sketch=dieprofile)
```

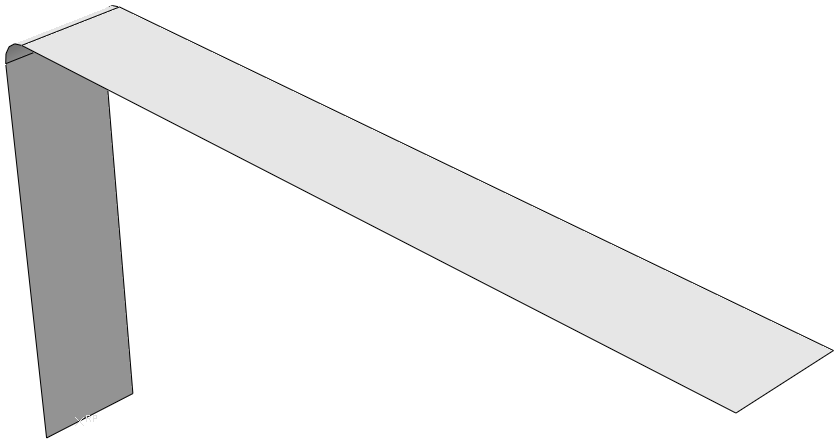

Figura 19 – Representação da peça da matriz no ABAQUS/CAE.

Por conseguinte, define-se a superfície e o ponto de referência que serão usados posteriormente.

```
# Create surfaces and sets

diepart.Surface(name='SURF-DIE',
side1Faces=diepart.faces.findAt(((133.333333, -0.5,
-3.333333), (15.0, -0.5, 0.0)), (((30.0+rd), -0.5,
3.333333), (30., (-0.5-rd), 0.0)), ((30.0, -40.25,
-3.333333), (30.0, -11.0, 0.0)), ))
diepart.ReferencePoint(point=(30.0,-80.0, 2.5))
diepart.Set(name='SET-DIE',
referencePoints=(diepart.referencePoints[3], ))
```

Com a matriz definida, segue-se para a definição do punção. Para tal, cria-se o esboço, a peça, a superfície e o ponto de referência utilizando a mesma ordem que foi aplicada na criação da matriz. Assim, Obtém-se o

correspondente ao que se pode visualizar na Figura 20 para o esboço e na Figura 21 para a peça.

```
# Create Punch
#Sketch

punchprofile=modelo.ConstrainedSketch(name=
'punchprofile', sheetSize=400.0)
punchprofile.Line(point1=(28.5, 80.0), point2=(28.5,
0.5))
punchprofile.VerticalConstraint(entity=
punchprofile.geometry[2])
punchprofile.Line(point1=(28.5, 0.5), point2=(0.0, 0.5))
punchprofile.HorizontalConstraint(entity=
punchprofile.geometry[3])
punchprofile.PerpendicularConstraint(entity1=
punchprofile.geometry[2],
entity2=punchprofile.geometry[3])
punchprofile.FixedConstraint(entity=
punchprofile.vertices[2])
punchprofile.FixedConstraint(entity=
punchprofile.vertices[0])
punchprofile.FilletByRadius(curve1=punchprofile.geometry
[2], curve2=punchprofile.geometry[3], nearPoint1=(28.5,
(rp+0.5)), nearPoint2=((28.5-rp), 0.5), radius=rp)
```

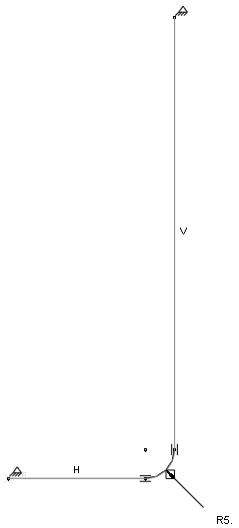

Figura 20 – Representação do esboço de metade simétrica do punção no ABAQUS/CAE.

```
#Part

punchpart=modelo.Part(dimensionality=THREE_D,
name='PART-PUNCH', type=ANALYTIC_RIGID_SURFACE)
punchpart.AnalyticRigidSurfExtrude(depth=20.0,
sketch=punchprofile)
```

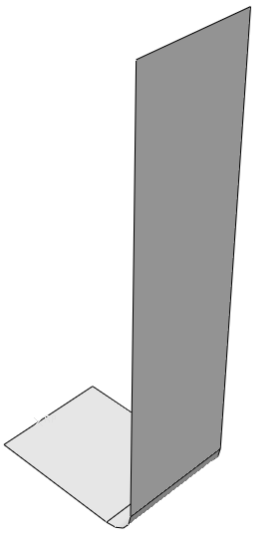

Figura 21 – Representação da parte de metade simétrica do punção no ABAQUS/CAE.

```
# Create surfaces and sets

punchpart.Surface(name='SURF-PUNCH',
side1Faces=punchpart.faces.findAt(((14.425, 0.5,
-3.333333), (0.0, 0.5, 0.0)), ((28.5, (rp+0.5),
3.333333), ((28.5-rp), 0.5, 0.0)), ((28.5, 40.25,
-3.333333), (28.5, 75.0, 0.0)), ))
punchpart.ReferencePoint(point=(0.0, 0.5, 2.5))
punchpart.Set(name='SET-PUNCH',
referencePoints=(punchpart.referencePoints[3], ))
```

A última ferramenta a definir-se é o cerra-chapas, implementando-se pela mesma ordem que as anteriores. Na Figura 22 e na Figura 23 podem-se observar o esboço e a peça deste, respetivamente.

```
# Create Holder
#Sketch

holderprofile=modelo.ConstrainedSketch(name=
'holderprofile', sheetSize=400.0)
holderprofile.Line(point1=(180.0, 10.0), point2=(180.0,
0.5))
holderprofile.VerticalConstraint(entity=
holderprofile.geometry[2])
holderprofile.Line(point1=(180.0, 0.5), point2=(30.0,
0.5))
holderprofile.HorizontalConstraint(entity=
holderprofile.geometry[3])
holderprofile.PerpendicularConstraint(entity1=
holderprofile.geometry[2], entity2=
holderprofile.geometry[3])
holderprofile.Line(point1=(30.0, 0.5), point2=(30.0,
10.0))
```

```
holderprofile.VerticalConstraint(entity=holderprofile.ge
ometry[4])
holderprofile.PerpendicularConstraint(entity1=holderprof
ile.geometry[3], entity2=holderprofile.geometry[4])
holderprofile.FixedConstraint(entity=holderprofile.verti
ces[3])
holderprofile.FixedConstraint(entity=holderprofile.verti
ces[0])
```

```
#Part

holderpart=modelo.Part(dimensionality=THREE_D,
name='PART-HOLDER', type=ANALYTIC_RIGID_SURFACE)
holderpart.AnalyticRigidSurfExtrude(depth=20.0,
sketch=holderprofile)
```

Figura 22 – Representação do esboço do cerra-chapas no ABAQUS/CAE.

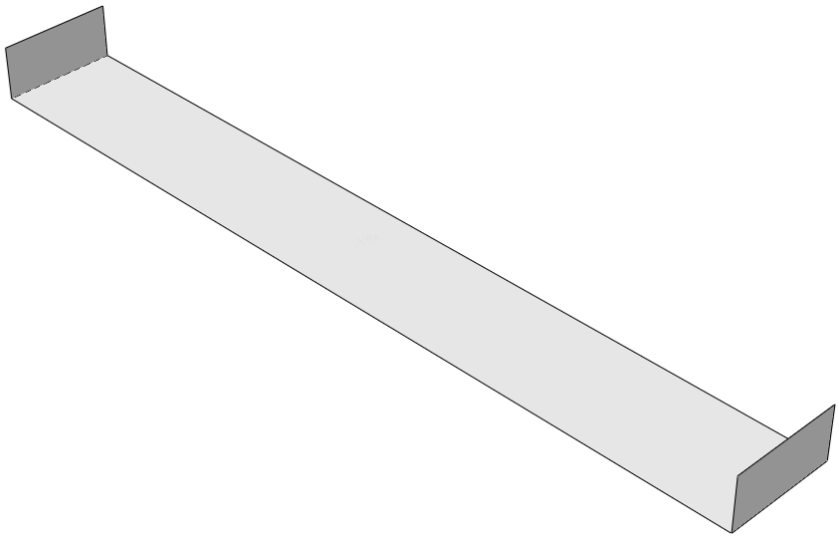

Figura 23 – Representação da parte do cerra-chapas no ABAQUS/CAE.

```
# Create surfaces and sets

holderpart.Surface(name='SURF-HOLDER',
side1Faces=holderpart.faces.findAt(((30.0, 5.25,
-3.333333), (30.0, 2.0, 0.0)), ((105.0, 0.5, -3.333333),
(45.0, 0.5, 0.0)), ((180.0, 5.25, -3.333333), (180.0,
7.0, 0.0)), ))
holderpart.ReferencePoint(point=(105.0, 5., 2.5))
holderpart.Set(name='SET-HOLDER', referencePoints=
(holderpart.referencePoints[3], ))
```

O cerra-chapas tem uma particularidade comparativamente com as outras ferramentas, que é relativa à implementação da inércia no ponto de referência do cerra-chapas.

```
# Create Inertia

holderpart.engineeringFeatures.PointMassInertia(alpha=
0.0, composite=0.0, mass=0.1, name='INERTIA-HOLDER',
region= holderpart.sets['SET-HOLDER'])
```

Após a definição das ferramentas, define-se a chapa. Para isso, utiliza-se a mesma ordem utilizada nas ferramentas para criar o esboço e a peça, obtendo-se o que se pode visualizar na Figura 24 para o esboço e na Figura 25 para a peça.

```
# Create Blank
#Sketch

blankprofile=modelo.ConstrainedSketch(name=
'blankprofile', sheetSize=200.0)
blankprofile.Line(point1=(0.0, 0.0), point2=(150.0,
0.0))
blankprofile.HorizontalConstraint(entity=blankprofile.ge
ometry[2])
```

```
#Part

blankpart=modelo.Part(dimensionality=THREE_D,
name='PART-BLANK', type=DEFORMABLE_BODY)
blankpart.BaseShellExtrude(depth=5.0,
sketch=blankprofile)
```

Figura 24 – Representação do esboço de metade simétrica da chapa no ABAQUS/CAE.

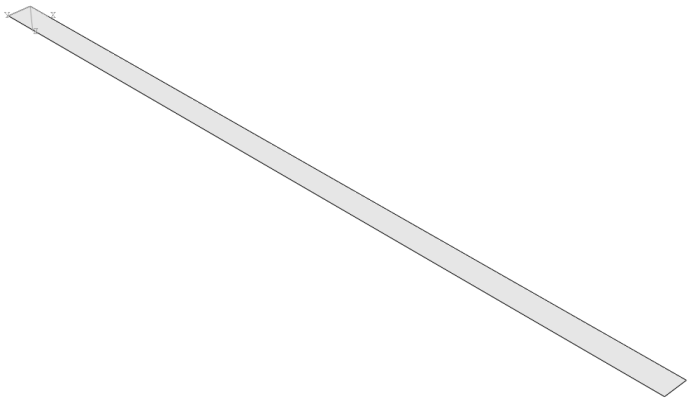

Figura 25 – Representação da parte de um quarto simétrico da chapa no ABAQUS/CAE.

Após a implementação da peça da chapa, cria-se duas superfícies; a superfície superior e a superfície inferior, e define-se duas arestas de referência; a aresta do centro da chapa e a aresta de simetria da chapa.

```
# Create Sets and Surfaces

blankpart.Surface(name='SURF-BLANKNEG',
side1Faces=blankpart.faces.findAt(((100.0, 0.0, 2.5),
(0.0, 0.0, 0.0)), ))
blankpart.Surface(name='SURF-BLANKPOS',
side2Faces=blankpart.faces.findAt(((100.0, 0.0, 2.5),
(0.0, 0.0, 0.0)), ))
blankpart.Set(edges=blankpart.edges.findAt(((0.0, 0.0,
2.5), ), ), name='SET-CENTER')
blankpart.Set(edges=blankpart.edges.findAt(((112.5, 0.0,
5.0), ), ((37.5, 0.0, 5.0), ), ), name='SET-SYMM')
```

Com todas as peças já definidas, segue-se para a criação do material, que tem como características as referidas na secção 4.2. Inicia-se a definição destas características pela densidade seguida pelas propriedades elásticas, módulo de elasticidade e coeficiente de Poisson, passando para as propriedades plásticas utilizando os pontos da curva tensão/deformação. Por último, define-se o nível de anisotropia deste material através da função potencial.

```
# Create Material

material=modelo.Material(name='material')
material.Density(table=((7.87e-09, ), ))
material.Elastic(table=((206.629E3, 0.298), ))
```

```
material.Plastic(table=((175.0,0.0), (180.47,0.01),
(208.17,0.02), (226.3,0.03), (240.12,0.04), (251.41,0.05),
(261.03,0.06), (269.45,0.07), (276.97,0.08), (283.77,0.09),
(289.99,0.1), (295.74,0.11), (301.09,0.12), (306.1,0.13),
(310.8,0.14), (315.25,0.15), (319.47,0.16), (323.49,0.17),
(327.32,0.18), (330.98,0.19), (334.5,0.2), (337.88,0.21),
(341.13,0.22), (344.27,0.23), (347.3,0.24), (350.23,0.25),
(353.07,0.26), (355.83,0.27), (358.5,0.28), (361.1,0.29),
(363.63,0.3), (366.1,0.31), (368.5,0.32), (370.84,0.33),
(373.13,0.34), (375.36,0.35), (377.55,0.36), (379.69,0.37),
(381.78,0.38), (383.83,0.39), (385.83,0.4), (387.8,0.41),
(389.73,0.42), (391.62,0.43), (393.48,0.44), (395.31,0.45),
(397.1,0.46), (398.86,0.47), (400.6,0.48), (402.3,0.49),
(403.98,0.5), (405.63,0.51), (407.26,0.52), (408.86,0.53),
(410.43,0.54), (411.99,0.55), (413.52,0.56), (415.03,0.57),
(416.52,0.58), (417.99,0.59), (419.44,0.6), (420.87,0.61),
(422.28,0.62), (423.68,0.63), (425.05,0.64), (426.41,0.65),
(427.75,0.66), (429.08,0.67), (430.39,0.68), (431.69,0.69),
(432.97,0.7), (434.24,0.71), (435.49,0.72), (436.73,0.73),
(437.95,0.74), (439.17,0.75), (440.37,0.76), (441.55,0.77),
(442.73,0.78), (443.89,0.79), (445.04,0.8), (446.18,0.81),
(447.31,0.82), (448.43,0.83), (449.54,0.84), (450.64,0.85),
(451.72,0.86), (452.8,0.87), (453.87,0.88), (454.93,0.89),
(455.97,0.9), (457.01,0.91), (458.04,0.92), (459.06,0.93),
(460.08,0.94), (461.08,0.95), (462.08,0.96), (463.06,0.97),
(464.04,0.98), (465.01,0.99), (465.98,1.0)))
material.plastic.Potential(table=((1.0000,1.03973,1.32188,1.12
798,1.0000,1.0000), ))
```

Com a definição do material terminada, cria-se a orientação do material na peça, implementando inicialmente o sistema cartesiano de coordenadas e de seguida a definição da orientação do material.

```
# Create orientation

blankpart.DatumCsysByThreePoints(coordSysType=CARTESIAN,
name='ROLLDIR', origin=(0.0, 0.0, 0.0), point1=(0.0,
0.0, 1.0), point2=(1.0, 0.0, 1.0))
blankpart.MaterialOrientation(additionalRotationField=''
, additionalRotationType= ROTATION_ANGLE, angle=0.0,
axis=AXIS_3, fieldName= '', localCsys=
blankpart.datums[6], orientationType= SYSTEM, region=
Region(faces= blankpart.faces.findAt(((100.0, 0.0,
3.333333), (0.0, -1.0, 0.0)), )))
```

Posteriormente, cria-se a secção, na qual define-se algumas peculiaridades, entre estas, anexa-se o material e define-se métodos de cálculo na secção. Por conseguinte, atribui-se esta à chapa.

```
# Create Section

modelo.HomogeneousShellSection(idealization=
NO_IDEALIZATION, integrationRule= SIMPSON, material=
'material', name='SECTION-SHELL', numIntPts= 5,
poissonDefinition= DEFAULT, preIntegrate= OFF,
temperature= GRADIENT, thickness= 1, thicknessField= '',
thicknessModulus=None, thicknessType= UNIFORM,
useDensity= OFF)
```

```
# Assign Section to Section

blankpart.SectionAssignment(offset= 0.0, offsetField=
'', offsetType= MIDDLE_SURFACE, region= Region( faces=
blankpart.faces.findAt(((100.0, 0.0, 3.333333), (0.0,
0.0, 0.0)), )), sectionName= 'SECTION-SHELL')
```

Após a secção segue-se a assemblagem, na qual insere-se todos as partes até agora definidas e implementa-se os eixos de coordenadas do conjunto das ferramentas com a chapa.

```
# Create Assembly


modelo.rootAssembly.DatumCsysByDefault(CARTESIAN)
blankinst=modelo.rootAssembly.Instance(dependent=ON,
name='PART-BLANK-1', part=blankpart)
dieinst=modelo.rootAssembly.Instance(dependent=ON,
name='PART-DIE-1', part=diepart)
holderinst=modelo.rootAssembly.Instance(dependent=ON,
name='PART-HOLDER-1', part=holderpart)
punchinst=modelo.rootAssembly.Instance(dependent=ON,
name='PART-PUNCH-1', part=punchpart)
```

Pode-se observar o resultado deste código na Figura 26.

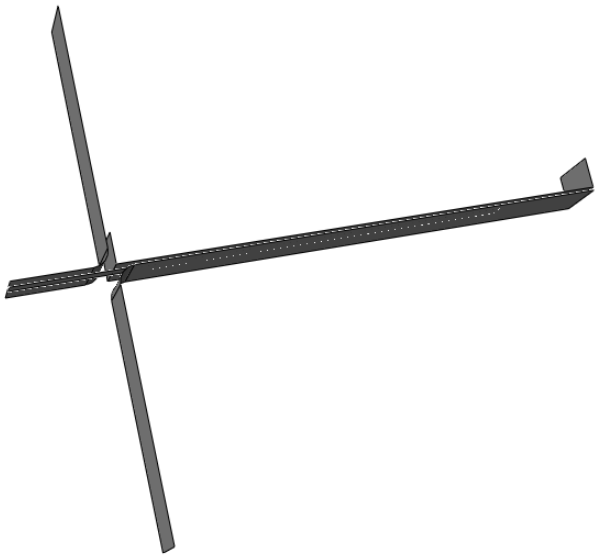

Figura 26 – Representação da assemblagem no ABAQUS/CAE.

Posteriormente, passa-se para a definição da malha. Nesta inicia-se pela denominação do formato de elemento, que neste caso é estruturado e quadrado. Passa-se para a implementação do número de elementos em cada uma das arestas e a definição do tipo de elemento, que neste caso é do tipo S4R[3]. Por último, cria-se a malha.

[3] O elemento S4R é um elemento casca de 4 nós e integração reduzida.

É de denotar também, que define-se os nós de referência necessários para a análise no pós-processamento.

```
# Create Mesh


blankpart.setMeshControls(elemShape= QUAD, regions=
blankpart.faces.findAt(((100.0, 0.0, 2.5), (0.0, 0.0,
0.0)), ), technique= STRUCTURED)
blankpart.seedEdgeByNumber(constraint= FINER, edges=
blankpart.edges.findAt(((75., 0.0, 5.0), ), ((75., 0.0,
0.0), ), ),number=180)
blankpart.seedEdgeByNumber(constraint= FINER, edges=
blankpart.edges.findAt(((0.0, 0.0, 2.5), ), ((150.0,
0.0, 2.5), ), ),number=1)
blankpart.setElementType(elemTypes= (ElemType(elemCode=
S4R, elemLibrary= STANDARD, hourglassControl= ENHANCED),
ElemType(elemCode= S4R, hourglassControl= ENHANCED,
elemLibrary= STANDARD)), regions=
(blankpart.faces.findAt(((100.0, 0.0, 2.5), )), ))
blankpart.generateMesh()
blankpart.Set(name= 'SET-REFNODES', nodes=
blankpart.nodes[0:181])
```

No ABAQUS/CAE, o condigo anterior representa o apresentado na Figura 27.

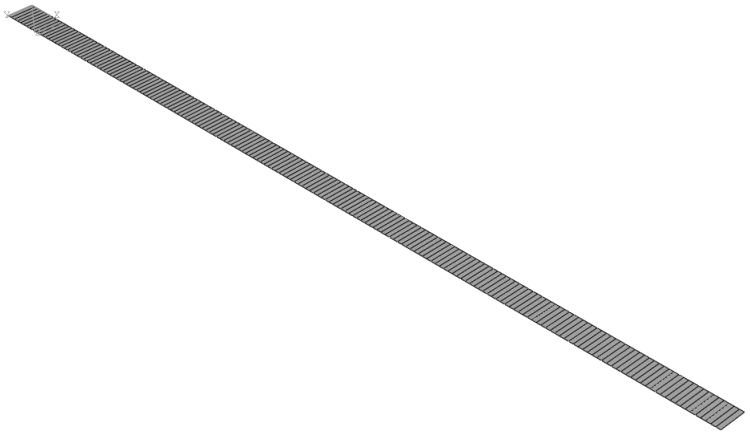

Figura 27 – Representação da malha da chapa no ABAQUS/CAE.

Com todas estas características do caso de estudo 1 definidas até este ponto, segue-se para a definição das etapas. Neste caso de estudo existem 6 etapas. A inicial, em que se define a maioria das características, a primeira etapa, na qua aplica-se a força do cerra-chapas, a segunda etapa, em que move-se o punção para baixo, a terceira etapa, que responsabiliza-se por mover para cima o punção, a quarta etapa, em que remove-se o punção e o cerra-chapas do contacto com a chapa, e a quinta etapa, na qual se move a matriz para baixo. Após isto, ainda determina-se as variáveis que ponderam-se obter da simulação.

```
# Create Steps


modelo.ImplicitDynamicsStep(description='Apply Blank
Holder Force', name='Step-1', previous='Initial',
timePeriod=ht, maxNumInc=10000, nlgeom=ON, minInc=1E-15,
initialInc=1E-15)
modelo.ImplicitDynamicsStep(description='Move Punch
Down', name='Step-2', previous='Step-1', timePeriod=
ct,maxNumInc=10000, nlgeom=ON, minInc=1E-15,
initialInc=1E-15)
modelo.steps['Step-2'].Restart(numberIntervals=1,
overlay=OFF, timeMarks=OFF)
modelo.ImplicitDynamicsStep(description='Move Punch Up',
name='Step-3', previous='Step-2', timePeriod=ct,
maxNumInc=10000, nlgeom=ON, minInc=1E-15, initialInc=
1E-15)
modelo.ImplicitDynamicsStep(description='Move Punch and
Holder Up', name='Step-4', previous='Step-3',
timePeriod=1, maxNumInc=10000, nlgeom=ON, initialInc=1E-
015, minInc=1E-015)
```

```
modelo.ImplicitDynamicsStep(description='Move Die Down',
name='Step-5', previous='Step-4', timePeriod=1,
maxNumInc= 10000, nlgeom=ON, initialInc=1E-015, minInc=
1E-015)
modelo.fieldOutputRequests['F-Output-
1'].setValues(numIntervals=10, variables=('S', 'TRIAX',
'PE', 'PEEQ', 'LE', 'U', 'RF', 'CSTRESS', 'STH',
'COORD'))
```

Passa-se então para a implementação das propriedades das interações entre as ferramentas e a chapa. Nestas propriedades define-se principalmente o atrito entre chapa e cada uma das ferramentas.

```
# Create Interaction Properties

modelo.ContactProperty('INTPROP-DIE')
modelo.interactionProperties['INTPROP-
DIE'].TangentialBehavior(dependencies= 0,
directionality= ISOTROPIC, elasticSlipStiffness= None,
formulation= PENALTY, fraction= 0.005,
maximumElasticSlip= FRACTION, pressureDependency= OFF,
shearStressLimit= None, slipRateDependency= OFF,
table=((0.15, ), ), temperatureDependency= OFF)
modelo.ContactProperty('INTPROP-PUNCH')
modelo.interactionProperties['INTPROP-
PUNCH'].TangentialBehavior(dependencies=0,
directionality=ISOTROPIC, elasticSlipStiffness=None,
formulation=PENALTY, fraction=0.005,
maximumElasticSlip=FRACTION, pressureDependency=OFF,
shearStressLimit=None, slipRateDependency=OFF,
table=((0.15, ), ), temperatureDependency=OFF)
modelo.ContactProperty('INTPROP-HOLDER')
```

```
modelo.interactionProperties['INTPROP-
HOLDER'].TangentialBehavior(dependencies=0,
directionality=ISOTROPIC, elasticSlipStiffness=None,
formulation=PENALTY, fraction=0.005,
maximumElasticSlip=FRACTION, pressureDependency=OFF,
shearStressLimit=None, slipRateDependency=OFF,
table=((0.15, ), ), temperatureDependency=OFF)
```

Posteriormente às propriedades das interações definidas, cria-se as interações. Nestas corresponde-se as propriedades das interações às superfícies em que são aplicadas.

```
# Create Interactions


modelo.SurfaceToSurfaceContactStd(clearanceRegion= None,
createStepName= 'Initial', datumAxis= None,
initialClearance= OMIT, interactionProperty= 'INTPROP-
DIE', master=dieinst.surfaces['SURF-DIE'], name='INT-
DIE', slave=blankinst.surfaces['SURF-BLANKNEG'],
sliding=FINITE)
modelo.SurfaceToSurfaceContactStd(clearanceRegion= None,
createStepName='Initial', datumAxis=None,
initialClearance=OMIT, interactionProperty='INTPROP-
PUNCH', master=punchinst.surfaces['SURF-PUNCH'],
name='INT-PUNCH', slave=blankinst.surfaces['SURF-
BLANKPOS'], sliding=FINITE)
modelo.SurfaceToSurfaceContactStd(clearanceRegion=None,
createStepName='Initial', datumAxis=None,
initialClearance=OMIT, interactionProperty='INTPROP-
HOLDER', master=holderinst.surfaces['SURF-HOLDER'],
name='INT-HOLDER', slave=blankinst.surfaces['SURF-
BLANKPOS'], sliding=FINITE)
```

De seguida, realiza-se a implementação das condições de fronteira. Inicialmente, implementa-se o código das condições de fronteira relativas aos deslocamentos e posteriormente as relativas a velocidade.

Nas condições fronteira relativa aos deslocamentos define-se principalmente as que se referem as condições iniciais do caso de estudo, libertando e fixando a respetiva condição em etapas nas quais isto é necessário. Além disto, na quarta e quinta etapa define-se condições finais de modo a permitir que a chapa não esteja em contacto com nenhuma ferramenta.

```
# Create Boundary Conditions (displacement)

modelo.DisplacementBC(amplitude=UNSET,
createStepName='Initial', distributionType=UNIFORM,
fieldName='', localCsys=None, name='BC-DIE',
region=dieinst.sets['SET-DIE'], u1=SET, u2=SET, u3=SET,
ur1=SET, ur2=SET, ur3=SET)
modelo.boundaryConditions['BC-
DIE'].setValuesInStep(stepName='Step-5', u2=-120)
modelo.DisplacementBC(amplitude=UNSET,
createStepName='Initial', distributionType=UNIFORM,
fieldName='', localCsys=None, name='BC-HOLDER',
region=holderinst.sets['SET-HOLDER'], u1=SET, u2=UNSET,
u3=SET, ur1=SET, ur2=SET, ur3=SET)
modelo.DisplacementBC(amplitude=UNSET,
createStepName='Initial', distributionType=UNIFORM,
fieldName='', localCsys=None, name='BC-PUNCH',
region=punchinst.sets['SET-PUNCH'], u1=SET, u2=SET,
u3=SET, ur1=SET, ur2=SET, ur3=SET)
modelo.boundaryConditions['BC-
PUNCH'].setValuesInStep(stepName='Step-2', u2=FREED)
modelo.DisplacementBC(amplitude=UNSET,
createStepName='Initial', distributionType=UNIFORM,
fieldName='', localCsys=None, name='BC-CENTER',
region=blankinst.sets['SET-CENTER'], u1=SET, u2=UNSET,
u3=UNSET, ur1=UNSET, ur2=SET, ur3=SET)
```

```
modelo.boundaryConditions['BC-
CENTER'].setValuesInStep(stepName='Step-3', u1=SET, u2=-
60.05, u3=SET, ur1=SET, ur2=SET, ur3=SET)
modelo.boundaryConditions['BC-
HOLDER'].setValuesInStep(stepName='Step-4', u2=60.0)
modelo.boundaryConditions['BC-
PUNCH'].setValuesInStep(stepName='Step-4', u2=60.0)
```

De seguida, implementa-se as condições fronteiras relativas as velocidades. Para tal, cria-se uma tabela que crie uma incrementação do valor de velocidade ao longo do tempo. No final, esta velocidade converte-se num determinado deslocamento. Depois implementa-se as velocidades para baixar e levantar o punção em cada uma das etapas respetivas.

```
# Create Boundary Conditions (velocity)

modelo.TabularAmplitude(data=((0.0, 0.0), (mct, 1.0),
(ct, 0.0)), name='AMP-PUNCH', smooth=SOLVER_DEFAULT,
timeSpan=STEP)
modelo.VelocityBC(amplitude='AMP-PUNCH',
createStepName='Step-2', distributionType=UNIFORM,
fieldName='', localCsys=None, name='BCp-2000',
region=punchinst.sets['SET-PUNCH'],v1=UNSET, v2=-pv,
v3=UNSET, vr1=UNSET, vr2=UNSET, vr3=UNSET)
modelo.VelocityBC(amplitude='AMP-PUNCH',
createStepName='Step-3', distributionType=UNIFORM,
fieldName='', localCsys=None, name='BCp+2000',
region=punchinst.sets['SET-PUNCH'],v1=UNSET, v2=2*pv,
v3=UNSET, vr1=UNSET, vr2=UNSET, vr3=UNSET)
```

Posteriormente, cria-se as cargas. Nestas, criam-se tabelas de forma a proporcionar um incremento de carga estável. Para isso, aplica-se a carga nas etapas em que esta é necessária.

```
# Create Loads

modelo.TabularAmplitude(data=((0.0, 0.0), (ht, 1.0)),
name='AMP-HOLDING', smooth=SOLVER_DEFAULT,
timeSpan=STEP)
modelo.TabularAmplitude(data=((0.0, 1.0), (ct, 1.32)),
name='AMP-STAMPING', smooth=SOLVER_DEFAULT,
timeSpan=STEP)
modelo.ConcentratedForce(amplitude='AMP-HOLDING', cf2=-
hf, createStepName='Step-1', distributionType=UNIFORM,
field='', localCsys=None, name='LOAD-HOLDER',
region=holderinst.sets['SET-HOLDER'])
modelo.loads['LOAD-
HOLDER'].setValuesInStep(amplitude='AMP-STAMPING',
stepName='Step-2')
```

Para finalizar o ensaio através de *script*, implementa-se a tarefa que realiza a simulação numérica.

```
# Create Job

job=mdb.Job(atTime=None, contactPrint=OFF,
description='', echoPrint=OFF, historyPrint=OFF,
model='Model-1', modelPrint=OFF,
multiprocessingMode=DEFAULT, name='S',
nodalOutputPrecision=SINGLE, numCpus=1, numDomains=1,
queue=None, scratch='', type=ANALYSIS,
userSubroutine='', waitHours=0, waitMinutes=0)
job.submit()
job.waitForCompletion()
```

Com todo o código que foi implementado e apresentado nesta secção, executa-se o pré-processamento e processamento, obtendo-se após isto obtém-se o ficheiro de resultados. Pode-se observar na Figura 28 o resultado da simulação, isto é, a geometria final da chapa e a posição das ferramentas.

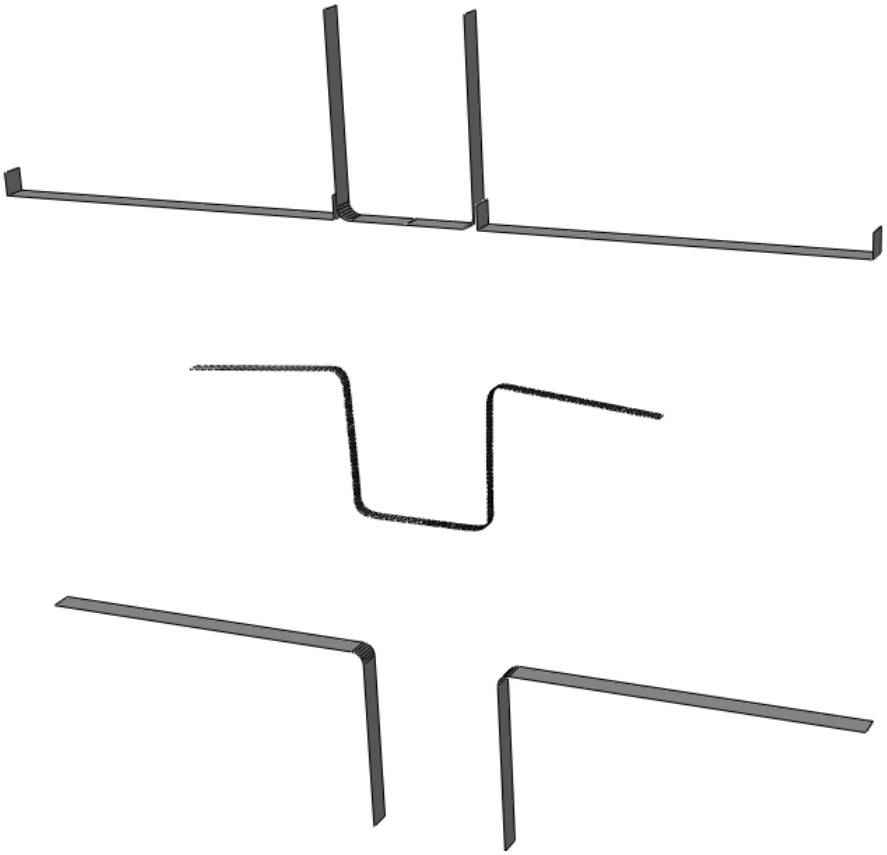

Figura 28 – Representação da geometria final da chapa e da posição das ferramentas.

5.3. Caso de estudo 2

Para este caso de estudo a simulação executa-se em duas dimensões, pois a terceira dimensão não iria acrescentar nenhum fator importante a este processo.

Baseado na metodologia já apresentada na secção anterior, obtém-se o *script* para a execução da simulação deste caso de estudo, explicando em seguida.

No início do *script* importa-se todas as funções que serão necessárias para a simulação deste caso:

```
session.journalOptions.setValues(replayGeometry=COORDINA
TE, recoverGeometry=COORDINATE)

from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
from abaqusConstants import *
import math
```

De seguida segue-se a definição das variáveis referentes ao caso de estudo, iniciando-se pelas variáveis relativamente à geometria da chapa, e posteriormente, às ferramentas. A única variável que se define quanto à chapa é a espessura.

```
#var da geometria
#espessura blank

t=2.0
```

Depois da chapa, cria-se a matriz. Para tal, necessita-se da definição das retas que constituem a matriz, definindo os seus declives e respetivas constantes.

```
## die
#retas

a=0.16
a1=0.0
b=4.0
b1=100.0-b*140.0
c=-3
c1=100.0-c*140.0
```

Através das retas anteriormente definidas, calculam-se os pontos de uma geometria mais rudimentar, sendo esta só definida por retas.

```
#pontos

x0=0.0
y0=0.0
x1=(b1-a1)/(a-b)
y1=x1*a+a1
x2=(c1-b1)/(b-c)
y2=x2*b+b1
x3=(a1-c1)/(c-a)
y3=x3*a+a1
x4=180.0
y4=x4*a+a1
x5=240.0
y5=0.0
```

Com os últimos dois pontos calcula-se o declive da última reta, que será necessário posteriormente para a definição da geometria do punção.

```
#declive da ultima reta

d=(y5-y4)/(x5-x4)
```

As últimas variáveis necessárias para a definição de toda a geometria são os três raios. Estes são considerados como fatores de entrada, e por isso,

são retirados de um ficheiro exterior, no qual serão inseridos antes de correr a simulação.

```
par=open('ci.txt','r')
r1=par.readline()
r2=par.readline()
r3=par.readline()
r1=abs(float(r1))
r2=abs(float(r2))
r3=abs(float(r3))
par.close()
```

Através do que foi definido anteriormente, calculam-se os pontos dos centros dos raios. Para chegar esta formulação foi necessário a manipulação da fórmula relativa à distância entre duas retas paralelas, em que a distância equivale ao valor do raio do respetivo círculo.

```
#centro dos raios

xr1=((sqrt(b**2+1)-sqrt(a**2+1))*(-r1)-b1+a1)/(b-a)
yr1=-(sqrt(a**2+1)*(-r1)-a*xr1-a1)
xr2=((sqrt(c**2+1)-sqrt(b**2+1))*(r2)-c1+b1)/(c-b)
yr2=-(sqrt(b**2+1)*(r2)-b*xr2-b1)
xr3=((sqrt(a**2+1)-sqrt(c**2+1))*(-r3)-a1+c1)/(a-c)
yr3=-(sqrt(a**2+1)*(-r3)-a*xr3-a1)
```

Calcula-se então o ponto de altura máxima da peça e acrescenta-se a espessura da chapa. Este valor será utilizado posteriormente para definir as coordenadas em que o punção se deve situar inicialmente sem estar em contacto com a chapa.

```
#altura total + espessura

h=r2+yr2+t
```

Passa-se então para o cálculo das coordenadas em que os círculos e as retas da geometria rudimentar são tangentes. Para tal é necessário a definição de retas perpendiculares através do cálculo dos declives.

```
#calculo dos pontos extremos dos arcos
#perpendiculares as retas passando pelos centros

at=-1/a
a2=yr1-at*xr1
a3=yr3-at*xr3
bt=-1/b
b2=yr1-bt*xr1
b3=yr2-bt*xr2
ct=-1/c
c2=yr2-ct*xr2
c3=yr3-ct*xr3
```

Através do cruzamento das retas calculadas anteriormente e as retas da geometria rudimentar, obtém-se os pontos de tangência entre os raios e as retas da geometria rudimentar.

```
#pontos extremos dos arcos

u1=(a2-a1)/(a-at)
v1=a*u1+a1
u2=(b2-b1)/(b-bt)
v2=b*u2+b1
u3=(b3-b1)/(b-bt)
v3=b*u3+b1
u4=(c2-c1)/(c-ct)
v4=c*u4+c1
u5=(c3-c1)/(c-ct)
v5=c*u5+c1
u6=(a3-a1)/(a-at)
v6=a*u6+a1
```

Com isto, tem-se todos os pontos necessários para a definição da geometria da matriz. Segue-se a definição dos pontos para a geometria do punção. Para isso define-se inicialmente as retas necessárias para obter a

geometria rudimentar do punção. As retas do punção são paralelas às da matriz compartilhando o mesmo declive. Deste modo, é unicamente necessário o cálculo das constantes destas retas através da fórmula da distância entre duas retas paralelas.

```
## punch
#retas

ap=a
ap1=-t*sqrt(a**2+1)+1*y0-a*x0
bp=b
bp1=-t*sqrt(b**2+1)+1*y2-b*x2
cp=c
cp1=-t*sqrt(c**2+1)+1*y2-c*x2
dp=d
dp1=-t*sqrt(d**2+1)+1*y5-d*x5
```

Com isto segue-se para a obtenção dos pontos da geometria. Obtém-se o primeiro ponto e o ponto final através das equações das retas e a coordenada em X de 0.0 e 240.0 respetivamente. O resto dos pontos são calculados através da interceção de retas.

```
#pontos

xp0=0.0
yp0=xp0*ap+ap1
yp0i=yp0-h
xp1=(bp1-ap1)/(ap-bp)
yp1=xp1*ap+ap1
yp1i=yp1-h
xp2=(cp1-bp1)/(bp-cp)
yp2=xp2*bp+bp1
yp2i=yp2-h
xp3=(ap1-cp1)/(cp-ap)
yp3=xp3*ap+ap1
```

```
yp3i=yp3-h
xp4=(dp1-ap1)/(ap-dp)
yp4=xp4*ap+ap1
yp4i=yp4-h
xp5=240.0
yp5=xp5*dp+dp1
yp5i=yp5-h
```

De seguida definem-se os raios do punção, os quais dependem dos raios da matriz e da espessura da chapa.

```
#raios

rp1=r1+t
rp2=r2-t
rp3=r3+t
```

Depois de obtidos os raios, utiliza-se uma metodologia para o cálculo do resto das variáveis da geometria do punção semelhante à utilizada no cálculo da geometria da matriz, com a diferença é que neste caso será inserido para cada coordenada relativa ao eixo Y um ponto adicional, no qual será subtraído o valor da altura total da peça mais a espessura. Assim, o punção não está em contacto com a chapa e a matriz, mantendo uma distância inicial igual a largura da chapa no ponto em que o punção se encontra mais próximo desta.

Calcula-se então o centro dos círculos e o respetivo ponto inicial.

```
#centro dos raios

xpr1=((sqrt(bp**2+1)-sqrt(ap**2+1))*(-rp1)-bp1+ap1) /
(bp-ap)
ypr1=-(sqrt(ap**2+1)*(-rp1)-ap*xpr1-a1)
ypr1i=ypr1-h
xpr2=((sqrt(cp**2+1)-sqrt(bp**2+1))*(rp2)-cp1+bp1) /
(cp-bp)
ypr2=-(sqrt(bp**2+1)*(rp2)-bp*xpr2-bp1)
```

```
ypr2i=ypr2-h
xpr3=((sqrt(ap**2+1)-sqrt(cp**2+1))*(-rp3)-ap1+cp1) /
(ap-cp)
ypr3=-(sqrt(ap**2+1)*(-rp3)-ap*xpr3-ap1)
ypr3i=ypr3-h
```

Segue-se para o cálculo dos extremos dos arcos do punção, tal como os pontos em que os círculos e as retas são tangentes. Para tal, começa-se pelo cálculo das retas perpendiculares a cada uma das retas do punção e que intercetam os centros dos raios.

```
#calculo dos pontos extremos dos arcos
#perpendiculares as retas passando pelos centros

apt=-1/ap
ap2=ypr1-apt*xpr1
ap3=ypr3-apt*xpr3
bpt=-1/bp
bp2=ypr1-bpt*xpr1
bp3=ypr2-bpt*xpr2
cpt=-1/cp
cp2=ypr2-cpt*xpr2
cp3=ypr3-cpt*xpr3
```

Com a intercessão das retas anteriormente calculadas e as retas iniciais do punção, obtém-se então os pontos dos extremos dos arcos.

```
#pontos exremos dos arcos

up1=(ap2-ap1)/(ap-apt)
vp1=ap*up1+ap1
vp1i=vp1-h
up2=(bp2-bp1)/(bp-bpt)
vp2=bp*up2+bp1
vp2i=vp2-h
up3=(bp3-bp1)/(bp-bpt)
```

```
vp3=bp*up3+bp1
vp3i=vp3-h
up4=(cp2-cp1)/(cp-cpt)
vp4=cp*up4+cp1
vp4i=vp4-h
up5=(cp3-cp1)/(cp-cpt)
vp5=cp*up5+cp1
vp5i=vp5-h
up6=(ap3-ap1)/(ap-apt)
vp6=ap*up6+ap1
vp6i=vp6-h
```

Com isto, todos os pontos necessários para a definição do punção no seu estado inicial estão definidos. As últimas variáveis a ser definidas são as variáveis referentes ao processo.

```
#var do processo

ht=round(h-t) #deslocamento do punçao
pv=1000.0 #velocidade do punçao
mct=ht/pv #tempo intermedio de calculo do step
ct=mct*2 #tempo de calculo do step
```

Passa-se então para a implementação do código relativo à simulação em si, sendo o que foi feito até agora, uma ajuda indispensável para a formulação e simplificação do que se explicará a seguir.

Começa-se então pela simples definição do modelo.

```
#model

modelo = mdb.Model(name='Model-1')
```

Utilizando as variáveis anteriormente definidas, define-se o esboço da matriz.

```
# Create Die
# Sketch

dieprofile=modelo.ConstrainedSketch(name='dieprofile',
sheetSize=1000.0)
dieprofile.Line(point1=(x0, y0), point2=(x1, y1))
dieprofile.Line(point1=(x1, y1), point2=(x2, y2))
dieprofile.Line(point1=(x2, y2), point2=(x3, y3))
dieprofile.Line(point1=(x3, y3), point2=(x4, y4))
dieprofile.Line(point1=(x4, y4), point2=(x5, y5))
dieprofile.FilletByRadius(curve1=dieprofile.geometry.fin
dAt((((x1-x0)/2+x0), ((y1-y0)/2+y0)),),
curve2=dieprofile.geometry.findAt((((x2-x1)/2+x1), ((y2-
y1)/2+y1)),), nearPoint1=(u1, v1), nearPoint2=(u2, v2),
radius=r1)
dieprofile.FilletByRadius(curve1=dieprofile.geometry.fin
dAt((((x3-x2)/2+x2), ((y3-y2)/2+y2)),),
curve2=dieprofile.geometry.findAt((((x4-x3)/2+x3), ((y4-
y3)/2+y3)),), nearPoint1=(u5, v5), nearPoint2=(u6, v6),
radius=r3)
dieprofile.FilletByRadius(curve1=dieprofile.geometry.fin
dAt((((x2-x1)/2+x1), ((y2-y1)/2+y1)),),
curve2=dieprofile.geometry.findAt((((x3-x2)/2+x2), ((y3-
y2)/2+y2)),), nearPoint1=(u3, v3), nearPoint2=(u4, v4),
radius=r2)
```

E assim, chega-se ao que corresponde no ABAQUS/CAE à Figura 29.

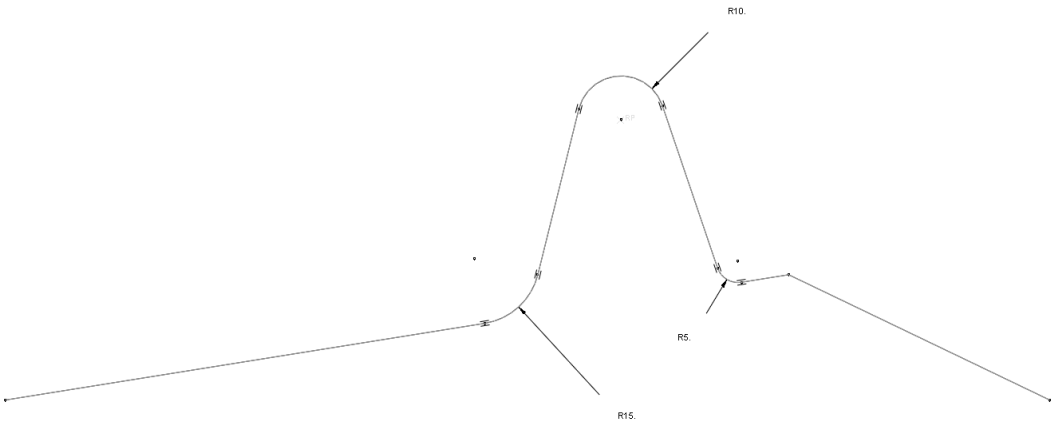

Figura 29 – Representação do esboço de metade simétrica da matriz no ABAQUS/CAE.

De seguida define-se a peça utilizando o esboço anteriormente implementado, considerando a peça como uma superfície rígida analítica de duas dimensões.

```
# Part

diepart=modelo.Part(dimensionality=TWO_D_PLANAR,
name='PART-DIE', type=ANALYTIC_RIGID_SURFACE)
diepart.AnalyticRigidSurf2DPlanar(sketch=dieprofile)
```

Com isto obtém-se o que corresponde no ABAQUS/CAE ao que se pode observar na Figura 30.

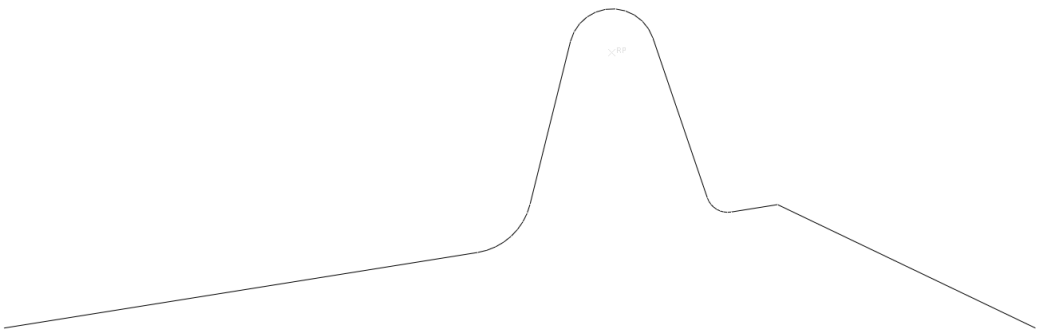

Figura 30 – Representação da peça de metade simétrica da matriz no ABAQUS/CAE.

Segue-se então a definição da superfície e do ponto de referência relativos à matriz.

```
# Surfaces and sets

diepart.Surface(name='SURF-DIE',
side2Edges=diepart.edges.findAt((((x1-x0)/2+x0), ((y1-
y0)/2+y0), 0.0), (u1, v1, 0.0)), (((x2-x1)/2+x1), ((y2-
y1)/2+y1), 0.0), (u3, v3, 0.0)), (((x3-x2)/2+x2), ((y3-
y2)/2+y2), 0.0),(u5, v5, 0.0)), (((x4-x3)/2+x3), ((y4-
y3)/2+y3), 0.0),(x4, y4, 0.0)), ))
diepart.ReferencePoint(point=(xr2, yr2, 0.0))
diepart.Set(name='SET-DIE',
referencePoints=(diepart.referencePoints[3], ))
```

Terminando isto, tem-se a matriz definida. Segue-se para o punção utilizando uma metodologia semelhante à que foi aplicada para a matriz. Inicia-se então pela definição do esboço utilizando as variáveis anteriormente definidas, obtendo o que corresponde ao representado na Figura 31.

```
# Create Punch
# Sketch

punchprofile=modelo.ConstrainedSketch(name=
'punchprofile', sheetSize=1000.0)
punchprofile.Line(point1=(xp0,yp0i), point2=(xp1,yp1i))
punchprofile.Line(point1=(xp1,yp1i), point2=(xp2,yp2i))
punchprofile.Line(point1=(xp2,yp2i), point2=(xp3,yp3i))
punchprofile.Line(point1=(xp3,yp3i), point2=(xp4,yp4i))
punchprofile.Line(point1=(xp4,yp4i), point2=(xp5,yp5i))
punchprofile.FilletByRadius(curve1=punchprofile.geometry
.findAt((((xp1-xp0)/2+xp0), ((yp1i-yp0i)/2+yp0i)),),
curve2=punchprofile.geometry.findAt((((xp2-xp1)/2+xp1),
((yp2i-yp1i)/2+yp1i)),), nearPoint1=(up1, vp1i),
nearPoint2=(up2, vp2i), radius=rp1)
```

```
punchprofile.FilletByRadius(curve1=punchprofile.geometry
.findAt((((xp3-xp2)/2+xp2), ((yp3i-yp2i)/2+yp2i)),),
curve2=punchprofile.geometry.findAt((((xp4-xp3)/2+xp3),
((yp4i-yp3i)/2+yp3i)),), nearPoint1=(up5, vp5i),
nearPoint2=(up6, vp6i), radius=rp3)
punchprofile.FilletByRadius(curve1=punchprofile.geometry
.findAt((((xp2-xp1)/2+xp1), ((yp2i-yp1i)/2+yp1i)),),
curve2=punchprofile.geometry.findAt((((xp3-xp2)/2+xp2),
((yp3i-yp2i)/2+yp2i)),), nearPoint1=(up3, vp3i),
nearPoint2=(up4, vp4i), radius=rp2)
```

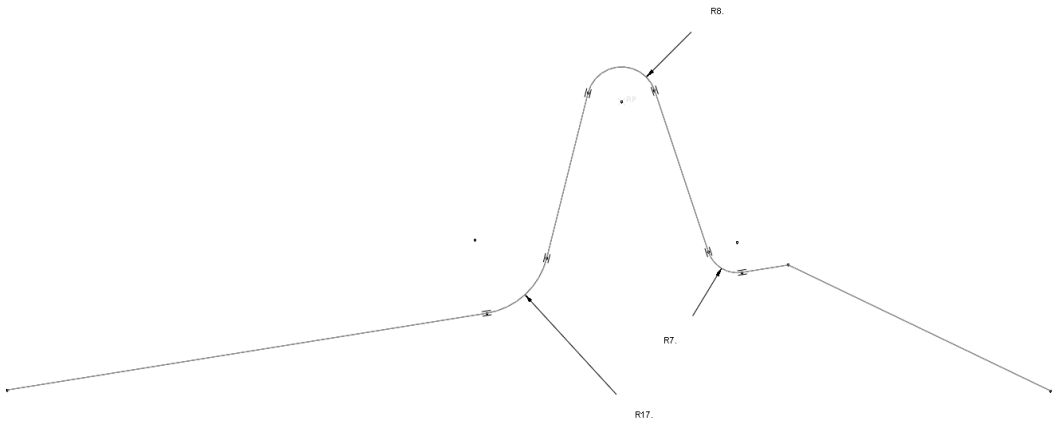

Figura 31 – Representação do esboço de metade simétrica do punção no ABAQUS/CAE.

De seguida, define-se a peça utilizando o esboço que foi implementado anteriormente. Tal como no caso da matriz a peça é uma superfície rígida analítica em duas dimensões. Pode-se observar o resultado correspondente no ABAQUS/CAE na Figura 32.

```
# Part

punchpart=modelo.Part(dimensionality=TWO_D_PLANAR,
name='PART-PUNCH', type=ANALYTIC_RIGID_SURFACE)
punchpart.AnalyticRigidSurf2DPlanar(sketch=punchprofile)
```

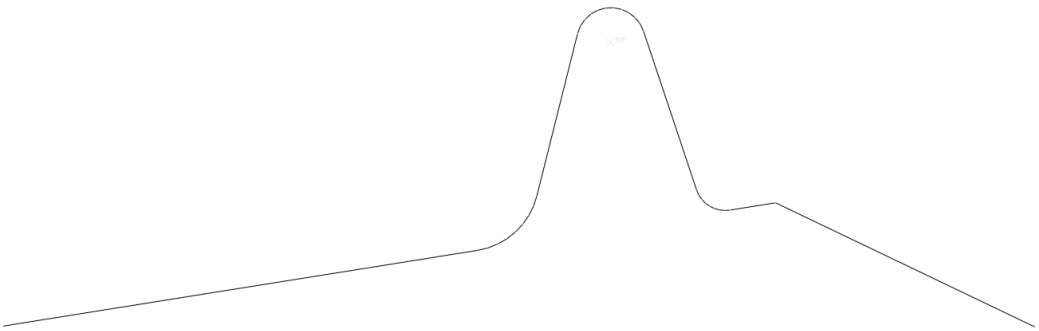

Figura 32 – Representação da peça de metade simétrica do punção no ABAQUS/CAE.

Por conseguinte, define-se a superfície e o ponto de referência relativos ao punção.

```
# Create surfaces and sets

punchpart.Surface(name='SURF-PUNCH',
side1Edges=punchpart.edges.findAt((((xp1-xp0)/2+xp0),
((yp1i-yp0i)/2+yp0i), 0.0), (up1, vp1i, 0.0)), ((((xp2-
xp1)/2+xp1), ((yp2i-yp1i)/2+yp1i), 0.0), (up3, vp3i,
0.0)), ((((xp3-xp2)/2+xp2), ((yp3i-yp2i)/2+yp2i),
0.0),(up5, vp5i, 0.0)), ((((xp4-xp3)/2+xp3), ((yp4i-
yp3i)/2+yp3i), 0.0),(xp4, yp4i, 0.0)), ))
punchpart.ReferencePoint(point=(xpr2, ypr2i, 0.0))
punchpart.Set(name='SET-PUNCH',
referencePoints=(punchpart.referencePoints[3], ))
```

Definidas todas as ferramentas, passa-se para a definição da chapa, começando pelo esboço. Com isto, obtém-se o representado na Figura 33.

```
# Create Blank
# Sketch

blankprofile=modelo.ConstrainedSketch(name='blankprofile
', sheetSize=1000.0)
blankprofile.rectangle(point1=(0.0, 0.0), point2=(240.0,
-t))
```

Figura 33 – Representação do esboço de metade simétrica da chapa no ABAQUS/CAE.

Posteriormente, Implementa-se a peça, a qual é um objeto deformável em duas dimensões, e se pode observar na Figura 34.

```
# Part

blankpart=modelo.Part(dimensionality=TWO_D_PLANAR,
name='PART-BLANK', type=DEFORMABLE_BODY)
blankpart.BaseShell(sketch=blankprofile)
```

Figura 34 – Representação da peça de metade simétrica da chapa no ABAQUS/CAE.

Ao contrário das ferramentas, a chapa tem duas superfícies de referência, a superior e a inferior, e uma aresta de referência, a aresta central que define o eixo de simetria.

```
# Create Sets and Surfaces

blankpart.Surface(name='SURF-BLANKNEG',
side1Edges=blankpart.edges.findAt(((100.0, -t, 0.0),
(0.0, -t, 0.0)), ))
blankpart.Surface(name='SURF-BLANKPOS',
side2Edges=blankpart.edges.findAt(((100.0, 0.0, 0.0),
(0.0, 0.0, 0.0)), ))
blankpart.Set(edges=blankpart.edges.findAt(((0.0, -t/2,
0.0), ), ), name='SET-CENTER')
```

Terminando isto, só resta a implementação da inércia nos pontos de referência da matriz e do punção para terminar a implementação das peças para o caso de estudo.

```
# Create Inertia


punchpart.engineeringFeatures.PointMassInertia(alpha=0.0
, composite=0.0, mass=0.1, name='INERTIA-PUNCH',
region=punchpart.sets['SET-PUNCH'])
diepart.engineeringFeatures.PointMassInertia(alpha=0.0,
composite=0.0, mass=0.1, name='INERTIA-DIE',
region=diepart.sets['SET-DIE'])
```

De seguida, implementa-se o material, que é o mesmo que foi aplicado no caso de estudo 1. É então definida a densidade, o módulo de elasticidade, o coeficiente de Poisson e as propriedades plásticas. O modelo plástico é definido através de pontos que descrevem uma curva tensão/deformação. Por último, define-se o nível de anisotropia deste material através da função potencial.

```
material=modelo.Material(name='material')
material.Density(table=((7.87e-09, ), ))
material.Elastic(table=((206.629E3, 0.298), ))
```

```
material.Plastic(table=((175.0,0.0), (180.47,0.01),
(208.17,0.02), (226.3,0.03), (240.12,0.04), (251.41,0.05),
(261.03,0.06), (269.45,0.07), (276.97,0.08), (283.77,0.09),
(289.99,0.1), (295.74,0.11), (301.09,0.12), (306.1,0.13),
(310.8,0.14), (315.25,0.15), (319.47,0.16), (323.49,0.17),
(327.32,0.18), (330.98,0.19), (334.5,0.2), (337.88,0.21),
(341.13,0.22), (344.27,0.23), (347.3,0.24), (350.23,0.25),
(353.07,0.26), (355.83,0.27), (358.5,0.28), (361.1,0.29),
(363.63,0.3), (366.1,0.31), (368.5,0.32), (370.84,0.33),
(373.13,0.34), (375.36,0.35), (377.55,0.36), (379.69,0.37),
(381.78,0.38), (383.83,0.39), (385.83,0.4), (387.8,0.41),
(389.73,0.42), (391.62,0.43), (393.48,0.44), (395.31,0.45),
(397.1,0.46), (398.86,0.47), (400.6,0.48), (402.3,0.49),
(403.98,0.5), (405.63,0.51), (407.26,0.52), (408.86,0.53),
(410.43,0.54), (411.99,0.55), (413.52,0.56), (415.03,0.57),
(416.52,0.58), (417.99,0.59), (419.44,0.6), (420.87,0.61),
(422.28,0.62), (423.68,0.63), (425.05,0.64), (426.41,0.65),
(427.75,0.66), (429.08,0.67), (430.39,0.68), (431.69,0.69),
(432.97,0.7), (434.24,0.71), (435.49,0.72), (436.73,0.73),
(437.95,0.74), (439.17,0.75), (440.37,0.76), (441.55,0.77),
(442.73,0.78), (443.89,0.79), (445.04,0.8), (446.18,0.81),
(447.31,0.82), (448.43,0.83), (449.54,0.84), (450.64,0.85),
(451.72,0.86), (452.8,0.87), (453.87,0.88), (454.93,0.89),
(455.97,0.9), (457.01,0.91), (458.04,0.92), (459.06,0.93),
(460.08,0.94), (461.08,0.95), (462.08,0.96), (463.06,0.97),
(464.04,0.98), (465.01,0.99), (465.98,1.0)))
material.plastic.Potential(table=((1.0000,1.03973,1.32188,1.12
798,1.0000,1.0000), ))
```

Posteriormente à definição do material, implementa-se a orientação deste na chapa, tal como o seu sistema de coordenadas.

```
# Create orientation

blankpart.DatumCsysByThreePoints(coordSysType=CARTESIAN,
name='ROLLDIR', origin=(0.0, 0.0,0.0), point1=(0.0, 0.0,
1.0), point2=(1.0, 0.0, 1.0))
blankpart.MaterialOrientation(additionalRotationField=''
, additionalRotationType=ROTATION_ANGLE, angle=0.0,
axis=AXIS_3, fieldName='', orientationType=SYSTEM,
region=Region(faces=blankpart.faces.findAt(((100.0, -
t/2, 0.0), (0.0, -t/2, 0.0)), )))
```

Passa-se então para a criação da secção do material.

```
# Create Section

modelo.HomogeneousSolidSection(material='material',
name='Section-Blank', thickness=None)
```

Aplica-se a secção criada na chapa, assim como a definição do material.

```
# Assign Section to Section

blankpart.SectionAssignment(offset=0.0, offsetField='',
offsetType=MIDDLE_SURFACE, region= Region(faces=
blankpart.faces.findAt(((100.0, -t/2, 0.0), (0.0, -t/2,
0.0)), )), sectionName='Section-Blank')
```

Por conseguinte implementa-se a assemblagem, na qual se insere todas as peças e se define os eixos globais.

```
# Create Assembly

modelo.rootAssembly.DatumCsysByDefault(CARTESIAN)
blankinst=modelo.rootAssembly.Instance(dependent=ON,
name='PART-BLANK-1', part=blankpart)
dieinst=modelo.rootAssembly.Instance(dependent=ON,
name='PART-DIE-1', part=diepart)
```

```
punchinst=modelo.rootAssembly.Instance(dependent=ON,
name='PART-PUNCH-1', part=punchpart)
```

Pode-se observar o que representa o código até aqui implementado no ABAQUS/CAE na Figura 35.

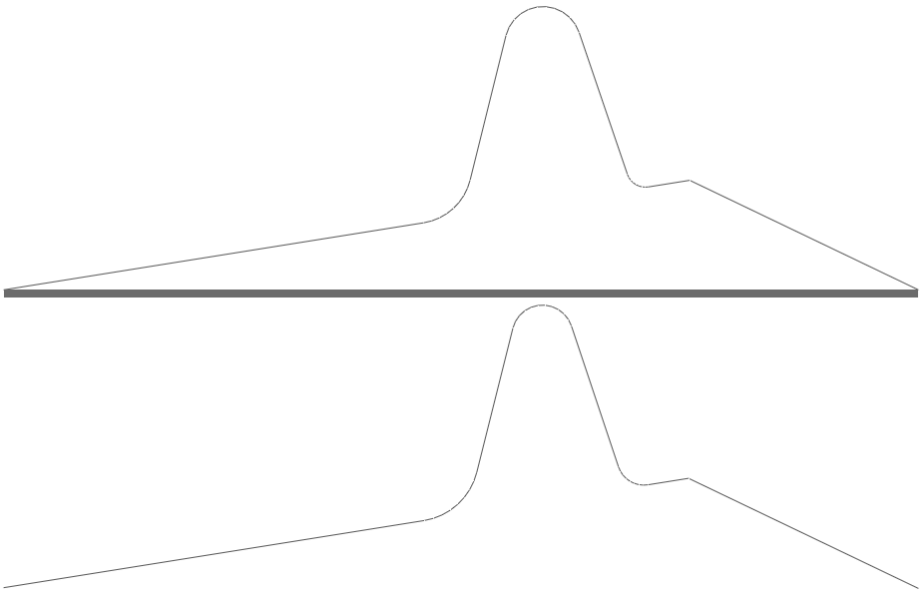

Figura 35 – Representação do assembly no ABAQUS/CAE.

De seguida, implementa-se a malha. Esta inicia-se pela definição dos critérios para aplicação da malha, nos quais indicam-se que os elementos são quadráticos e são aplicados através de uma técnica estruturada. Segue-se a definição do número de elementos por aresta. Por conseguinte, define-se o tipo de elemento, o qual é definido pelo código CPS4R[4]. Por último gera-se a malha e definem-se os nós de referência.

```
# Create Mesh


blankpart.setMeshControls(elemShape=QUAD,
technique=STRUCTURED,
regions=blankpart.faces.findAt(((100.0, -t/2, 0.0),
(0.0, -t/2, 0.0)), ))
```

[4] O elemento CPS4R é um elemento casca de 4 nós e integração reduzida

```
blankpart.seedEdgeByNumber(constraint=FINER,
edges=blankpart.edges.findAt(((120.0, 0.0, 0.0), ),
((120.0, -t, 0.0), ), ),number=240)
blankpart.seedEdgeByNumber(constraint=FINER,
edges=blankpart.edges.findAt(((0.0, -t/2, 0.0), ),
((240.0, -t/2, 0.0), ), ),number=2)
blankpart.setElementType(elemTypes=
(ElemType(elemCode=CPS4R, elemLibrary=STANDARD)),
regions=(blankpart.faces.findAt(((100.0, -t/2, 0.0),
(0.0, -t/2, 0.0)),),))
blankpart.generateMesh()
blankpart.Set(name='SET-REFNODES',
nodes=blankpart.nodes[0:241])
```

Na Figura 36 encontra-se uma representação de como esta malha se representa no ABAQUS/CAE.

Figura 36 – Representação da malha da chapa no ABAQUS/CAE.

Posteriormente, são definidas as 4 etapas. Pode-se dividir as etapas por:

- Aproximação do punção a chapa,
- Subida do punção,
- Descida do punção,
- Subida da matriz.

Em cada uma destas etapas define-se o tempo que esta necessita até concluir, o número máximo de incrementos necessários, o tamanho do incremento mais pequeno e do incremento inicial e também se existe ou não linearidade na etapa.

Após as etapas definidas, definem-se as variáveis que se pretende estudar no ficheiro de saída (output).

```
# Create Steps

modelo.ImplicitDynamicsStep(description='Punch aprouch',
name='Step-1', previous='Initial', timePeriod=0.01,
maxNumInc=10000,nlgeom=ON,minInc=1E-15,initialInc=1E-15)
modelo.ImplicitDynamicsStep(description='Move Punch Up',
name='Step-2', previous='Step-1',timePeriod=ct,
maxNumInc=10000,nlgeom=ON,minInc=1E-15,initialInc=1E-15)
modelo.steps['Step-2'].Restart(numberIntervals=1,
overlay=OFF, timeMarks=OFF)
modelo.ImplicitDynamicsStep(description='Move Punch
Down', name='Step-3', previous='Step-2', timePeriod=ct,
maxNumInc=10000,nlgeom=ON,minInc=1E-15,initialInc=1E-15)
modelo.ImplicitDynamicsStep(description='Move Die Up',
name='Step-4', previous='Step-3', timePeriod=ct,
maxNumInc=10000,nlgeom=ON,minInc=1E-15,initialInc=1E-15)
modelo.fieldOutputRequests['F-Output-1'].setValues(
numIntervals=10, variables=('S', 'TRIAX', 'PE', 'PEEQ',
'LE', 'U', 'RF', 'CSTRESS', 'STH', 'COORD'))
```

Segue-se para a definição das propriedades das interações entre a chapa e as ferramentas, o punção e a matriz, definindo o atrito e as suas características em cada caso.

```
# Create Interaction Properties

modelo.ContactProperty('INTPROP-DIE')
modelo.interactionProperties['INTPROP-
DIE'].TangentialBehavior(dependencies=0,
directionality=ISOTROPIC, elasticSlipStiffness=None,
formulation=PENALTY, fraction=0.005,
maximumElasticSlip=FRACTION, pressureDependency=OFF,
shearStressLimit=None, slipRateDependency=OFF,
table=((0.01, ), ), temperatureDependency=OFF)
modelo.ContactProperty('INTPROP-PUNCH')
```

```
modelo.interactionProperties['INTPROP-
PUNCH'].TangentialBehavior(dependencies=0,
directionality=ISOTROPIC, elasticSlipStiffness=None,
formulation=PENALTY, fraction=0.005,
maximumElasticSlip=FRACTION, pressureDependency=OFF,
shearStressLimit=None, slipRateDependency=OFF,
table=((0.01, ), ), temperatureDependency=OFF)
```

De seguida aplicam-se as propriedades das interações de contacto entre as superfícies.

```
# Create Interactions


modelo.SurfaceToSurfaceContactStd(clearanceRegion=None,
createStepName='Initial', datumAxis=None,
initialClearance=OMIT,interactionProperty='INTPROP-DIE',
master=dieinst.surfaces['SURF-DIE'], name='INT-DIE',
slave=blankinst.surfaces['SURF-BLANKPOS'],
sliding=FINITE)
modelo.SurfaceToSurfaceContactStd(clearanceRegion=None,
createStepName='Initial', datumAxis=None,
initialClearance=OMIT, interactionProperty='INTPROP-
PUNCH', master=punchinst.surfaces['SURF-PUNCH'], name=
'INT-PUNCH', slave=blankinst.surfaces['SURF-BLANKNEG'],
sliding=FINITE)
```

Segue-se então a definição das condições fronteira, as quais foram divididas em dois tipos: as relativas aos deslocamentos e as relativas a velocidade.

Nas condições fronteiras referentes aos deslocamentos, começa-se pela definição das condições inicias do processo, anulando todos os deslocamentos. Segue-se para a definição da aproximação do punção na respetiva etapa, com a posterior permissão do movimento na etapa em que o

punção sobe e desce e por ultimo a fixação dos deslocamentos do centro na etapa em que se retira o punção.

```
# Create Boundary Conditions (displacement)

modelo.DisplacementBC(amplitude=UNSET, createStepName=
'Initial', distributionType=UNIFORM, fieldName='',
localCsys=None, name='BC-DIE', region=
dieinst.sets['SET-DIE'], u1=SET, u2=SET, ur3=SET)
modelo.DisplacementBC(amplitude=UNSET, createStepName=
'Initial', distributionType=UNIFORM, fieldName='',
localCsys=None, name='BC-PUNCH', region=
punchinst.sets['SET-PUNCH'], u1=SET, u2=SET, ur3=SET)
modelo.boundaryConditions['BC-
PUNCH'].setValuesInStep(stepName='Step-1', u2=t)
modelo.boundaryConditions['BC-
PUNCH'].setValuesInStep(stepName='Step-2', u2=FREED)
modelo.DisplacementBC(amplitude=UNSET, createStepName=
'Initial', distributionType=UNIFORM, fieldName='',
localCsys=None, name='BC-CENTER', region=
blankinst.sets['SET-CENTER'], u1=SET, u2=UNSET, ur3=SET)
modelo.boundaryConditions['BC-
CENTER'].setValuesInStep(stepName='Step-3', u2=SET)
```

Nas condições fronteira relativas às velocidades, inicia-se pela criação de incrementação linear de uma determinada amplitude segundo um determinado tempo. Segue-se para aplicação desta incrementação linear em conjunto com a velocidade às etapas, nas quais existe a deformação da chapa através do punção, tanto na subida como na descida, e da matriz, na sua subida.

```
# Create Boundary Conditions (velocity)

modelo.TabularAmplitude(data=((0.0, 0.0), (mct, 1.0),
(ct, 0.0)), name='AMP-PUNCH', smooth=SOLVER_DEFAULT,
timeSpan=STEP)
modelo.VelocityBC(amplitude='AMP-PUNCH',
createStepName='Step-2', distributionType=UNIFORM,
fieldName='', localCsys=None, name='BCp+1000',
region=punchinst.sets['SET-PUNCH'],v1=UNSET, v2=pv,
vr3=UNSET)
modelo.VelocityBC(amplitude='AMP-PUNCH',
createStepName='Step-3', distributionType=UNIFORM,
fieldName='', localCsys=None, name='BCp-1000',
region=punchinst.sets['SET-PUNCH'],v1=UNSET, v2=-pv,
vr3=UNSET)
modelo.VelocityBC(amplitude='AMP-PUNCH',
createStepName='Step-4', distributionType=UNIFORM,
fieldName='', localCsys=None, name='BCd+1000',
region=dieinst.sets['SET-DIE'],v1=UNSET, v2=pv,
vr3=UNSET)
```

Por último existe a definição do ensaio, tal como a sua submissão.

```
# Create Job

job=mdb.Job(atTime=None,contactPrint=OFF,description='',
echoPrint=OFF, historyPrint=OFF, model= 'Model-1',
modelPrint=OFF, multiprocessingMode= DEFAULT, name='T',
nodalOutputPrecision=SINGLE, numCpus=1, numDomains= 1,
queue=None,scratch='',type= ANALYSIS,userSubroutine='',
waitHours=0, waitMinutes=0)
job.submit()
job.waitForCompletion()
```

Com todo o ensaio implementado, tem-se o pré-processamento e processamento da simulação relativa ao caso de estudo 2 terminada. Na

Figura 37 pode-se observar uma representação da chapa e das ferramentas no final da simulação.

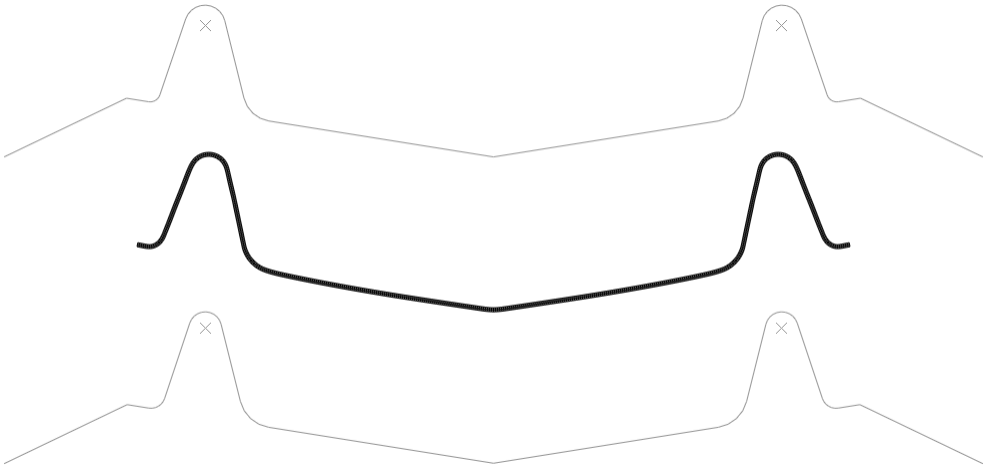

Figura 37 – Representação da geometria da chapa e da disponibilização das ferramentas no final da simulação no ABAQUS/VIEWER.

6. Otimização em conformação plástica

6.1. Introdução

A otimização é atualmente uma ferramenta importante que pode ser aplicada na conformação plástica. No presente trabalho existem vários objetivos no âmbito da otimização, entre eles:

- A definição da função objetivo através do *script* em linguagem Python,
- Criar uma subrotina que acople o *script* relativo ao ABAQUS e o método de otimização em estudo.
- Testar metodologia em casos de estudo.
- Usar casos de estudo com características diferentes.

Para conseguir atingir o acima mencionado, necessita-se da implementação do método do gradiente conjugado, com as alterações de Fletcher-Reeves, aos casos em estudo. Adicionalmente necessita-se de criar uma interface capaz de acoplar as simulações do ABAQUS com o método de otimização, tendo como ponte de ligação a função objetivo e as variáveis de entrada.

Com a implementação de todo o problema em estudo na simulação em função das variáveis de entrada e da obtenção de uma função objetivo, e com a ajuda da aplicação de uma interface que executa a simulação obtendo a função objetivo, de seguida, e através do método de otimização, minimiza-se esta função alterando as variáveis de entrada. As variáveis de otimização são usadas para correr a simulação novamente, seguindo este ciclo até se obter a melhor otimização possível no problema em estudo. Este processo pode-se observar na Figura 38.

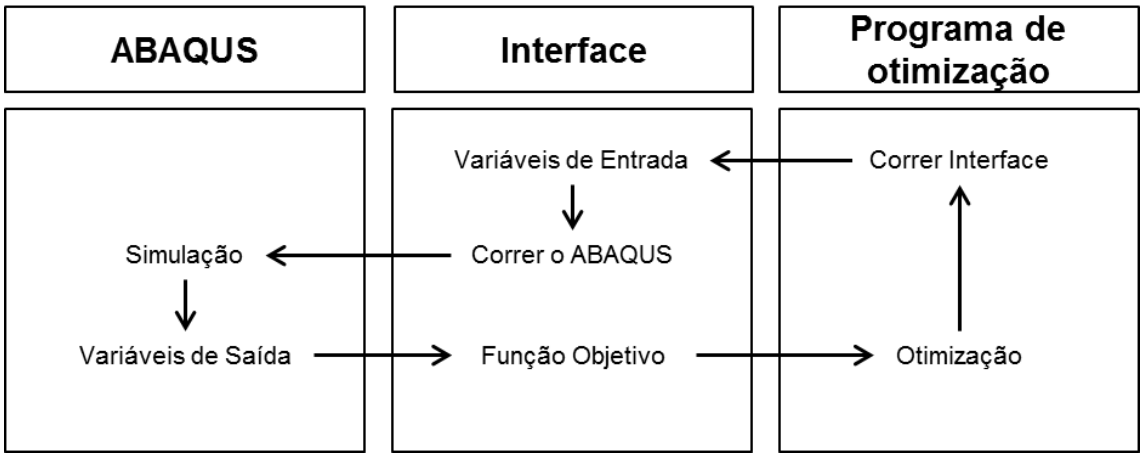

Figura 38 – Esquema da ligação dos vários programas utilizados através do interface.

Aplicando isto ao caso de estudo 1, é criada a metodologia de otimização de forma a minimizar o retorno elástico. A função a minimizar é calculada através da diferença de três ângulos, atingido isto a alteração de três variáveis de entrada: a força do cerra-chapas, o raio do punção e o raio da matriz.

Tal como no caso de estudo referido anteriormente, no caso de estudo 2 também se minimiza o retorno elástico. Para este caso, o retorno elástico é calculado através de diferença de declives de três retas. Para chegar a este resultado alteram-se as variáveis de entradas referentes a três raios presentes na geometria da matriz e punção.

6.2. Caso de estudo 1

Divide-se a implementação da otimização no caso de estudo 1 em três partes. A primeira parte é o cálculo da função objetivo utilizando *script*. A interface da simulação utilizando *script*, que consiste num *script* que inclui a simulação através da execução dos outros *scripts* aplicados. E por último, a função objetivo utilizando uma subrotina Fortran, a qual é responsável por acoplar o *script* interface ao algoritmo de otimização.

6.2.1. Função objetivo utilizando *script*

Para a aplicação da metodologia de otimização no caso de estudo 1 é necessário o cálculo da função objetivo, sendo esta calculada através de um *script* Python. Para tal é necessário utilizar o código que vai ser explicado em seguida.

Inicia-se, como em qualquer *script* Python ligado ao ABAQUS, pela importação das funções que serão utilizadas neste *script*.

```
from odbAccess import *
import sys
import math
import os
```

Após isto, define-se o odb e os ficheiros a serem escritos neste caso de estudo.

```
# abrir odb e output

odb = openOdb(path='S.odb')
resfile = open('S0.dat', "w")
out0file = open('OS0.dat', "w")
out1file = open('OS1.dat', "w")
```

De seguida escolhe-se a última etapa do processo de estampagem, a peça que será estudada, e os nós de referência.

```
# ultimo step part e number
# definir as cordenadas

refnodeset='SET-REFNODES'
stepname=odb.steps.keys()[-1]
partname=odb.rootAssembly.instances.keys()[0]
nframes=len(odb.steps[stepname].frames)
refnodes=odb.rootAssembly.instances[partname].nodeSets[r
efnodeset]
```

Depois, segue-se para a importação da posição de todos os nós no início e no fim da última etapa para cálculos posteriores e, se necessário, poder-se identificar possíveis erros, caso o programa não estiver a funcionar da forma correta.

```
# achar frame data

for iframe in [0,-1]: # only first and last frames
    frame=odb.steps[stepname].frames[iframe]
    try:
        coordinates=
frame.fieldOutputs['COORD'].getSubset(region= refnodes)
    for icoord in coordinates.values:
        if iframe == 0:
            format = '%15.7e %15.7e %15.7e\n'
            out0file.write(format %
(tuple(icoord.data)))
        else:
            format = '%15.7e %15.7e %15.7e\n'
            out1file.write(format %
(tuple(icoord.data)))
```

Com o implementado até agora, inicia-se o cálculo dos ângulos necessários para chegar a função objetivo. Começa-se por encontrar o ponto central da chapa.

```
#calcular os angulos teta1 teta2 e raios
# achar nos

XO=coordinates.values[-1].data[0]
YO=coordinates.values[-1].data[1]
```

Seguindo para procura de dois pontos na face vertical da chapa, sendo um deles através do ponto anteriormente calculado.

```
# calcular ponto A

for iA in range(len(coordinates.values)-1,-1,-1):
    XA=coordinates.values[iA].data[0]
    YA=coordinates.values[iA].data[1]
    DIST_OA=abs(YO-YA)
    if DIST_OA>15.:
        break
for iB in range(iA,-1,-1):
    XB=coordinates.values[iB].data[0]
    YB=coordinates.values[iB].data[1]
    DIST_AB=sqrt((XA-XB)**2+(YA-YB)**2)
    if DIST_AB>25.:
        break
```

De seguida, propõe-se dois pontos na face horizontal da chapa junto as extremidades.

```
# calcular ponto E e F

XE=coordinates.values[39].data[0]
YE=coordinates.values[39].data[1]
XF=coordinates.values[9].data[0]
YF=coordinates.values[9].data[1]
```

Com estes quatro pontos encontrados, pode-se calcular os três ângulos usados para o cálculo da função objetivo. O primeiro ângulo, teta1, é o ângulo criado perto do raio do punção. O teta2 corresponde ao ângulo criado perto do raio da matriz e alfa é o ângulo da face junto as extremidades.

```
# calcular anglos e cutvaturar

ALFA1=atan((YB-YA)/(XB-XA))
ALFA2=atan((YE-YF)/(XF-XE))
TETA1=180.*(1.-ALFA1/math.pi)
TETA2=180.*(1.-(ALFA1+ALFA2)/math.pi)
```

```
ALFA=180.*ALFA2/math.pi
format = '%15.7e %15.7e %15.7e\n'
resfile.write(format %(TETA1, TETA2, ALFA))
```

Para terminar o programa fecha-se os ficheiros de texto que se abriram no início do programa.

```
out0file.close()
out1file.close()
resfile.close()
odb.close()
```

6.2.2. Interface da subrotina utilizando *script*

Para a interligação de todos os *scripts* que foram implementados no caso de estudo 1 é utilizado um *script* interface. Este é responsável pela leitura das variáveis de entrada da simulação a partir de um ficheiro, tal como a execução do pré-processamento, da simulação, do pós-processamento, e no final define as variáveis necessárias para calcular a função objetivo e escreve-as num ficheiro. Assim, pode-se correr todos os *scripts* referentes a simulação sem a necessidade de os correr individualmente e, simultaneamente, facilita a alteração dos ficheiros usados.

No início, tal como em todos os *scripts* até agora explicados, importam-se todas as funções que serão usadas neste *script* e em todos os *scripts* que são evocados por este.

```
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
```

```
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
from abaqus import *
from abaqusConstants import *
import visualization
import os
import datetime
import shutil
from odbAccess import *
import time
```

De seguida, lêem-se as variáveis de entrada do *script* de pré-processamento explicado na secção 5.2, e define-se estas variáveis neste *script*.

```
par=open('Si.txt','r')
hf=par.readline()
rp=par.readline()
rd=par.readline()
hf=abs(float(hf))
rp=abs(float(rp))
rd=abs(float(rd))
par.close()
```

Com isto, passa-se para a execução do *script* de pré-processamento e a submissão da simulação (processamento).

```
execfile("S1.py")
mdb.saveAs(pathName='S.cae')
job.submit()
job.waitForCompletion()
```

Em seguida, executa-se o *script* de pós-processamento.

```
execfile("S2.py")
```

Com isto a implementação do código necessário para correr a simulação numérica do caso de estudo 1 em ABAQUS esta terminada.

6.2.3. Função objetivo utilizando subrotina

Para a otimização do caso de estudo tem que se correr o *script* em Python da simulação numérica no código Fortran onde está implementado o método de otimização.

Para tal é necessário a implementação de uma subrotina que sempre que for nomeada executa os *scripts* da simulação em ABAQUS e escreve e lê todas as variáveis necessárias para o fazer.

No início, define-se a subrotina a ser utilizada e os dados que esta necessita para correr.

```
subroutine func(n, x, f, NFV)
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
DIMENSION X(1)
real*8 :: MeanAngle, TETA1, TETA2, f
write(*,*)"in func"
```

A seguir, elimina-se o ficheiro de saída da simulação anterior para que, se por algum motivo a simulação não acabe como esperado, esta não induza em erro os cálculos da otimização. Sem este procedimento e no caso de erro de simulação a otimização utilizaria os resultados anteriormente recebidos.

```
OPEN (UNIT=5, FILE='S0.dat', STATUS='UNKNOWN')
CLOSE (UNIT=5, STATUS="DELETE")
```

Após isto, escreve-se as variáveis de entrada da simulação numérica no ficheiro a que esta recorre para a leitura dos mesmos. Na primeira vez que este programa é executado, este toma valores iniciais indicados pelo utilizador, mas a partir desse momento, ele altera estes valores de forma a minimizar a função objetivo.

```
open(unit=34,file="Si.txt")
write(*,*)"x=",x(1),x(2),x(3)
do i=1,n
  write(34,*)x(i)
enddo
close(34)
```

Com isto, é de seguida executado o *script* Python para obter a função objetivo.

```
call system('abaqus cae nogui=S.py')
```

Depois da simulação terminar, importam-se as variáveis de saída através de leitura do ficheiro onde estes são escritos.

```
open(unit=35,file='S0.dat')
read(35,*)TETA1,TETA2,MeanAngle
close(35)
```

Com estes, calcula-se então a função objetivo, que consiste na soma quadrática da diferença entre estes valores e os valores para os quais estes devem-se aproximar.

```
Lim1 = 90.
Lim2 = 90.
Lim3 = 0.
penal1=(TETA1-Lim1)**2
penal2=(TETA2-Lim2)**2
penal3=(MeanAngle-Lim3)**2
f = penal1+penal2+penal3
write(*,*)"Valor da funcao:",f
write(*,*)"Angles=",TETA1,TETA2,MeanAngle
write(*,*)"para o limite de",Lim1,Lim2,Lim3
```

Com a função calculada só resta terminar a subrotina para que uma iteração finalize.

```
return
end
```

Terminado isto, a subrotina que acopla a simulação é terminada e o método de otimização prossegue iterativamente. O programa baseado no método do gradiente conjugado com as alterações de Fletcher-Reeves necessita de executar esta subrotina inúmeras vezes, inserindo valores de entrada diferentes de forma a minimizar a função objetivo que foi considerada para este caso de estudo.

6.2.4. Resultados da otimização do caso de estudo 1

Utilizando o que foi implementado na secção 5.2 e nas subsecções 6.2.1, 6.2.2 e 6.2.3 e com o programa baseado no método referido na secção 3.3 tem-se uma evolução das variáveis de entrada que levou a uma diminuição da função objetivo. Esta evolução pode ser observada através das Figuras 39 e 40 e da Tabela 2.

Tabela 2 – Valores referentes ao caso de estudo 1.

Força do cerra-chapas (KN)	Raio do punção (mm)	Raio da Matriz (mm)	Função objetivo ($^{\circ 2}$)
750,000000000000	5,00000000000000	5,00000000000000	33,8435462897630
749,999993956111	4,99538502268695	5,00158128697313	33,8129493473730
749,999993960554	4,99537614584953	5,00158953945459	33,4254475485309
749,999993962387	4,99537475237950	5,00158978339829	31,0742099077616
749,999993962545	4,99537448668900	5,00158981143079	28,5365613659832
749,999993962400	4,99537441529778	5,00158977183143	27,8321079080946
749,999993962398	4,99537441483863	5,00158977156468	27,8267270878356
749,999993962398	4,99537441475090	5,00158977151610	27,8265744513921
749,999993962398	4,99537441474072	5,00158977151033	27,8265744513921

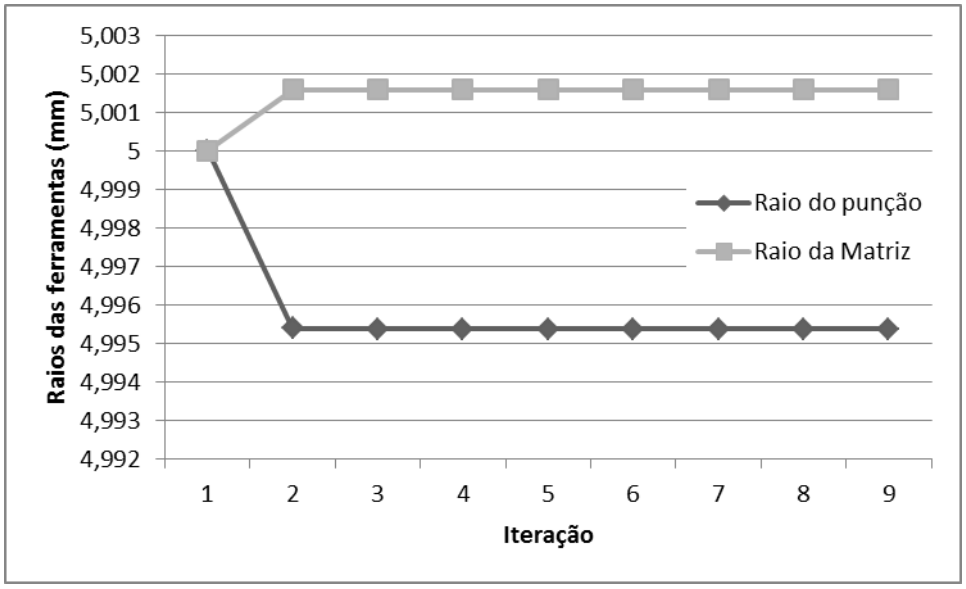

Figura 39 – Evolução das variáveis referentes ao caso de estudo 1.

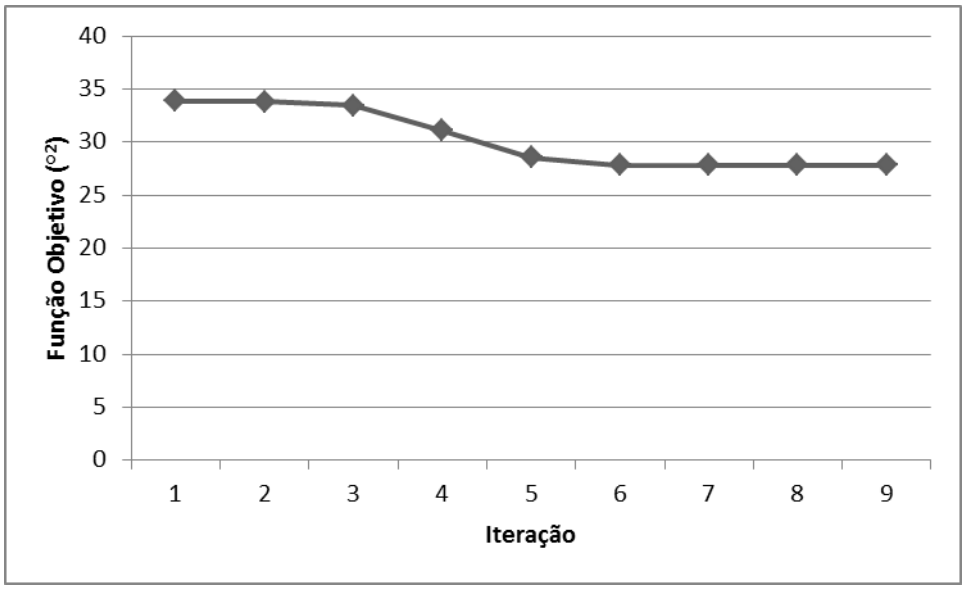

Figura 40 – Evolução da função objetivo referentes ao caso de estudo 1.

Como se pode observar através da Tabela 2 e da Figura 40, ouve um melhoramento de aproximadamente 18% da função objetivo no final do programa em relação ao início. O melhoramento baixo sugere que: ou obteve-se um mínimo local e não um mínimo geral ou os valores de entrada iniciais já estariam muito próximos dos valores ótimos.

Através das Figuras 41, 42 e 43 pode-se observar a pequena melhoria anteriormente indicada do retorno elástico, a qual era o objetivo deste caso de estudo.

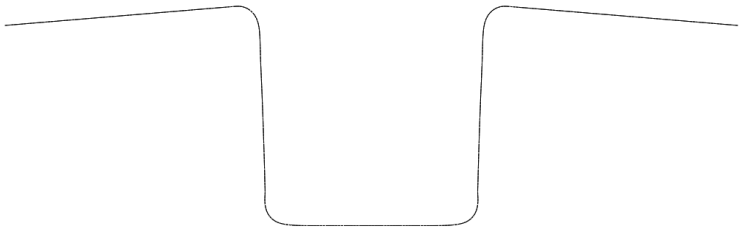

Figura 41 – Representação da chapa após a conformação plástica na primeira iteração.

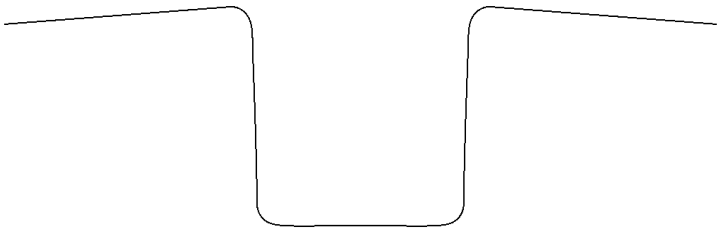

Figura 42 – Representação da chapa após a conformação plástica na última iteração.

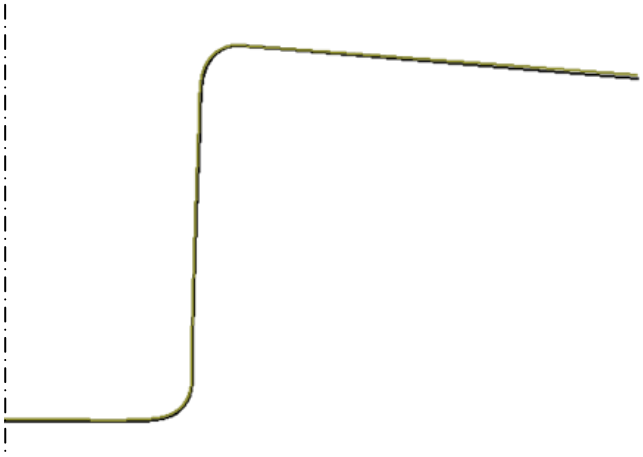

Figura 43 – Sobreposição das formas conformadas inicial e final do processo de otimização, sendo a mais escura a inicial e a mais clara a final.

6.3. Caso de estudo 2

Tal como no caso de estudo anterior a otimização é dividida essencialmente em três grandes partes. A primeira é a obtenção da função objetivo utilizando o *script* Python. A segunda é uma interface de toda a simulação utilizando novamente o *script* com linguagem Python. E a ultima é referente à subrotina utilizando a linguagem Fortran que é responsável pela execução do *script* de interface da simulação, tal como a modificação das variáveis de entrada e a obtenção das variáveis de saída no algoritmo de otimização.

6.3.1. Função objetivo utilizando *script*

Para o cálculo da função objetivo, a qual se pretende minimizar, é necessário após a conformação plástica terminar calcular o declive de três retas, que estejam em contacto com a matriz durante a conformação, e comparar estes valores com os valores do declive das retas da matriz em contacto com estas.

Para iniciar qualquer *script* em linguagem Python para o ABAQUS é necessário a importação das funções que serão utilizadas *a posteriori*.

```
from odbAccess import *
import sys
import math
import os
```

Segue-se o cálculo das distâncias entre pontos extremos dos arcos e a coordenada central e entre pontos extremos de diferentes arcos, para seguidamente serem utilizadas para a obtenção dos declives.

```
#var

xab=u1-x0 #distancia-x extremo1-arco1 centro
xoc=u2-x0 #distancia-x extremo2-arco1 centro
```

```
xcd=u3-u2 #distancia-x extremo1-arco2 extremo2-arco1
ycd=v3-v2 #distancia-y extremo1-arco2 extremo2-arco1
xoe=u4-x0 #distancia-x extremo2-arco2 centro
xef=u5-u4 #distancia-x extremo1-arco3 extremo2-arco2
yef=v5-v4 #distancia-y extremo1-arco3 extremo2-arco1
```

Procede-se então para a abertura do ficheiro de Output Database (odb) e os ficheiros em que se escreve as coordenadas de todos os nós em estudo para estudo posterior.

```
# odb e output

odb = openOdb(path='T.odb')
out0file = open('co0.txt', "w")
out1file = open('co1.txt', "w")
```

Depois define-se a etapa de análise, sendo esta a ultima.

```
# step part number
# coord

refnodeset='SET-REFNODES'
stepname=odb.steps.keys()[-1]
partname=odb.rootAssembly.instances.keys()[0]
nframes=len(odb.steps[stepname].frames)
refnodes=odb.rootAssembly.instances[partname].nodeSets[
refnodeset]
```

De seguida encontram-se os valores da posição de todos os elementos no início e no fim da última etapa da conformação, sendo escritas em ficheiros de texto. Estes serão utilizados posteriormente em cálculos e, se for necessário, serão utilizados pelo utilizador para observar onde possa existir possíveis erros.

```
# frame data

for iframe in [0,-1]: # only first and last frames
    frame=odb.steps[stepname].frames[iframe]
    try:
        coordinates=
frame.fieldOutputs['COORD'].getSubset(region=refnodes)
    for icoord in coordinates.values:
        if iframe == 0:
            format = '%15.7e %15.7e\n'
            out0file.write(format %
(tuple(icoord.data)))
        else:
            format = '%15.7e %15.7e\n'
            out1file.write(format %
(tuple(icoord.data)))
```

Segue-se a obtenção das coordenadas dos nós que serão utilizados para o cálculo dos três declives. A maioria destes pontos foram calculados tendo como base as abcissas antes do retorno elástico. Por essa razão este caso de estudo apresenta bons resultados para pequenas e medias movimentações das abcissas. Começa-se pela obtenção das coordenadas do nó central, definidas pelo código seguinte:

```
#calcular os declives
# achar nos

XO=coordinates.values[-1].data[0]
YO=coordinates.values[-1].data[1]
```

Para o cálculo do primeiro declive é necessário obter as coordenadas de dois pontos que se encontram na reta deste. Esses dois pontos são os pontos A e B, definido por:

```
# calcular ponto A e B

for iA in range(len(coordinates.values)-1,-1,-1):
    XA=coordinates.values[iA].data[0]
    YA=coordinates.values[iA].data[1]
    DIST_OA=abs(XO-XA)
    if DIST_OA>(0.1*xab):
        break

for iB in range(iA,-1,-1):
    XB=coordinates.values[iB].data[0]
    YB=coordinates.values[iB].data[1]
    DIST_AB=sqrt((XA-XB)**2+(YA-YB)**2)
    if DIST_AB>(xab*0.25):
        break
```

Após estes, passa-se para o cálculo do segundo declive, no qual é necessário obter os pontos C e D.

```
# calcular ponto C e D

for iC in range(len(coordinates.values)-1,-1,-1):
    XC=coordinates.values[iC].data[0]
    YC=coordinates.values[iC].data[1]
    DIST_OC=abs(XO-XC)
    if DIST_OC>(xoc+xcd*0.1):
        break

for iD in range(iC,-1,-1):
    XD=coordinates.values[iD].data[0]
    YD=coordinates.values[iD].data[1]
    DIST_CD=sqrt((XD-XC)**2+(YD-YC)**2)
    if DIST_CD>(sqrt(xcd**2+ycd**2)*0.25):
        break
```

Por último, são obtidos os responsáveis pelo cálculo do último declive, os quais se denominam por E e F.

```
# calcular ponto E e F

for iE in range(len(coordinates.values)-1,-1,-1):
    XE=coordinates.values[iE].data[0]
    YE=coordinates.values[iE].data[1]
    DIST_OE=abs(XO-XE)
    if DIST_OE>(xoe+xef*0.1):
        break

for iF in range(iE,-1,-1):
    XF=coordinates.values[iF].data[0]
    YF=coordinates.values[iF].data[1]
    DIST_EF=sqrt((XF-XE)**2+(YF-YE)**2)
    if DIST_EF>(sqrt(xef**2+yef**2)*0.25):
        break
```

A seguir a obter as coordenadas de todos os 6 pontos necessários, calculam-se os declives.

```
# calcular declives

mab=(YB-YA)/(XB-XA)
mcd=(YD-YC)/(XD-XC)
mef=(YF-YE)/(XF-XE)
```

Sabendo os declives que foram necessários para criar a matriz e os calculados anteriormente procedeu-se para o cálculo dos desvios dos declives, sendo estes caracterizados entre a diferença dos declives das retas da superfície da chapa que esteve em contacto com a matriz e o declive das retas da matriz.

```
ea=sqrt((mab-a)**2)
eb=sqrt((mcd-b)**2)
ec=sqrt((mef-c)**2)
```

No final os ficheiros de texto e o ficheiro Output Database que foram abertos no início são fechados.

```
out0file.close()
out1file.close()
odb.close()
```

6.3.2. Interface da subrotina utilizando *script*

Para uma simplificação de todos os *scripts* até agora aplicados no caso de estudo 2 é utilizado um *script* interface. Este lê as variáveis de entrada da simulação a partir de um ficheiro, executa o pré-processamento, a simulação, o pós-processamento, e no final calcula a função objetivo e escreve-a num ficheiro. Desta forma pode-se correr toda a simulação e o pós-processamento sem a necessidade de chamar vários *scripts* e, simultaneamente, facilita a alteração dos ficheiros usados.

Inicialmente, importam-se todas as funções que serão aplicadas neste *script* e em todos os *scripts* que este executa.

```
from odbAccess import *
from abaqusConstants import *
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
from abaqus import *
```

```
from abaqusConstants import *
import visualization
import os
import datetime
import shutil
import time
import sys
import subprocess
```

Passa-se para a importação das variáveis de entrada, sendo estas os raios usados para a geometria da matriz e para o cálculo dos raios do punção.

```
par=open('ci.txt','r')
r1=par.readline()
r2=par.readline()
r3=par.readline()
r1=abs(float(r1))
r2=abs(float(r2))
r3=abs(float(r3))
par.close()
```

Segue-se a execução do *script* de pré-processamento e da simulação da conformação plástica da chapa, o qual já foi explicado na secção 5.3.

```
execfile("c1.py")
mdb.saveAs(pathName='T.cae')
job.submit()
job.waitForCompletion()
```

De seguida, implementa-se o código que irá executar o *script* do pós-processamento (ver subsecção anterior) obtendo as variáveis necessárias para o cálculo da função objetivo.

```
execfile("c2.py")
```

Com a soma das variáveis de saída, que define os desvios dos declives, calcula-se a função objetivo.

```
F=str(ea+eb+ec)
```

No final escreve-se a função objetivo num ficheiro de texto, para esta ser importada e minimizada no algoritmo de otimização.

```
funcob=open('co.txt', 'w')
funcob.write(F)
funcob.close()
```

Assim, todos os *scripts* referentes à simulação do caso de estudo estão implementados.

6.3.3. Função objetivo utilizando subrotina

De maneira a minimizar as condições que se pensa desnecessárias, neste caso de estudo o retorno elástico, tem que se correr um algoritmo de otimização que altera as funções de entrada tentando minimizar estas condições.

Para o descrito anteriormente acontecer dentro do programa de otimização, necessita-se de uma subrotina específica. Esta subrotina tem que alterar as variáveis entrada do caso de estudo, correr o programa de simulação e obter o resultado da função objetivo. Nesta subsecção explica-se como pode ser formulada esta subrotina para o presente caso de estudo.

Inicia-se pela importação das variáveis necessárias na subrotina em questão bem como a sua definição.

```
subroutine func(n, x, f, NFV)
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
DIMENSION X(1)
real*8 :: f
write(*,*)"in func"
```

De forma a prevenir a utilização de resultados não válidos elimina-se o ficheiro da anterior função objetivo. Assim, se de alguma forma a simulação executada no programa ABAQUS der algum tipo de erro e não termine, a otimização terminar automaticamente

```
open (UNIT=5, FILE='co.txt', STATUS='UNKNOWN')
close (UNIT=5, STATUS="DELETE")
```

Antes da execução da simulação, necessita-se alterar os valores de entrada, os quais são escritos num ficheiro de texto. Na primeira vez em que se corre o algoritmo, estes valores de entrada são pré-definidos pelo utilizador do programa de otimização.

```
open(unit=34,file="ci.txt")
write(*,*)"x=",x(1),x(2),x(3)
do i=1,n
  write(34,*)x(i)
enddo
close(34)
```

Com o anteriormente aplicado, passa-se para a execução da simulação executando o *script* que foi explicado na subsecção 6.3.2.

```
call system('abaqus cae nogui=c.py')
```

Após a simulação, lê-se a função objetivo através de um ficheiro de texto.

```
open(unit=35,file='co.txt')
read(35,*)f
close(35)
write(*,*)"Valor da funcao:",f
```

No final, termina-se a subrotina e volta-se ao programa de otimização com o valor requerido da função objetivo.

```
return
end
```

O programa de otimização executará esta subrotina inúmeras vezes, alterando os valores de entrada e obtendo diferentes valores da função objetivo. Este programa utiliza o algoritmo de otimização do gradiente conjugado com as alterações de Fletcher-Reeves, descrito na secção 3.3, e pretende-se que diminua a função até a um mínimo.

6.3.4. Resultados da otimização do caso de estudo 2

Com o que foi definido na secção 5.3 e nas subsecções 6.3.1, 6.3.2 e 6.3.3 e com um programa de otimização do método do gradiente conjugado com as alterações de Fletcher-Reeves diminuiu-se o retorno elástico deste caso de estudo. Pode-se comprovar este facto através da observação da tabela 3 e das Figuras 44 e 45.

Tabela 3 – Valores referentes ao caso de estudo 2.

Raio 1 (mm)	Raio 2 (mm)	Raio 3 (mm)	Função objetivo
15,0000000000000	10,0000000000000	5,00000000000000	1,235252750990000
14,9894217419012	10,9252033736430	4,99388697445781	0,110260547806000
14,9894216223916	10,9252006506528	4,99388688713667	0,110257823988000
14,9894216222776	10,9252006496714	4,99388688710432	0,110208184688000
14,9894216221960	10,9252006488857	4,99388688706720	0,108204298375000
14,9894216221960	10,9252006488857	4,99388688706719	0,108204298375000

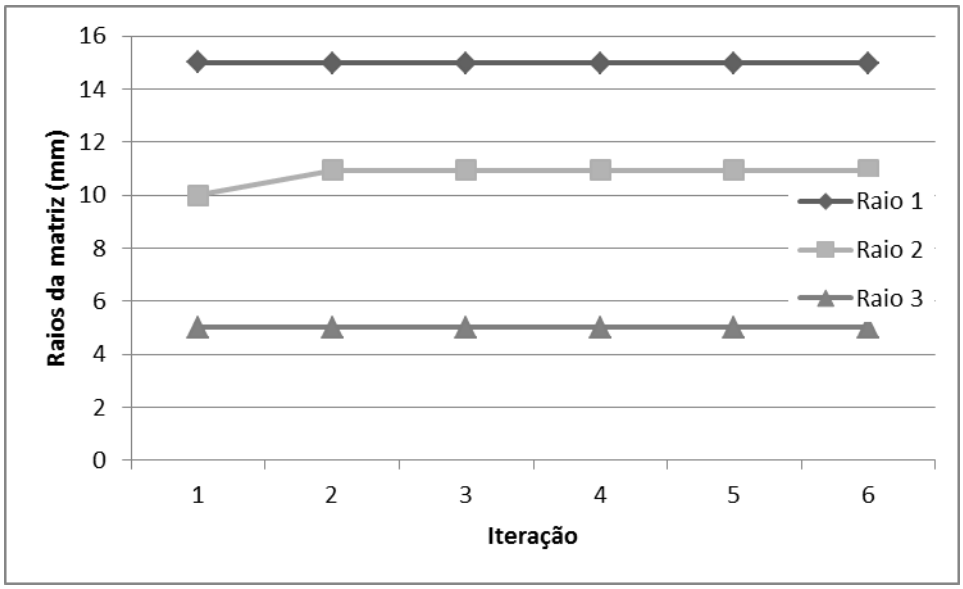

Figura 44 – Evolução das variáveis referentes ao caso de estudo 2.

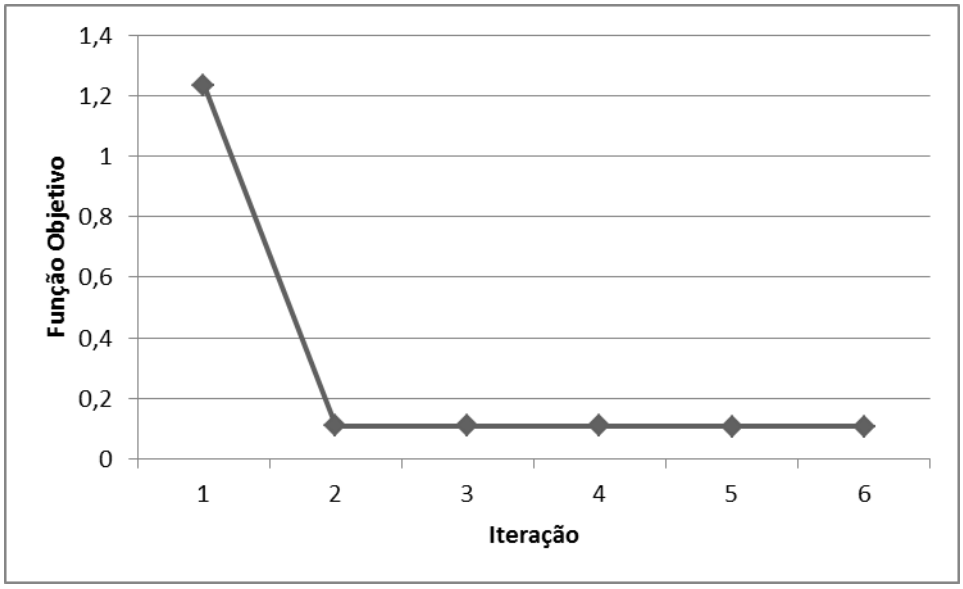

Figura 45 – Evolução da função objetivo referente ao caso de estudo 2.

Tendo em conta que alterar unicamente os fillets da geometria limita o resultado deste problema, não seria possível existirem melhorias muito significativas. Contudo, como se pode observar através da Tabela 3 e da Figura 45, neste caso de estudo existiu uma melhoria na função objetivo de aproximadamente 91%.

Nas Figuras 46 e 47 pode-se observar a chapa após a conformação plástica no início e no fim da otimização, respetivamente. E na Figura 48 pode-se observar a sobreposição das anteriores formas de maneira a comparar e observar a melhoria anteriormente indicada em relação ao retorno elástico, pois a forma que tem declives mais próximos da forma sem retorno elástico é a final.

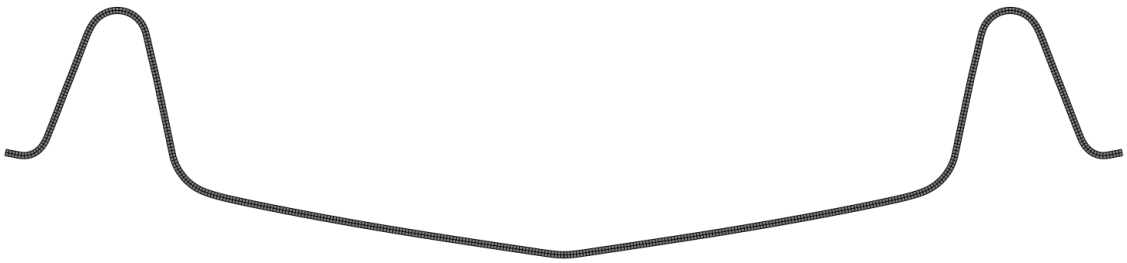

Figura 46 – Representação da chapa após a conformação plástica na primeira iteração.

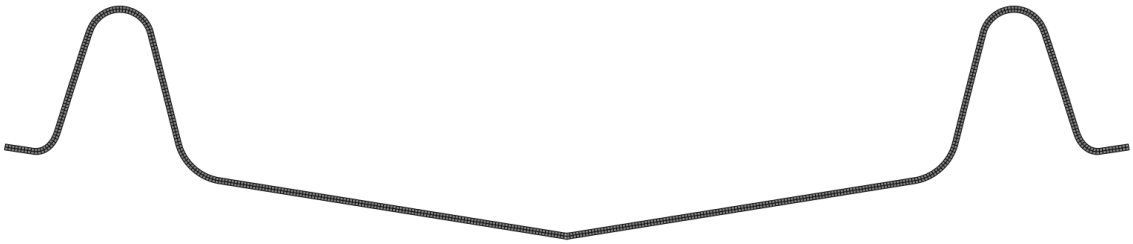

Figura 47 – Representação da chapa após a conformação plástica na última iteração.

Figura 48 – Sobreposição das formas comparadas inicial e final do processo de otimização, sendo a inicial a mais escura, a final a clara e a verde a forma inicial sem retorno elástico.

IV Discussão

7. Considerações finais

7.1. Conclusão geral

Atualmente, a simulação numérica é uma ferramenta de extrema importância no projeto da indústria de conformação plástica de chapas metálicas, pois esta permite a previsão de vários defeitos muitas vezes presentes neste processo. Alguns, como é o caso do retorno elástico, são fáceis de prever através da ajuda de um programa de simulação numérica, tal como é o caso ABAQUS. E com a correção de erros durante o projeto, diminui-se os custos da peça final.

No entanto, cada vez mais a previsão destes defeitos não é suficiente para a satisfação de todos os requisitos da indústria de conformação plástica de chapas metálicas. Existe uma necessidade de, sabendo a previsão dos defeitos, alterar a geometria da ferramenta de forma a obterem-se peças com o acabamento desejado.

O presente trabalho pretende ter uma participação positiva no desenvolvimento de ambas as áreas e na sua interligação. Chegando a conclusões que permitem esclarecer dúvidas e obter melhores capacidades dentro destas áreas com grande potencial a nível industrial.

Sumariamente, as principais conclusões tiradas deste trabalho são:

- A simulação e a otimização interligadas propiciam excelentes respostas e melhoramentos inegáveis dentro dos processos industriais, sendo a indústria de conformação plástica de chapas metálicas um exemplo,
- A parametrização de problemas dentro da simulação é algo que facilita e permite a melhor resolução dos mesmos,
- A escolha das variáveis iniciais de entrada nos problemas de otimização é um processo complexo e que necessita de um bom conhecimento dentro desta área,
- A linguagem de programação Python é de fácil aprendizagem e uso, permitindo a abertura de muitas portas na simulação numérica e otimização na área de conformação plástica de chapas metálicas,

- A utilização de *scripts* em linguagem Python aplicados ao ABAQUS é uma enorme ferramenta, a qual permite retirar vantagens tanto da simulação como da interligação com a otimização.

7.2. Trabalhos futuros

Tendo em conta o presente trabalho realizado, sugere-se diversas propostas para os trabalhos futuros:

- Estudo da escolha de melhores variáveis iniciais de entrada para o presente trabalho,
- Estudo de NURBS, da sua aplicação na simulação numérica e na otimização utilizando *scripts* em linguagem Python,
- Desenvolvimento de outros casos de estudo relativos a conformação plástica de chapas metálicas, utilizando *scripts* em linguagem Python no programa ABAQUS, mas também relativos a outros processos e outras áreas da indústria utilizando estas ferramentas,
- Estudo e desenvolvimento de diversos algoritmos de otimização, permitindo a comparação e discussão de resultados.

Referências

[1] S. Silva, “Simulação numérica e optimização em conformação plástica de chapas metálicas,” *Universidade de Aveiro, Tese de Mestrado*, 2010.

[2] A. Santos, J. F. Duarte, and A. B. Rocha, *Tecnologia de Embutidura*, Colecção T. INEGI, 2005.

[3] P. Pião, “Modelação e Simulação Numérica de Processos de Conformação Plástica de Metais,” *Universidade de Aveiro, Tese de Mestrado*, 2010.

[4] P. Teixeira, “‘ Benchmarks ’ Experimentais e Modelação Numérica por Elementos Finitos de Processos de Conformação Plástica,” *Universidade de Aveiro, Tese de Doutoramento*, 2005.

[5] T. Grilo, “Estudo de modelos constitutivos anisotrópicos para chapas metálicas,” *Universidade de Aveiro, Tese de Mestrado*, 2011.

[6] D. Yang, D. Ahn, C. Lee, C. Park, and T. . Kim, “Integration of CAD/CAM/CAE/RP for the development of metal forming process,” *J. Mater. Process. Technol.*, vol. 125–126, pp. 26–34, 2002.

[7] H. A. Flegel, “The challenge of car manufacturing in the 21st century,” *Int. Conf. New Dev. Forg. Technol.*, pp. 135–150, 2001.

[8] J. L. C. Alves, “Simulação numérica do processo de estampagem de chapas metálicas, modelação mecânica e métodos numéricos,” *Universidade do Minho, Tese de Doutoramento*, 2003.

[9] J. Cao and M. Boyce, “Optimization of sheet metal forming processes by instabilty analysis,” *Proc. NUMIFORM 92, Simul. Mater. Process. Methods Appl. Shan–Fu Shen, Paul Dawson*, 1995.

[10] K. Roll, “Simulation of sheet metal forming–necessary developments in the future,” *Proc. Numisheet Conf. Interlaken, Switz.*, no. September, pp. 59–68, 2008.

[11] M. Tisza, “Numerical modelling and simulation in sheet metal forming,” *J. Mater. Process. Technol.*, vol. 151, no. 1–3, pp. 58–62, Sep. 2004.

[12] M. L. Wenner, “Overview - simulation of sheet metal forming,” *Numisheet 2005 Proceedings 6th Internatioal Conf. Numer. Simul. 3D Sheet Met. Form. Process.*, pp. 3–7, 2005.

[13] K. Lange, K. Pöhlandt, and R. Raghupathi, *Handbook of metal forming*. 1985.

[14] K. Kazama and N. Mori, “Report on experimental conditions for benchmarks,” *IMS/3DS - Digit. Die Des. Syst. Proj. Rep.*, 2001.

[15] J. Carvalho, “Metodologia de optimização de processos de conformação plástica,” *Universidade de Aveiro, Tese de Mestrado*, 2007.

[16] G. Sachs, “Automobile engineers,” *Pro. Inst. Automob. Eng.*, p. 588, 1935.

[17] X. P. Li, A. Messina, and P. P. Strona, “Numerical simulation requirements in automotive components manufacturing processes design,” *Proc. Met. Form. Simul. Ind.*, 1994.

[18] M. Rohleder, K. Roll, L. Menezes, M. Oliveira, A. Andersson, and F. Krantz, “Standardization of Input-Output Data for Benchmark Tests,” *MS/3DS - Digit. Die Des. Syst. Proj. Rep.*, 2002.

[19] F. Teixeira-Dias, R. Sousa, R. Valente, J. Pinho-da-Cruz, *Método dos Elementos Finitos*. Edições Técnicas e Profissionais, 2010.

[20] A. Antoniou and W. Lu, “Practical optimization, algorithms and engineering applications.” New York, 2007.

[21] A. Andrade-campos, “Modelação e Análise numérica do Comportamento Mecânico e Térmico de Ligas de Alumínio,” *Universidade de Aveiro, Tese de Doutoramento*, 2005.

[22] R. Silva, “Desenvolvimento de metodologias para identificação de parâmetros e otimização de forma em simulações numéricas de processos de conformação plástica,” *Universidade de Aveiro, Tese de Doutoramento*, 2012.

[23] J. F. M. Caseiro, “Estratégias evolucionárias de optimização de parâmetros reais,” *Universidade de Aveiro, Tese de Mestrado*, 2009.

[24] E. Chong and S. Zak, *An Introduction to Optimization*, 3rd Ed. John Wiley & Sons, Inc., 2001.

[25] P. Teixeira, A. Andrade-campos, A. D. Santos, F. M. A. Pires, and J. M. A. C. De Sá, “Optimization strategies for springback compensation in sheet metal forming,” *First ECCOMAS Young Investig. Conf.*, no. 24–27 April, 2012.