UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO
ENGENHARIA DE COMPUTAÇÃO

ENZO CUNHA DO NASCIMENTO

SIMULAÇÃO DAS FORMAS POLARES DE MÁQUINAS SÍNCRONAS

GOIÂNIA
2024

UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO NO REPOSITÓRIO INSTITUCIONAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio do Repositório Institucional (RI/UFG), regulamentado pela Resolução CEPEC no 1240/2014, sem ressarcimento dos direitos autorais, de acordo com a Lei no 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo dos Trabalhos de Conclusão dos Cursos de Graduação disponibilizado no RI/UFG é de responsabilidade exclusiva dos autores. Ao encaminhar(em) o produto final, o(s) autor(a)(es)(as) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do Trabalho de Conclusão de Curso de Graduação (TCCG)

Nome(s) completo(s) do(a)(s) autor(a)(es)(as): ENZO CUNHA DO NASCIMENTO

Título do trabalho: SIMULAÇÃO DAS FORMAS POLARES DE MÁQUINAS SÍNCRONAS

2. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador) Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante: a) consulta ao(à)(s) autor(a)(es)(as) e ao(à) orientador(a); b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo do TCCG. O documento não será disponibilizado durante o período de embargo.

Casos de embargo:

- Solicitação de registro de patente;

- Submissão de artigo em revista científica;

- Publicação como capítulo de livro.

Obs.: Este termo deve ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **Bernardo Pinheiro De Alvarenga**, **Professor do Magistério Superior**, em 31/07/2024, às 19:26, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Enzo Cunha Do Nascimento**, **Discente**, em 31/07/2024, às 19:28, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **4707698** e o código CRC **AE7260BA**.

ENZO CUNHA DO NASCIMENTO

SIMULAÇÃO DAS FORMAS POLARES DE MÁQUINAS SÍNCRONAS

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia Elétrica, Mecânica e de Computação da Universidade Federal de Goiás como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Bernardo Pinheiro de Alvarenga

GOIÂNIA
2024

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Nascimento, Enzo Cunha do
SIMULAÇÃO DAS FORMAS POLARES DE MÁQUINAS SÍNCRONAS [manuscrito] / Enzo Cunha do Nascimento. - 2024.
60 f.: il.

Orientador: Prof. Dr. Bernardo Pinheiro de Alvarenga.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Escola de Engenharia Elétrica, Mecânica e de Computação (EMC), Engenharia da Computação, Goiânia, 2024.
Bibliografia. Apêndice.
Inclui gráfico, tabelas, algoritmos, lista de figuras, lista de tabelas.

1. Máquinas Síncronas. 2. Geometria. 3. Simulação por Elementos Finitos. 4. Análise de Dados. I. Alvarenga, Bernardo Pinheiro de, orient. II. Título.

CDU 621.3

UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO

Ao(s) vinte e seis dias do mês de julho do ano de dois mil e vinte e quatro iniciou-se a sessão pública de defesa do Trabalho de Conclusão de Curso (TCC) intitulado “SIMULAÇÃO DAS FORMAS POLARES DE MÁQUINAS SÍNCRONAS”, de autoria de ENZO CUNHA DO NASCIMENTO, do curso de ENGENHARIA DE COMPUTAÇÃO, da Escola de Engenharia Elétrica, Mecânica e de Computação da UFG. Os trabalhos foram instalados pelo(a) prof. Bernardo Alvarenga/EMC-UFG (orientador) com a participação dos demais membros da Banca Examinadora: prof. Enes Gonçalves Marra (EMC-UFG) e prof. José Wilson Lima Nerys (EMC-UFG). Após a apresentação, a banca examinadora realizou a arguição do estudante. Posteriormente, a Banca Examinadora atribuiu a nota final de 10 (dez) , tendo sido o TCC considerado APROVADO.

Proclamados os resultados, os trabalhos foram encerrados e, para constar, lavrou-se a presente ata que segue assinada pelos Membros da Banca Examinadora.

Documento assinado eletronicamente por **Bernardo Pinheiro De Alvarenga**, **Professor do Magistério Superior**, em 30/07/2024, às 19:46, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Enes Goncalves Marra**, **Professor do Magistério Superior**, em 30/07/2024, às 19:48, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Jose Wilson Lima Nerys**, **Professor do Magistério Superior**, em 30/07/2024, às 19:49, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **4707598** e o código CRC **3E5E7DA0**.

Referência: Processo nº 23070.015124/2024-47 SEI nº 4707598

AGRADECIMENTOS

Agradeço a todos os professores que fizeram parte da minha formação durante o meu tempo de estudo na UFG.

Ao meu orientador Prof. Dr. Bernardo Pinheiro de Alvarenga, com sua expertise crucial para a realização desse trabalho.

E especialmente aos meus pais, sem os quais nada disso seria possível.

RESUMO

As máquinas síncronas desempenham um papel crucial em sistemas de geração e transmissão de energia elétrica. Por isso, a simulação das formas polares de máquinas síncronas é muito útil para a análise do comportamento dessas máquinas em diferentes condições operacionais. Este trabalho foca na modelagem e simulação das formas polares das máquinas síncronas utilizando o aplicativo Flux2D. Inicialmente são apresentados conceitos básicos da operação do aplicativo e comandos de automação da simulação com Python de exemplos simples. Depois é realizada a simulação de densidade de fluxo magnético de entreferro para uma máquina síncrona de polos salientes de forma automatizada. O desempenho da máquina é analisado no regime magnetostático. É obtida uma rotina que determina uma geometria “otimizada” da cabeça polar em função de parâmetros geométricos.

Palavras-chaves: Máquinas Síncronas. Geometria. Simulação por Elementos Finitos. Análise de Dados.

ABSTRACT

Synchronous machines play a crucial role in power generation and transmission systems. Therefore, simulating the polar forms of synchronous machines is very useful for analyzing their behavior under various operational conditions. This study focuses on the modeling and simulation of the polar forms of synchronous machines using the Flux2D application. Initially, basic concepts of the application's operation and automation commands for simulation with Python are presented through simple examples. Then, the simulation of air gap magnetic flux density for a salient pole synchronous machine is performed in an automated manner. The machine's performance is analyzed in the magnetostatic regime. A routine is developed to determine an "optimized" pole head geometry based on geometric parameters.

Keywords: Synchronous Machines, Geometry, Simulation by Finite Elements, Data Analysis.

LISTA DE FIGURAS

Figura 1 - Criação de um parâmetro 12
Figura 2 - Criação de um sistema de coordenadas 13
Figura 3 – Criação de um ponto 14
Figura 4 – Criação de uma linha 14
Figura 5 – Criação das faces 15
Figura 6 – Resultado de uma transformação geométrica 16
Figura 7 – Definição da malha de elementos finitos 16
Figura 8 – Definição das regiões associadas a cada face 17
Figura 9 - Definição dos pontos 22
Figura 10 - Definição das linhas 23
Figura 11 – Identificando faces 23
Figura 12 – Resultado da transformação 24
Figura 13 – Fechando o domínio 25
Figura 14 – Ilustração da malha gerada pelos comandos da Tabela 9 26
Figura 15 – Relacionando regiões às faces 27
Figura 16 – Visualização da densidade de fluxo magnético 27
Figura 17 - Máquina síncrona com geometria de rotor de um arco 29
Figura 18 - Variação do raio externo do rotor 30
Figura 19 - Variação da largura da sapata polar 31
Figura 20 – Mapa de densidade de fluxo magnético – caso 1 32
Figura 21 – Configuração de fluxo magnético – caso 1 32
Figura 22 – Mapa de densidade de fluxo magnético – caso 2 33
Figura 23 – Configuração de fluxo magnético – caso 2 33
Figura 24 – Mapa de densidade de fluxo magnético – caso 3 34
Figura 25 – Configuração de fluxo magnético – caso 3 34
Figura 26 – Mapa de densidade de fluxo magnético – caso 4 35
Figura 27 – Configuração de fluxo magnético – caso 4 35
Figura 28 – Densidade de fluxo magnético – casos 1 a 4 36
Figura 29 – Densidade de fluxo magnético – caso 1 39
Figura 30 – Mapa de densidade de fluxo magnético – melhor caso 40
Figura 31 – Fluxo magnético – melhor caso 41
Figura 32 – Densidade de fluxo magnético de entreferro – melhor caso 41
Figura 33 – Espectro da densidade de fluxo magnético de entreferro – melhor caso 42

LISTA DE TABELAS

Tabela 1 - Comando para criação de parâmetros geométricos 20
Tabela 2 - Comando para criação de sistema de coordenadas 20
Tabela 3 - Comandos para criação dos pontos..... 21
Tabela 4 - Comandos para criação das linhas 22
Tabela 5 - Comandos para criar e utilizar uma transformação 24
Tabela 6 – Fechando o domínio 24
Tabela 7 - Comandos para criação da malha 25
Tabela 8 - Comandos para criar e atribuir regiões..... 26
Tabela 9 - Comandos para criação e resolução de cenário de simulação..... 27
Tabela 10 - Parâmetros para estudo da máquina síncrona 31
Tabela 11 - Fluxo e densidade de fluxo para os gráficos da Figura 28..... 36
Tabela 12 - Valores dos harmônicos para o espectro da curva da Figura 29..... 39

SUMÁRIO

RESUMO **6**
ABSTRACT **7**
LISTA DE FIGURAS **8**
LISTA DE TABELAS **9**
SUMÁRIO **10**
1 INTRODUÇÃO **11**
2 CAPACIDADES DO FLUX **12**
2.1 Modelagem geométrica 12
2.1.1 Parâmetros geométricos 12
2.1.2 Sistemas de coordenadas 13
2.1.3 Pontos e linhas 14
2.1.2 Faces 15
2.1.3 Transformações 15
2.1.4 Malha 16
2.1.5 Periodicidade 16
2.1.6 Materiais e fontes 17
2.1.7 Condições de contorno 18
3 INTEGRAÇÃO ENTRE PYTHON E FLUX **19**
3.1 Automação e reprodutibilidade 19
3.2 Análise de dados 19
3.3 Comandos básicos 20
3.3.1 Parâmetros geométricos 20
3.3.2 Sistemas de coordenadas 20
3.3.3 Pontos 21
3.3.4 Linhas 22
3.3.5 Faces 23
3.3.6 Transformações 24
3.3.7 Domínio 24
3.3.8 Malha 25
3.3.9 Materiais 26
3.3.10 Simulação 27
3.3.11 Aplicação 27
4 PROCEDIMENTOS **29**
4.1 Modelagem geométrica 29
4.2 Cenários de simulação 30
4.3 Resultados a partir da variação de parâmetros 31
4.3.1 Caso 1. 32
4.3.2 Caso 2. 33
4.3.3 Caso 3. 34
4.3.4 Caso 4. 35
4.3.5 Densidade de fluxo magnético de entreferro 36
5 ESTUDO DE MELHORIA DA GEOMETRIA **38**
5.1 A distorção harmônica 38
5.2 Resultados 40
6 CONCLUSÃO **43**
REFERÊNCIAS BIBLIOGRÁFICAS **44**
GLOSSÁRIO **46**
APÊNDICE A – Script PyFlux para modelagem geométrica da máquina de um arco **47**

1 INTRODUÇÃO

A crescente demanda por energia em escala global tem impulsionado o desenvolvimento de geradores mais eficientes, entre os quais as máquinas síncronas se destacam por serem responsáveis por maior parte da geração de energia no Brasil, onde 55% da matriz de energia elétrica é proveniente de usinas hidrelétricas [1].

Neste contexto, este trabalho utiliza o *software Flux2D* da Altair [2] — que permite a modelagem detalhada desde a geometria até a simulação do comportamento eletromagnético, térmico e mecânico das máquinas, possibilitando identificar problemas antes da construção de protótipos físicos ou otimizando projetos já existentes — e a linguagem de programação *Python*, que permite o envio de instruções previamente escritas e parametrizadas para o *Flux2D*.

A hipótese deste trabalho é que o uso da programação em *Python* permite a automação de todo o processo de construção, simulação e otimização no *Flux2D* enquanto garante precisão na reprodução do projeto. Os objetivos específicos para testar a hipótese são construir a geometria de máquinas síncronas (cujo desenho de cabeça polar é dito “de um arco”), determinar a densidade de fluxo magnético de entreferro e analisar os efeitos da variação de parâmetros geométricos de forma automática com a linguagem de programação *Python*.

Deste modo, o capítulo 2 descreve o processo de modelagem geométrica de um motor de indução disponibilizado pela Altair.

Depois, no capítulo 3 é mostrado a integração da linguagem de programação Python dentro do ambiente Flux para o processo de modelagem de um exemplo simples.

O capítulo 4 descreve os procedimentos realizados para automatizar o desenvolvimento de uma máquina síncrona específica utilizando dos conceitos apresentados nos capítulos anteriores.

O capítulo 5 apresenta um estudo para a melhoria da geometria de forma comparativa de acordo com o parâmetro de qualidade estabelecido.

Finalmente, a conclusão resume os resultados e discute propostas para a continuidade do presente trabalho.

2 CAPACIDADES DO FLUX

O *software* da Altair é uma ferramenta de simulação de campo eletromagnético que facilita o estudo detalhado de máquinas elétricas, incluindo máquinas síncronas. Suas capacidades abrangem múltiplas áreas de análise que devem ser pensadas durante o desenvolvimento e otimização de projetos envolvendo máquinas elétricas:

2.1 Modelagem geométrica

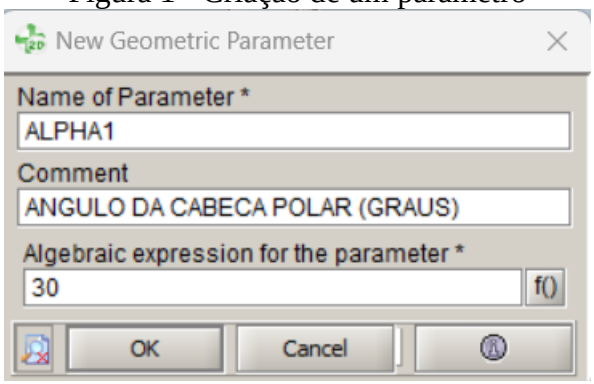
Como uma ferramenta de simulação, o *Flux2D* possui um ambiente de modelagem para definir um projeto do zero ou importar um modelo CAD (desenho assistido por computador) [3,4]. A seguir é ilustrada a modelagem de um motor de indução de acordo com o manual disponibilizado pela Altair [5].

2.1.1 Parâmetros geométricos

A primeira etapa da modelagem geométrica é a definição dos parâmetros geométricos. Os parâmetros geométricos podem ser um valor ou uma função de outros parâmetros, o que permite definir e alterar as próximas etapas de maneira dinâmica.

A Figura 1 ilustra a criação de um parâmetro numérico que define o ângulo da cabeça polar, utilizando a “interface” do programa Flux2D.

Figura 1 - Criação de um parâmetro

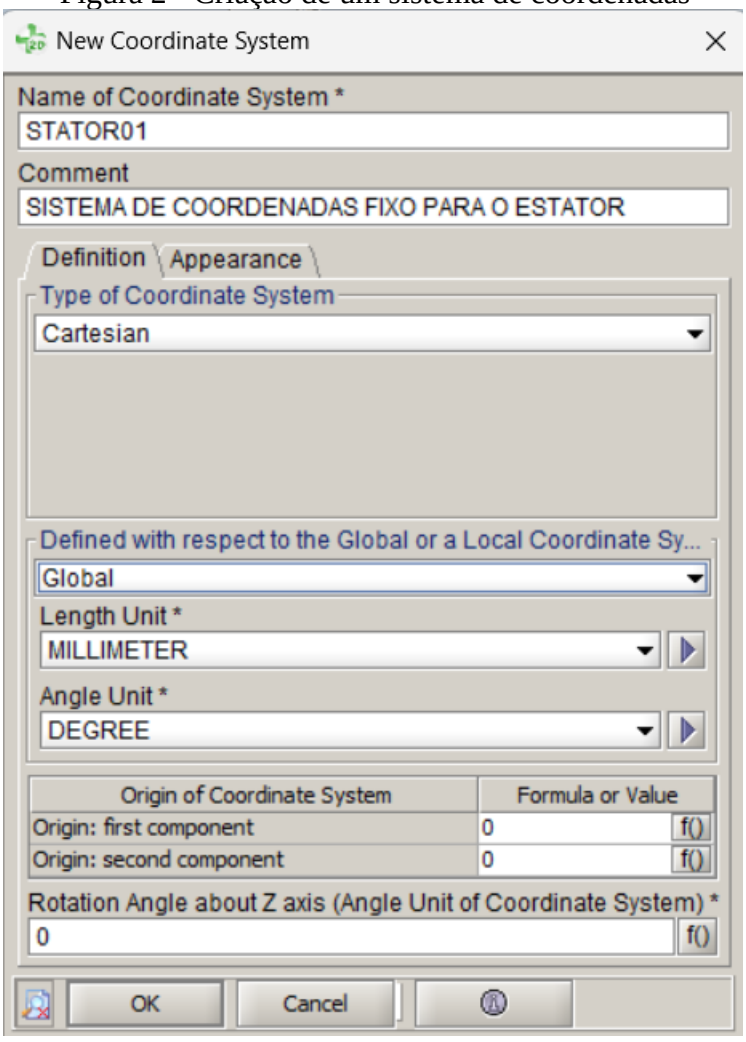


2.1.2 Sistemas de coordenadas

A seguir, é necessário definir os sistemas de coordenadas. Os sistemas de coordenadas facilitam a conversão entre coordenadas cartesianas centradas na origem e outras coordenadas e permitem definir a posição e orientação dos componentes do modelo que se movem, como o rotor, de maneira independente de outros componentes que permanecem fixos, como o estator.

A Figura 2 mostra a forma de criar os sistemas de coordenadas no aplicativo *Flux2D*.

Figura 2 - Criação de um sistema de coordenadas



2.1.3 Pontos e linhas

Os pontos e as linhas são os elementos que começam a dar forma às estruturas geométricas do projeto.

As Figuras 3 e 4 mostram a forma de criação de pontos e linhas, utilizando a “interface” do *Flux2D*. Observa-se que pontos e linhas criados podem se utilizar livremente de parâmetros e sistemas de coordenadas já criados.

Figura 3 – Criação de um ponto

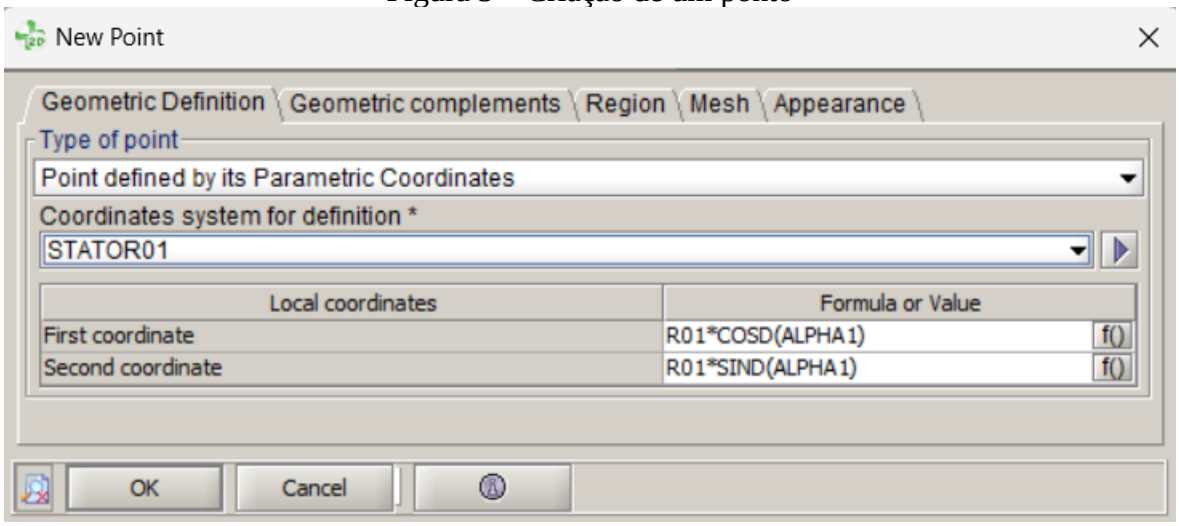
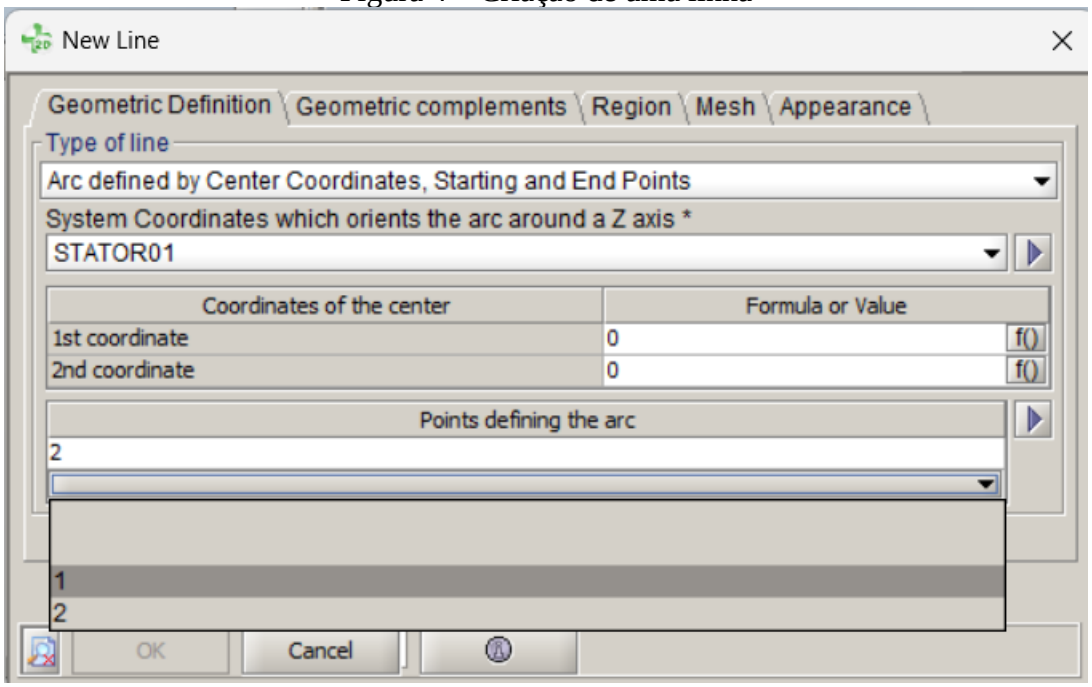


Figura 4 – Criação de uma linha

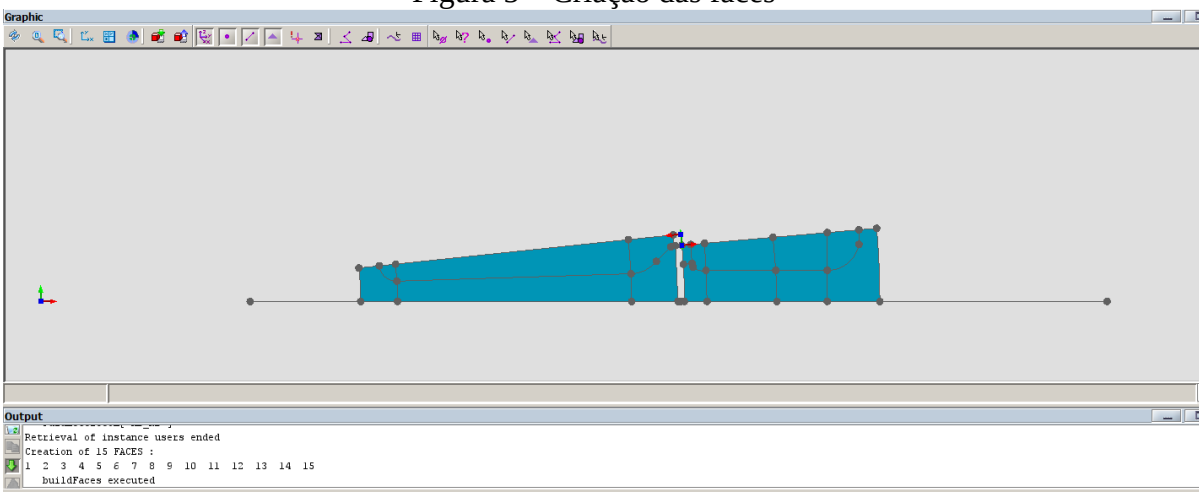


2.1.2 Faces

Nessa etapa, o *Flux2D* é capaz de identificar as estruturas fechadas formadas pelos segmentos de reta e arcos. A identificação destas estruturas fechadas resulta nas chamadas “faces”, que são regiões da tela que podem assumir propriedades físicas determinadas e constituem, por assim dizer, o domínio para a resolução das equações diferenciais que representam o tipo de estudo.

Como exemplo, a Figura 4 ilustra a modelagem das faces a partir de linhas previamente definidas.

Figura 5 – Criação das faces

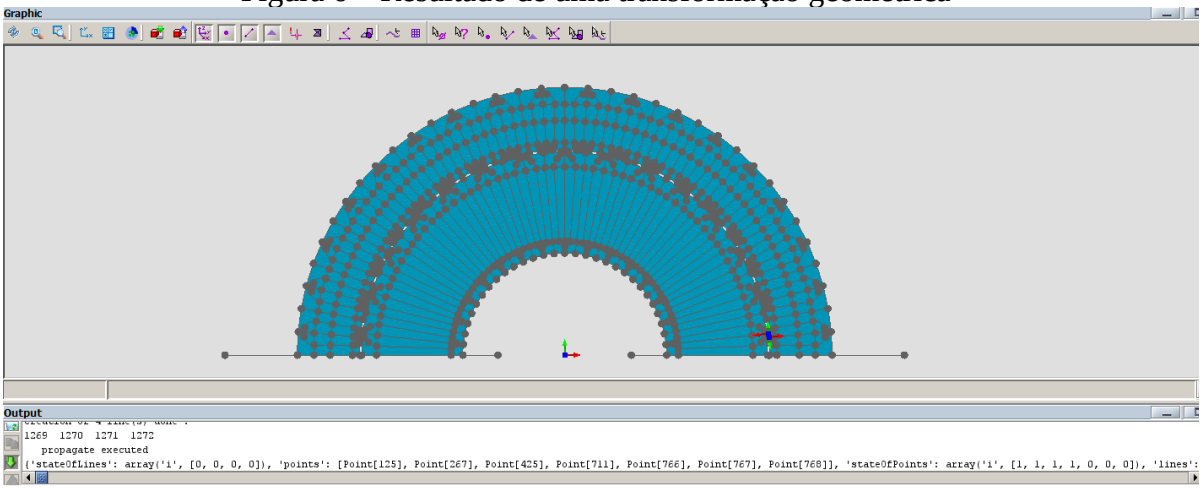


2.1.3 Transformações

Definida uma parte simétrica da geometria, o *Flux2D* permite rapidamente replicar uma combinação de pontos, linhas e faces por meio de transformações.

Como exemplo, a Figura 6, ilustra como é possível reproduzir partes simétricas de uma geometria.

Figura 6 – Resultado de uma transformação geométrica

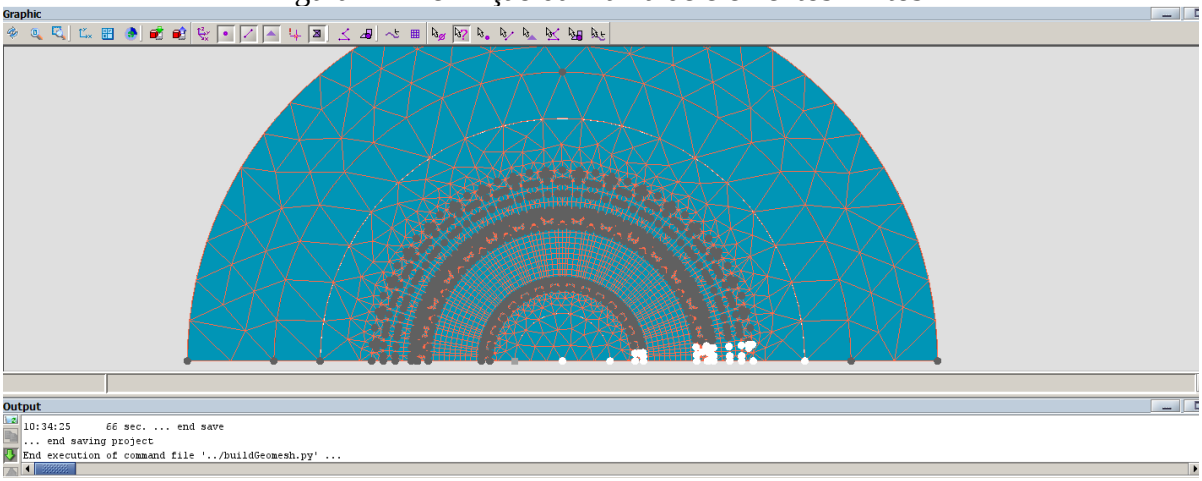


2.1.4 Malha

O próximo passo é a criação da malha, isso é, a discretização do domínio para que o *Flux2D* consiga resolver os cálculos necessários por métodos numéricos.

Como exemplo, a Figura 7 ilustra a malha gerada para a máquina da Figura 6 após alguns ajustes para fechar o domínio.

Figura 7 – Definição da malha de elementos finitos



2.1.5 Periodicidade

Pode-se verificar que a máquina ilustrada na Figura 6 não está completamente modelada, representando apenas a parte superior do rotor e estator. Devido à simetria em

relação ao eixo horizontal, podemos utilizar essa representação para realizar as simulações por meio da definição da periodicidade, que envolve definição do número de polos e ajustes associados à malha de elementos finitos.

Como resultado, a definição da periodicidade permite reduzir o tempo de simulação e a quantidade de memória necessária [6].

2.1.6 Materiais e fontes

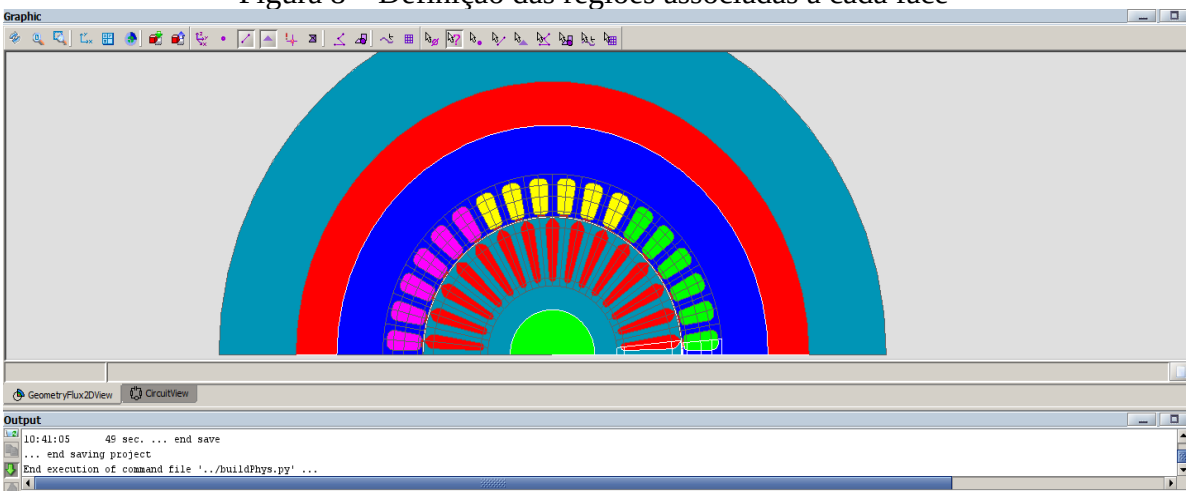
Após representar todas as faces da geometria, o próximo passo é definir as propriedades físicas de cada uma. Isso pode ser feito através da criação de regiões com materiais predefinidos ou determinadas propriedades físicas a cada uma das faces.

O *Flux2D* possui uma biblioteca de materiais com propriedades magnéticas, elétricas e térmicas que podem ser utilizadas para definir os materiais, mas também é possível definir propriedades customizadas.

Do mesmo modo, uma face pode ser associada a uma região que está conectada externamente a um elemento de circuito ou uma fonte de tensão ou corrente.

Como exemplo, a Figura 8 ilustra a máquina após a caracterização das regiões necessárias ao estudo.

Figura 8 – Definição das regiões associadas a cada face



2.1.7 Condições de contorno

As condições de contorno são necessárias para definir o comportamento do campo magnético nas bordas do domínio.

As duas condições utilizadas nas modelagens deste trabalho são: o campo magnético é tangencial na região interna do rotor e nulo na borda externa da região de ar após o estator. Estas condições são bastante comuns neste tipo de estudo.

O próximo capítulo mostra a integração com a linguagem de programação Python e apresenta um exemplo simples de automação.

3 INTEGRAÇÃO ENTRE PYTHON E FLUX

Após o trabalho “mecânico” apresentado no capítulo 2, o presente capítulo descreve a integração do Flux com a linguagem de programação Python por meio do *PyFlux*.

O *PyFlux* da Altair Flux é uma linguagem desenvolvida para os *softwares* de modelagem da empresa, incluindo o *Flux2D*, combinando *Python* com comandos específicos do *PyFlux*. Ela permite a criação de *scripts* para automação e personalização de tarefas dentro do ambiente *Flux2D* [8].

3.1 Automação e reprodutibilidade

A automação de processos é uma das maiores vantagens de se dar ao trabalho de criar um *script* para automatizar a realização de um determinado processo em um projeto, como na mudança dos cenários de simulação e coleta dos resultados. Em uma simulação mais extensa, utilizando técnicas como *grid search*, o tempo total de execução e a quantidade de cenários de simulação podem aumentar exponencialmente, dificultando uma execução precisa, sem erros humanos durante a execução.

A capacidade de automatizar uma tarefa extensa e repetitiva de maneira eficiente e consistente permite que os desenvolvedores foquem em outras tarefas mais importantes ou criem lotes de execução, dividindo a tarefa ou criando ainda mais cenários de simulação, sendo limitados agora apenas pela capacidade computacional das máquinas.

Além disso, a programação de todas as instruções realizadas em um projeto, documentadas e parametrizadas, também garante um aspecto fundamental da pesquisa e desenvolvimento: a reprodutibilidade. A reprodutibilidade permite que outros possam reproduzir o projeto e validar os resultados alcançados.

3.2 Análise de dados

A análise de dados é uma área onde o *PyFlux* apresenta certas limitações devido ao seu interpretador Python, o *Jython* [9]. O *Jython* é uma implementação da linguagem de script Python que é escrita na linguagem Java e integrada com a plataforma Java [10], possuindo suas vantagens em outras áreas, mas possuindo algumas restrições em comparação com a implementação mais utilizada da linguagem de programação *Python*, o *CPython*.

Na prática, isso afeta os módulos que podem ser utilizados no ambiente do *Flux2D*. As principais bibliotecas especializadas na análise de dados como *Numpy* e *Pandas* ou na visualização de dados como *Matplotlib* e *Seaborn* não são compatíveis com o *Jython*, restringindo o que pode ser feito no ambiente do *Flux2D*.

O *PyFlux* ainda oferece diversas visualizações dos resultados das simulações, incluindo algumas visualizações eletromagnéticas e térmicas avançadas, o que pode ser suficiente para muitas aplicações, no entanto, para análises mais completas, pode ser necessário exportar os dados para outro ambiente.

3.3 Comandos básicos

A seguir são descritos alguns comandos básicos do *PyFlux* para a construção de um caso simples de simulação.

3.3.1 Parâmetros geométricos

Para a criação de parâmetros geométricos, utiliza-se o comando mostrado na Tabela 2.

Tabela 1 - Comando para criação de parâmetros geométricos

```
1 ParameterGeom(name='EF : COMPRIMENTO DO ENTREFERRO',
                expression='2')
```

3.3.2 Sistemas de coordenadas

Utiliza-se o comando mostrado na Tabela 3 para a criação de um sistema de coordenadas.

Tabela 2 - Comando para criação de sistema de coordenadas

```
1 CoordSysCartesian(name='ORIGEM',
                    parentCoordSys=Local(coordSys=CoordSys['XY1']),
                    origin=['0', '0'],
                    rotationAngles=RotationAngles(angleZ='0'),
                    visibility=Visibility['VISIBLE'])
```

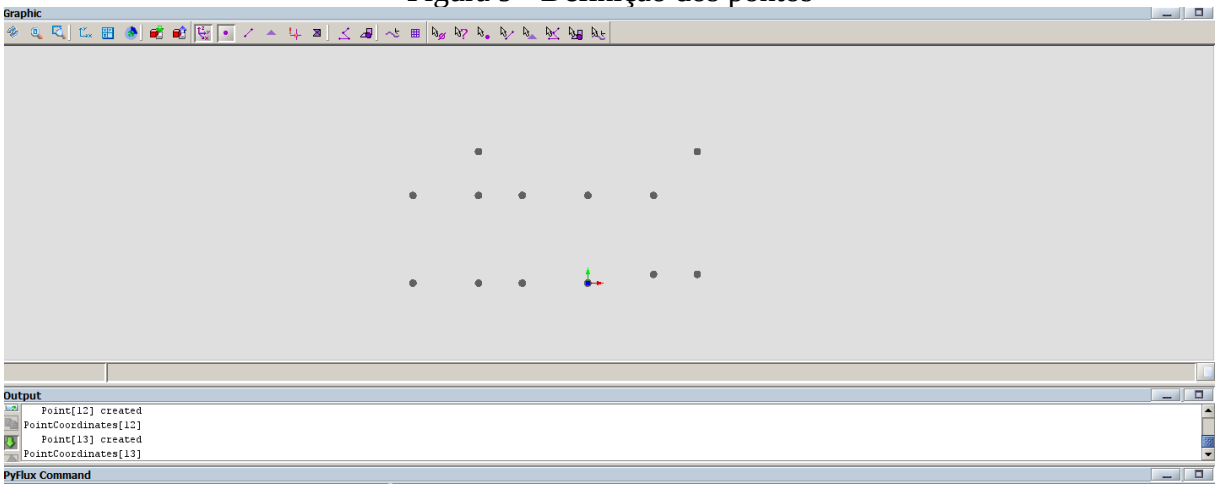
3.3.3 Pontos

Os comandos mostrados na Tabela 3 são usados para definir os pontos ilustrados na Figura 10.

Tabela 3 - Comandos para criação dos pontos

1	`PointCoordinates(color=Color['Grey'], visibility=Visibility['VISIBLE'], coordSys=CoordSys['ORIGEM'], uvw=['-20', '10'], nature=Nature['STANDARD'])`
2	`PointCoordinates(color=Color['Grey'], visibility=Visibility['VISIBLE'], coordSys=CoordSys['ORIGEM'], uvw=['-12.5', '10'])`
3	`PointCoordinates(color=Color['Grey'], visibility=Visibility['VISIBLE'], coordSys=CoordSys['ORIGEM'], uvw=['-7.5', '10'])`
4	`PointCoordinates(color=Color['Grey'], visibility=Visibility['VISIBLE'], coordSys=CoordSys['ORIGEM'], uvw=['0', '10'])`
5	`PointCoordinates(color=Color['Grey'], visibility=Visibility['VISIBLE'], coordSys=CoordSys['ORIGEM'], uvw=['7.5', '10'])`
6	`PointCoordinates(color=Color['Grey'], visibility=Visibility['VISIBLE'], coordSys=CoordSys['ORIGEM'], uvw=['-12.5', '15'])`
7	`PointCoordinates(color=Color['Grey'], visibility=Visibility['VISIBLE'], coordSys=CoordSys['ORIGEM'], uvw=['12.5', '15'])`
8	`PointCoordinates(color=Color['Grey'], visibility=Visibility['VISIBLE'], coordSys=CoordSys['ORIGEM'], uvw=['-20', '0'])`
9	`PointCoordinates(color=Color['Grey'], visibility=Visibility['VISIBLE'], coordSys=CoordSys['ORIGEM'], uvw=['-12.5', '0'])`
10	`PointCoordinates(color=Color['Grey'], visibility=Visibility['VISIBLE'], coordSys=CoordSys['ORIGEM'], uvw=['-7.5', '0'])`
11	`PointCoordinates(color=Color['Grey'], visibility=Visibility['VISIBLE'], coordSys=CoordSys['ORIGEM'], uvw=['0', '0'])`
12	`PointCoordinates(color=Color['Grey'], visibility=Visibility['VISIBLE'], coordSys=CoordSys['ORIGEM'], uvw=['7.5', '0.5*EF'])`
13	`PointCoordinates(color=Color['Grey'], visibility=Visibility['VISIBLE'], coordSys=CoordSys['ORIGEM'], uvw=['12.5', '0.5*EF'])`

Figura 9 - Definição dos pontos



3.3.4 Linhas

Utiliza-se os comandos mostrados na Tabela 4 para definir linhas. As linhas do exemplo estão ilustradas na Figura 10.

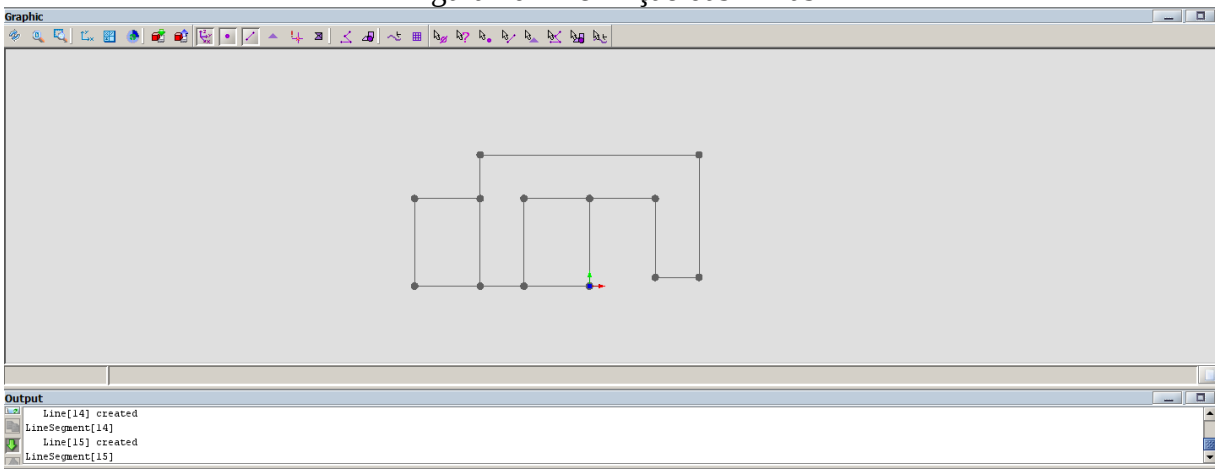
Tabela 4 - Comandos para criação das linhas

1	`LineSegment(color=Color['Grey'], visibility=Visibility['VISIBLE'], defPoint=[Point[8], Point[1]])`
2	`LineSegment(color=Color['Grey'], visibility=Visibility['VISIBLE'], defPoint=[Point[1], Point[2]])`
3	`LineSegment(color=Color['Grey'], visibility=Visibility['VISIBLE'], defPoint=[Point[2], Point[6]])`
4	`LineSegment(color=Color['Grey'], visibility=Visibility['VISIBLE'], defPoint=[Point[6], Point[7]])`
5	`LineSegment(color=Color['Grey'], visibility=Visibility['VISIBLE'], defPoint=[Point[7], Point[13]])`
6	`LineSegment(color=Color['Grey'], visibility=Visibility['VISIBLE'], defPoint=[Point[13], Point[12]])`
7	`LineSegment(color=Color['Grey'], visibility=Visibility['VISIBLE'], defPoint=[Point[12], Point[5]])`
8	`LineSegment(color=Color['Grey'], visibility=Visibility['VISIBLE'], defPoint=[Point[5], Point[4]])`
9	`LineSegment(color=Color['Grey'], visibility=Visibility['VISIBLE'], defPoint=[Point[4], Point[3]])`
10	`LineSegment(color=Color['Grey'], visibility=Visibility['VISIBLE'], defPoint=[Point[3], Point[10]])`
11	`LineSegment(color=Color['Grey'], visibility=Visibility['VISIBLE'],` `LineSegment(color=Color['Grey'], visibility=Visibility['VISIBLE'], defPoint=[Point[2], Point[9]])`

Tabela 5 - Continuação da Tabela 4

12	`LineSegment(color=Color['Grey'], visibility=Visibility['VISIBLE'], defPoint=[Point[8], Point[9]])`
13	`LineSegment(color=Color['Grey'], visibility=Visibility['VISIBLE'], defPoint=[Point[4], Point[11]])`
14	`LineSegment(color=Color['Grey'], visibility=Visibility['VISIBLE'], defPoint=[Point[10], Point[11]])`

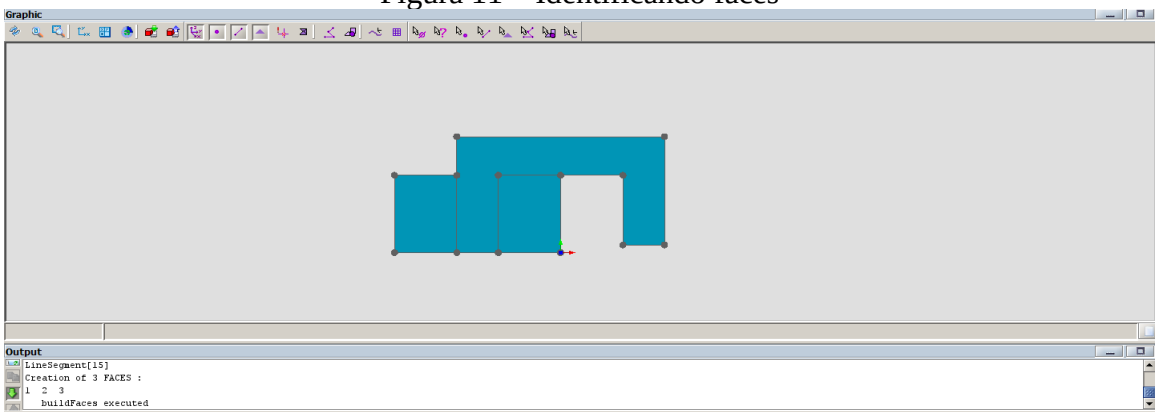
Figura 10 - Definição das linhas



3.3.5 Faces

Utiliza-se o comando `buildFaces()` para que o *Flux2D* identifique as faces da geometria. O resultado é indicado na Figura 11.

Figura 11 – Identificando faces



3.3.6 Transformações

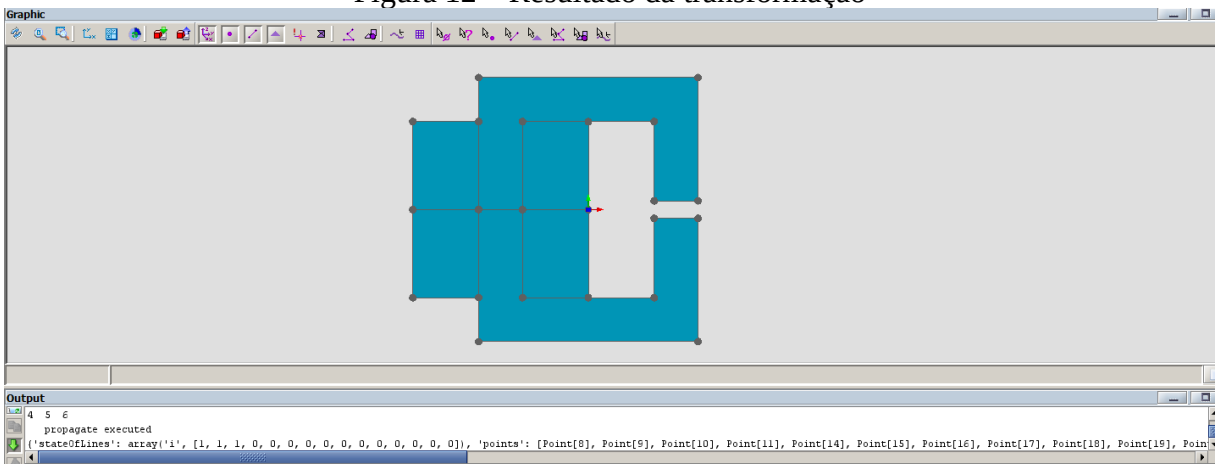
Para criar e utilizar uma transformação, utilizam-se os comandos mostrados na Tabela 6. A Figura 12 ilustra o resultado da transformação.

Tabela 5 - Comandos para criar e utilizar uma transformação

1	`TransfSymmetryLine2PT(name='REFLETE : REFLETE FACES EM RELACAO A UM EIXO',`
	`coordSys=CoordSys['ORIGEM'],`
	`firstPoint=['-20', '0', '0'],`
	`secondPoint=['20', '0', '0'])`

2	`FaceAutomatic[ALL].propagate(transformation=Transf['REFLETE'],`
	`repetitionNumber=1,`
	`buildingOption='FacesWithMeshGenerator',`
	`regionPropagation='None')`

Figura 12 – Resultado da transformação



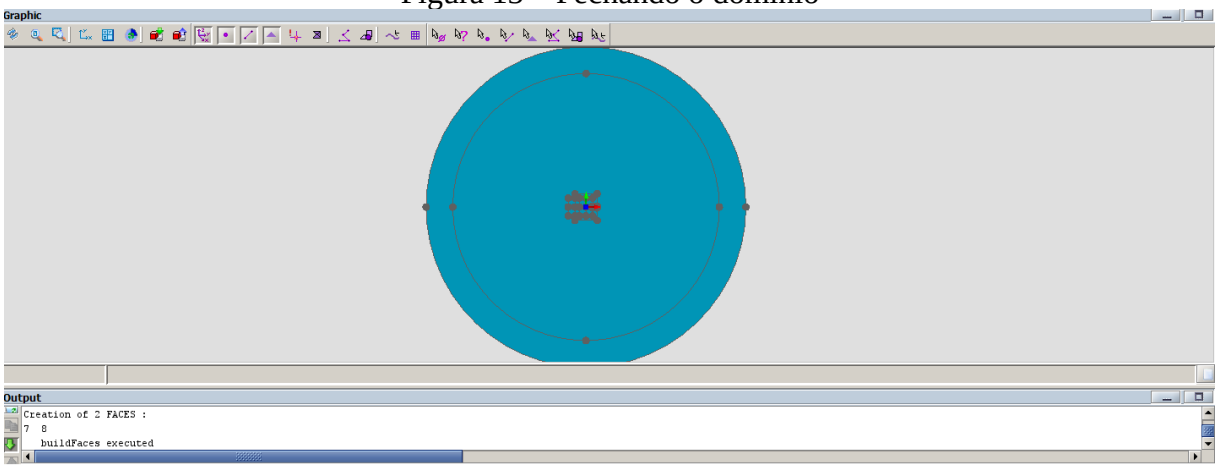
3.3.7 Domínio

Nesse caso, os comandos da Tabela 7 são utilizados para fechar o domínio, simular um domínio infinito e identificar as faces novamente. A Figura 13 ilustra o resultado desses comandos.

Tabela 6 – Fechando o domínio

1	`InfiniteBoxDisc(DISCOID=['150',` `'180'])`
2	`buildFaces()`

Figura 13 – Fechando o domínio



3.3.8 Malha

A tabela 8 mostra os comandos utilizados na criação das definições da malha e como ela é atribuído para alguns elementos específicos. A Figura 14 ilustra a malha criada para esse caso e note que os pontos pretos são um nó da malha, sendo mais concentrados na região do entreferro devido ao interesse em resultados mais precisos na região.

Tabela 7 - Comandos para criação da malha

1	`MeshPoint(name='MeshPoint_1',`
	`lengthUnit=LengthUnit['MILLIMETER'],`
	`value='0.5',`
	`color=Color['Grey'])`

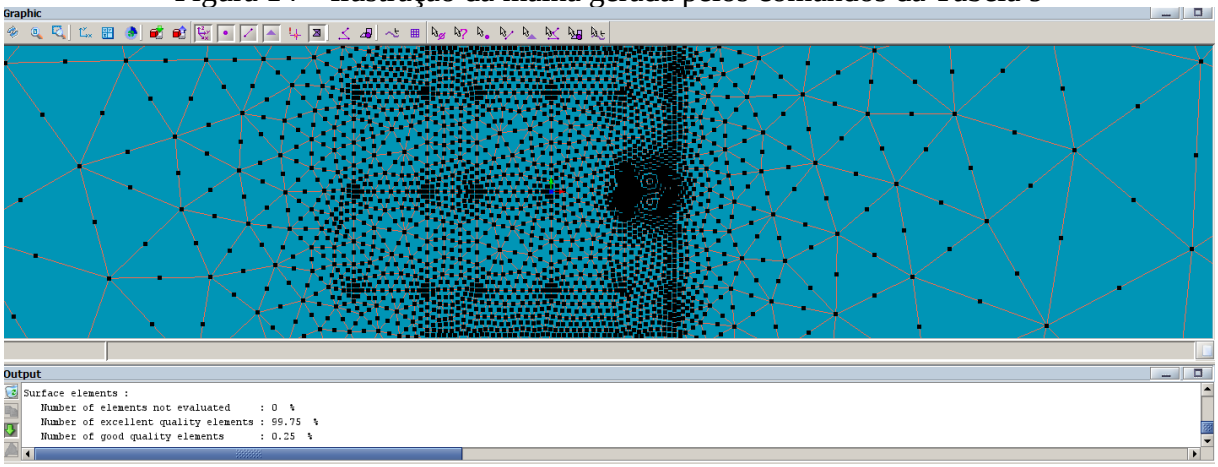
2	`MeshPoint(name='MESHPOINT_2',`
	`lengthUnit=LengthUnit['MILLIMETER'],`
	`value='1',`
	`color=Color['Grey'])`

3	`MeshLineArithmetic(name='MeshLine_1',`
	`color=Color['Grey'],`
	`number=14)`

4	`MeshLineArithmetic(name='MESHLINE_2',`
	`color=Color['Grey'],`
	`number=24)`

5	`PointCoordinates[12,13].assignMeshPoint(meshPoint=MeshPoint['MESHPOINT_1'])`
6	`PointCoordinates[7,5,6,4,3,11,10,9,1,2,8].assignMeshPoint(meshPoint=MeshPoint['MESHPOINT_2'])`
7	`LineSegment[5].assignMeshLine(meshLine=MeshLine['MESHLINE_1'])`
8	`LineSegment[4].assignMeshLine(meshLine=MeshLine['MESHLINE_2'])`
9	`meshDomain()`

Figura 14 – Ilustração da malha gerada pelos comandos da Tabela 9



3.3.9 Materiais

A tabela 9 mostra os comandos utilizados para definir os materiais e as regiões ilustrados na Figura 15.

Tabela 8 - Comandos para criar e atribuir regiões

1	`Material(name='ACO01',`
	`  propertyBH=PropertyBhNonlinearJmu(initialMur='1000',`
	`  js='1.2'))`

2	`RegionFace(name='AR',`
	`  magneticDC2D=MagneticDC2DFaceVacuum(),`
	`  visibility=Visibility['VISIBLE'])`

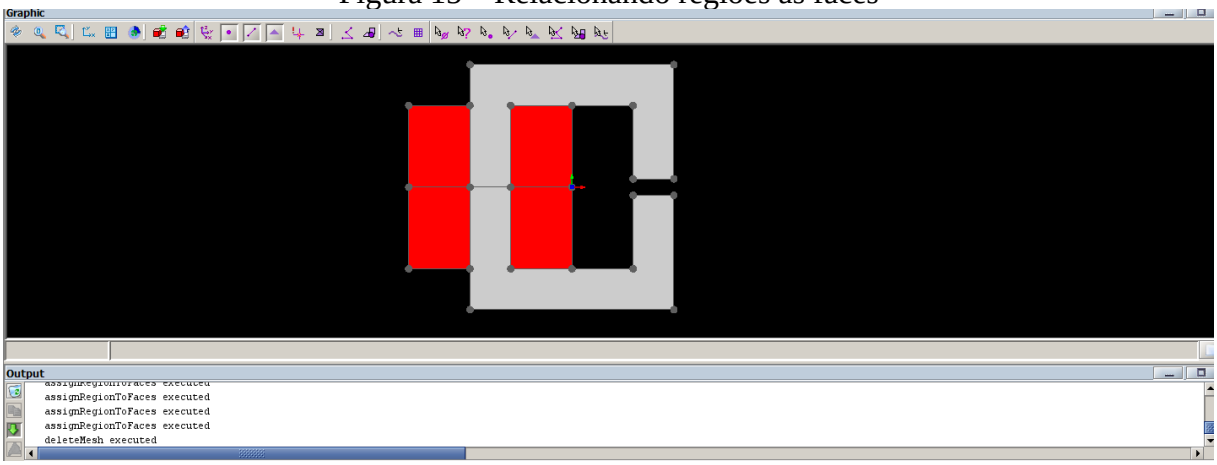
3	`RegionFace(name='BOBINA1',`
	`magneticDC2D=MagneticDC2DFaceFormulaConductor(currentDensity='14.8'),`
	`  visibility=Visibility['VISIBLE'])`

4	`RegionFace(name='BOBINA2',`
	`magneticDC2D=MagneticDC2DFaceFormulaConductor(currentDensity='-14.8'),`
	`  visibility=Visibility['VISIBLE'])`

5	`RegionFace(name='NUCLEO',`
	`  magneticDC2D=MagneticDC2DFaceMagnetic(material=Material['ACO01']),`
	`  visibility=Visibility['VISIBLE'])`

6	`assignRegionToFaces(face=[Face[7]],` `  region=RegionFace['AR'])`
7	`assignRegionToFaces(face=[Face[1], Face[4]],` `  region=RegionFace['BOBINA1'])`
8	`assignRegionToFaces(face=[Face[3], Face[6]],` `  region=RegionFace['BOBINA2'])`
9	`assignRegionToFaces(face=[Face[2], Face[5]],` `  region=RegionFace['NUCLEO'])`

Figura 15 – Relacionando regiões às faces



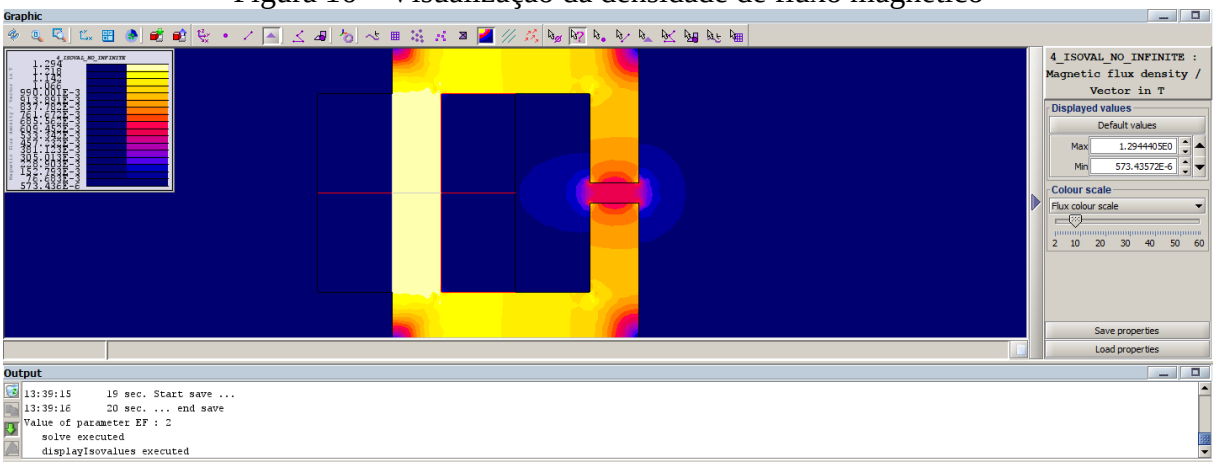
3.3.10 Simulação

A tabela 10 mostra os comandos utilizados para criar e solucionar um cenário de simulação simples. A Figura 16 ilustra a densidade de fluxo magnético simulada.

Tabela 9 - Comandos para criação e resolução de cenário de simulação

1	`Scenario(name='Scenario_1',` `adaptive=InactivatedAdaptive())`
2	`Scenario['SCENARIO_1'].solve(projectName='CIRCUITO_SIMPLES.FLU')`
3	`displayIsovalues()`

Figura 16 – Visualização da densidade de fluxo magnético



3.3.11 Aplicação

Tendo em vista os conceitos apresentados até aqui, desenvolvemos no próximo capítulo uma aplicação automatizada para construção de geometria de uma máquina síncrona real. O

objetivo é ter a automação da construção e permitir o estudo da influência dos parâmetros utilizados no desempenho da máquina.

4 PROCEDIMENTOS

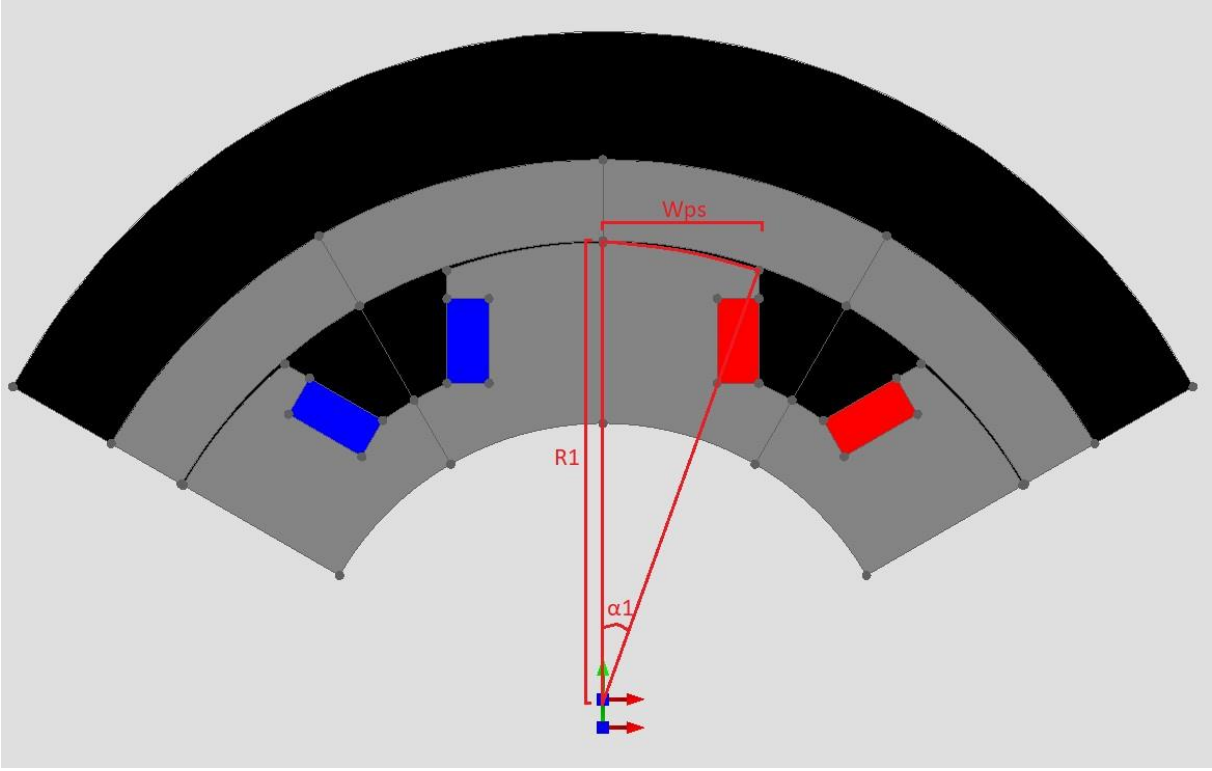
Utilizando os conceitos básicos da automação do capítulo 3, o presente capítulo desenvolve o estudo de uma máquina síncrona específica a partir de parâmetros previamente determinados.

É realizada a parametrização da geometria e realizado o código *PyFlux* com o objetivo de automatizar a simulação e obtenção de resultados.

4.1 Modelagem geométrica

Para a modelagem da geometria, foi utilizada de referência a máquina síncrona Camlica, ilustrada no documento da Alstom [11]. A máquina possui uma geometria de um arco nos polos do rotor e foi simplificada em relação à original para facilitar o processo de modelagem. A Figura 17 ilustra a modelagem geométrica e destaca o arco do rotor. O código do *script* criado ao longo do trabalho para reproduzir essa modelagem encontra-se no Apêndice A.

Figura 17 - Máquina síncrona com geometria de rotor de um arco



4.2 Cenários de simulação

A partir da geometria modelada, os cenários de simulação buscam calcular a densidade de fluxo magnético ao longo do entreferro da máquina, a partir da linha média entre o rotor e o estator, enquanto os parâmetros geométricos da máquina são alterados.

Os parâmetros utilizados nesse trabalho que definem o arco mostrado na Figura 17 são: (1) o raio externo do rotor R_1 (mm), e (2) a largura da sapata polar W_{ps} (mm). O ângulo α_1 (rad ou graus) pode ser obtido como função dos dois parâmetros anteriores, de acordo com a expressão (1).

$$\alpha_1 = 2 \cdot \text{sen}^{-1}\left[\frac{W_P}{2R_1}\right] \tag{1}$$

Cada combinação desses parâmetros cria um cenário de simulação distinto, permitindo uma análise de como a variação de cada parâmetro geométrico impacta no fator a ser analisado.

As Figuras 18 e 19 ilustram o efeito da variação desses parâmetros na geometria. Observe na Figura 18 que mesmo com a variação do raio externo do rotor, a geometria foi parametrizada de forma que o entreferro mínimo é constante enquanto na Figura 19, a variação da sapata polar afeta a largura da região onde se encontram as bobinas.

Figura 18 - Variação do raio externo do rotor

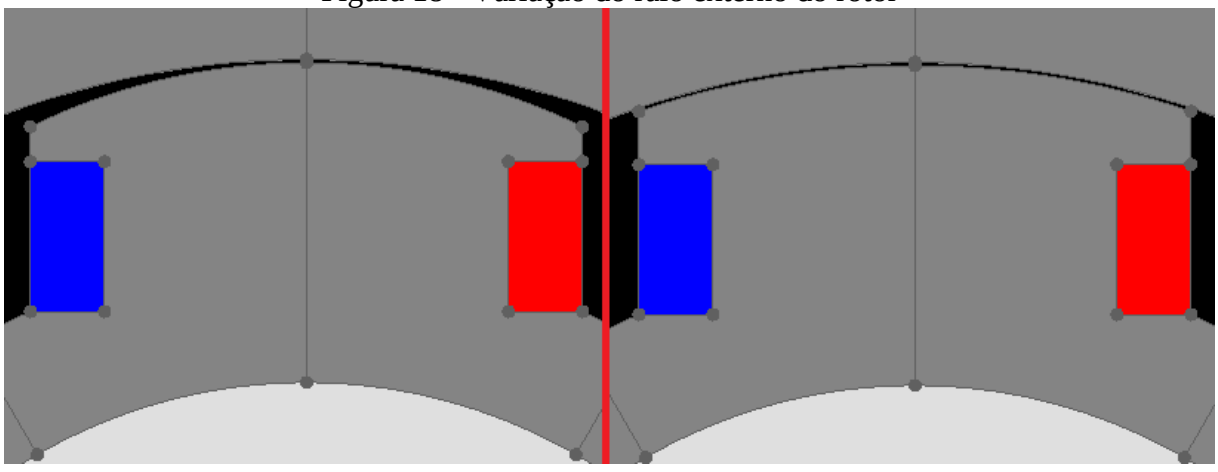
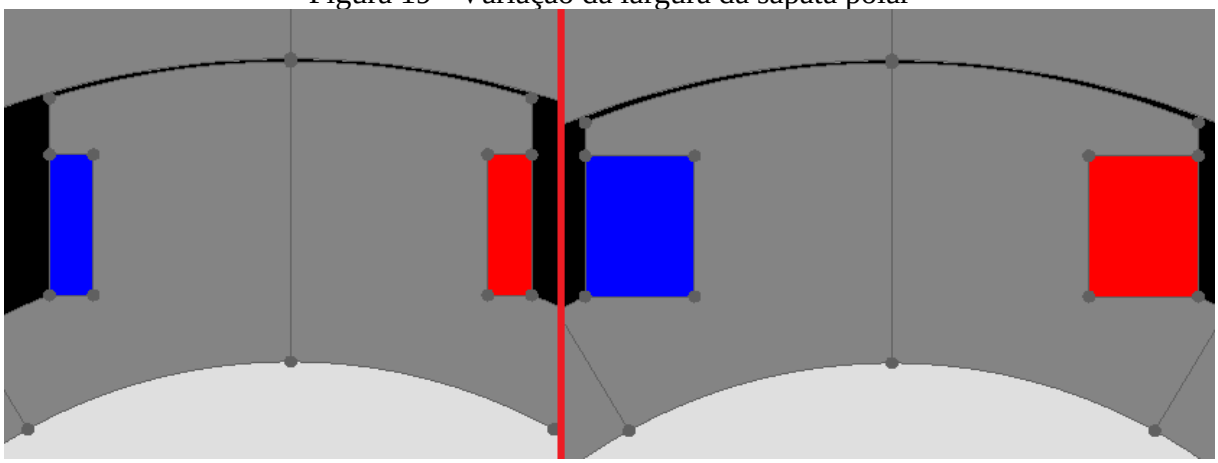


Figura 19 - Variação da largura da sapata polar



4.3 Resultados a partir da variação de parâmetros

A partir dos cenários de simulação, apresentam-se a seguir alguns resultados da simulação da máquina síncrona, a título ilustrativo.

Para verificação da influência da variação dos parâmetros conforme definidos no item 4.2, realiza-se o cálculo da densidade de fluxo magnético na linha média do entreferro para diferentes parâmetros, sempre de forma automatizada.

Definem-se, para cada caso, os valores do raio externo do rotor e do ângulo da cabeça polar, conforme ilustrado na Tabela 11.

Tabela 10 - Parâmetros para estudo da máquina síncrona

Caso	R_1 (mm)	α_1 (graus)
1	950	38,1
2	950	34,9
3	850	42,1
4	850	39,9

Ressalta-se que todos os resultados apresentados são obtidos de forma automatizada através das instruções *PyFlux*, de acordo com os comandos descritos no Apêndice A.

4.3.1 Caso 1.

O mapa de densidade de fluxo magnético e a configuração de fluxo magnético para o presente caso são mostrados, respectivamente, nas Figuras 20 e 21.

Figura 20 – Mapa de densidade de fluxo magnético – caso 1

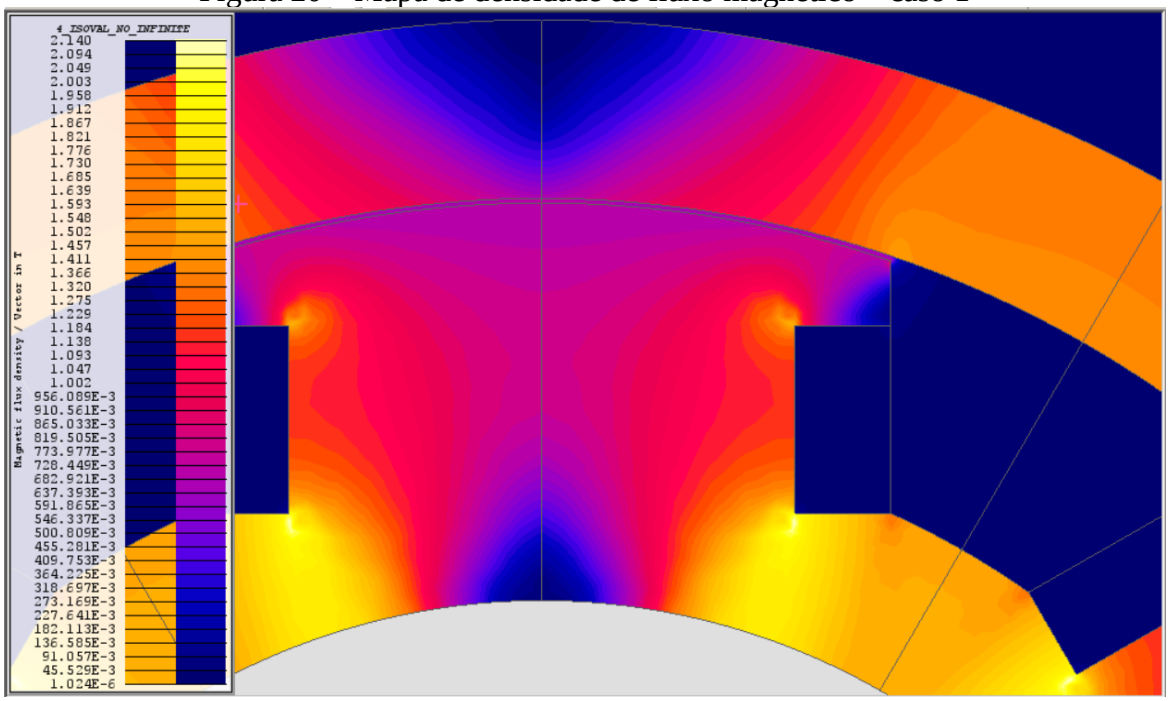
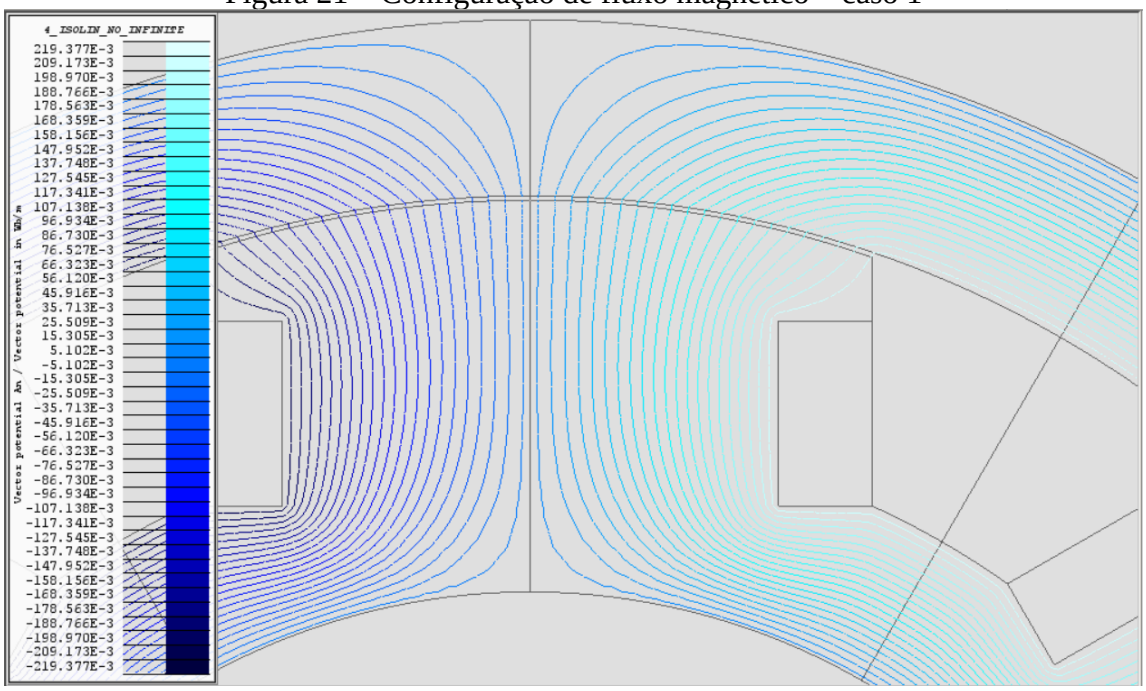


Figura 21 – Configuração de fluxo magnético – caso 1



4.3.2 Caso 2.

Para o presente caso, o mapa de densidade de fluxo magnético e a configuração de fluxo magnético são mostrados, respectivamente, nas Figuras 22 e 23.

Figura 22 – Mapa de densidade de fluxo magnético – caso 2

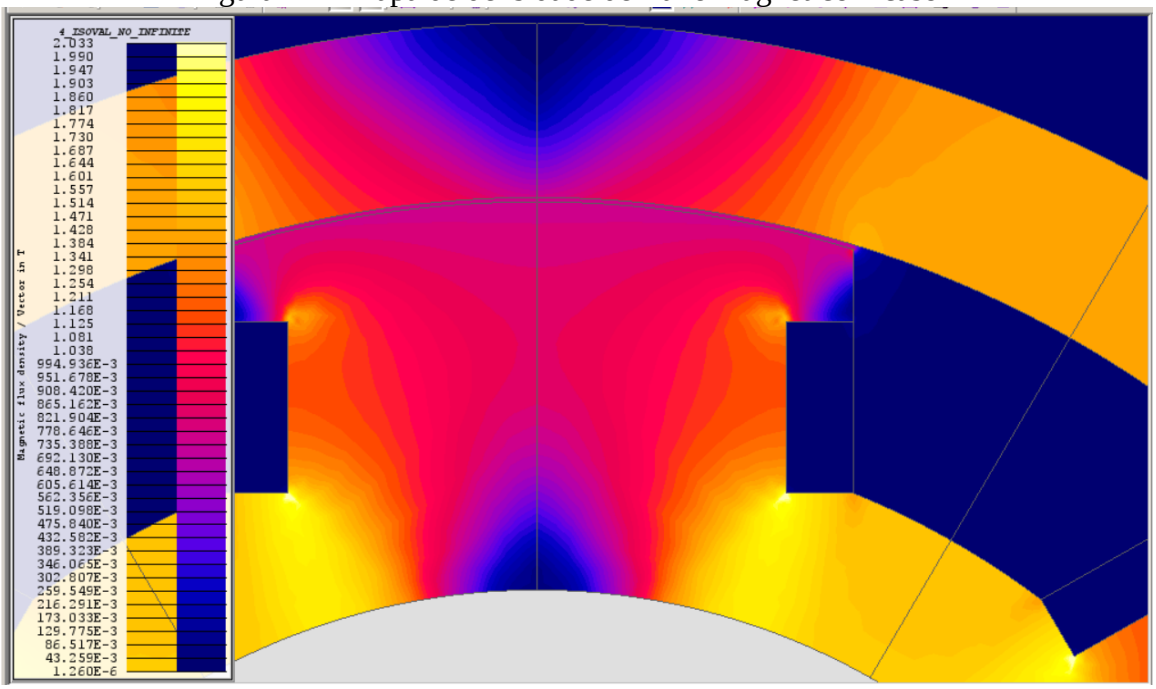
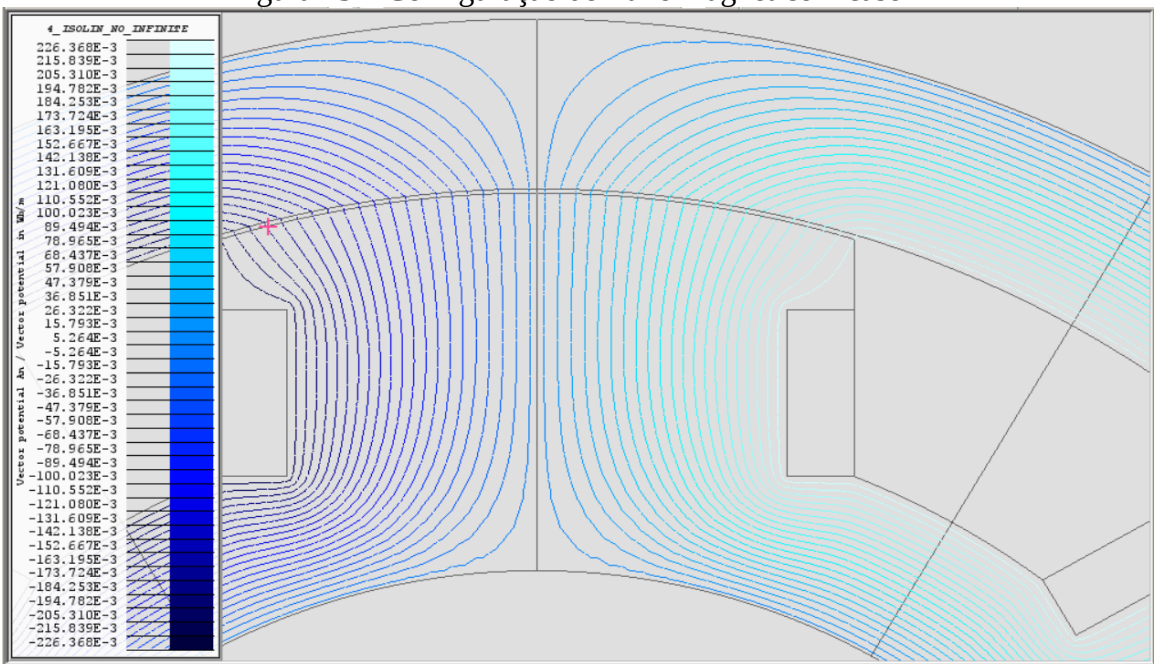


Figura 23 – Configuração de fluxo magnético – caso 2



4.3.3 Caso 3.

Neste caso, o mapa de densidade de fluxo magnético e a configuração de fluxo magnético são mostrados, respectivamente, nas Figuras 24 e 25.

Figura 24 – Mapa de densidade de fluxo magnético – caso 3

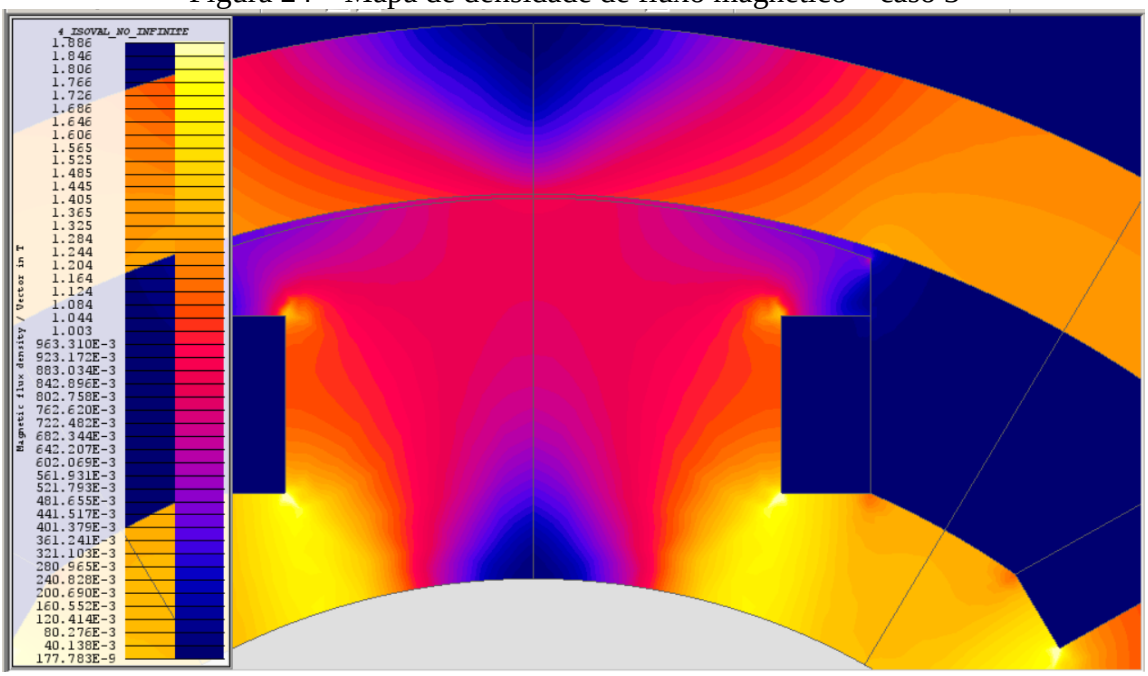
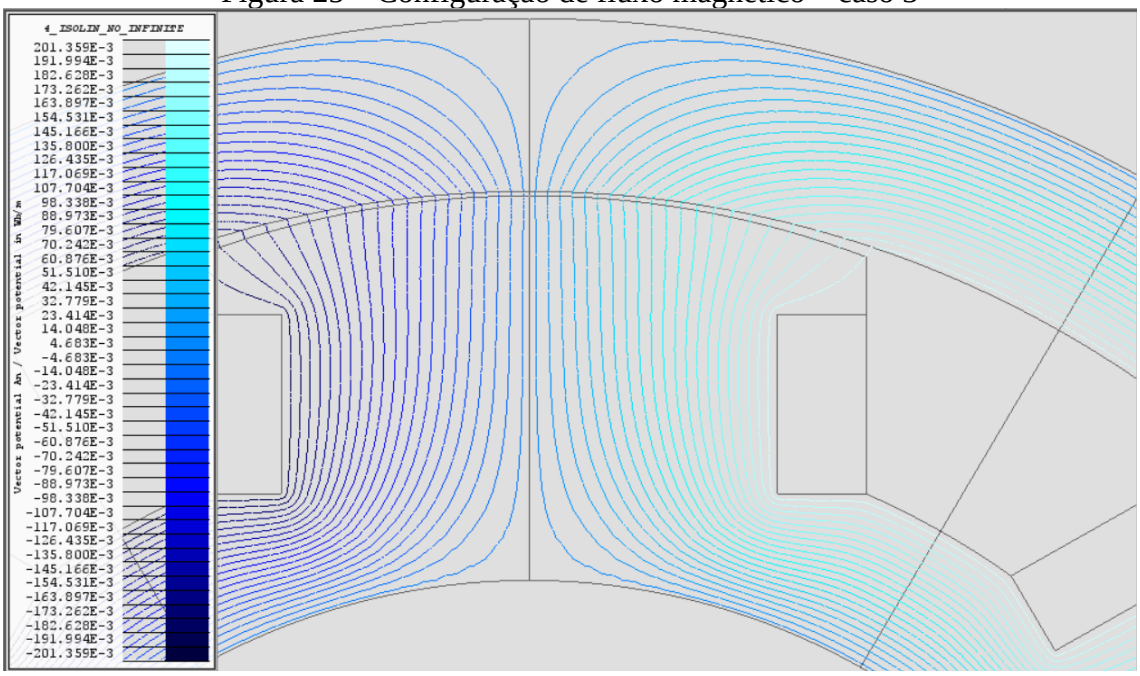


Figura 25 – Configuração de fluxo magnético – caso 3



4.3.4 Caso 4.

O mapa de densidade de fluxo magnético e a configuração de fluxo magnético para o presente caso são mostrados, respectivamente, nas Figuras 26 e 27.

Figura 26 – Mapa de densidade de fluxo magnético – caso 4

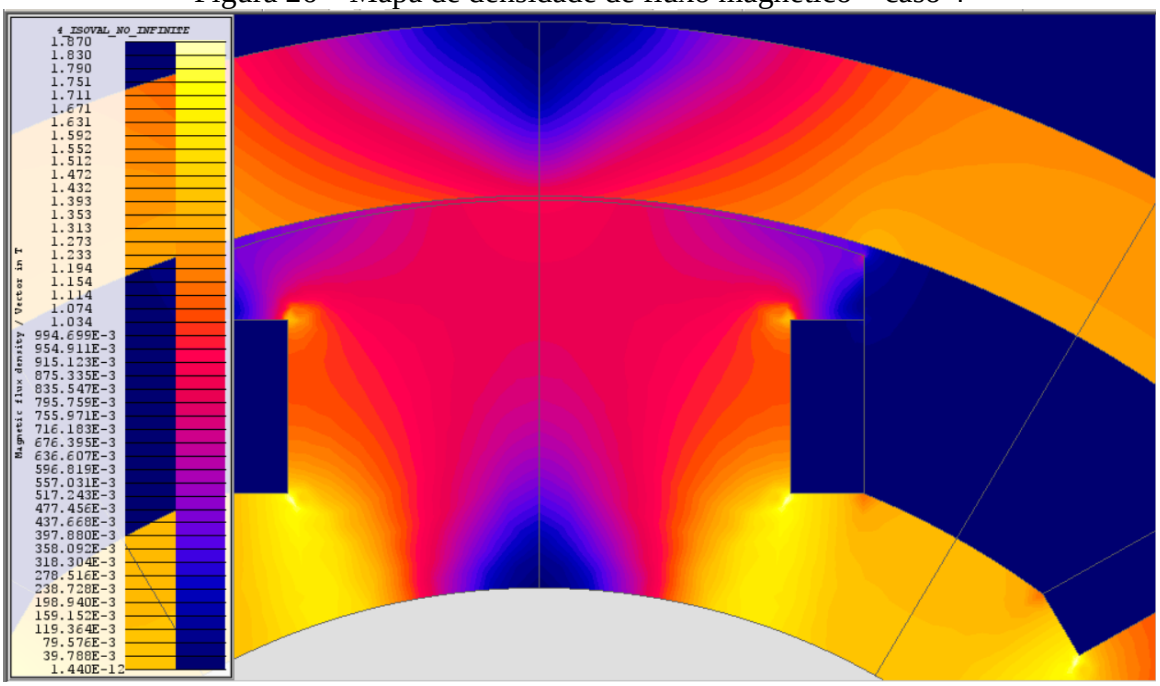
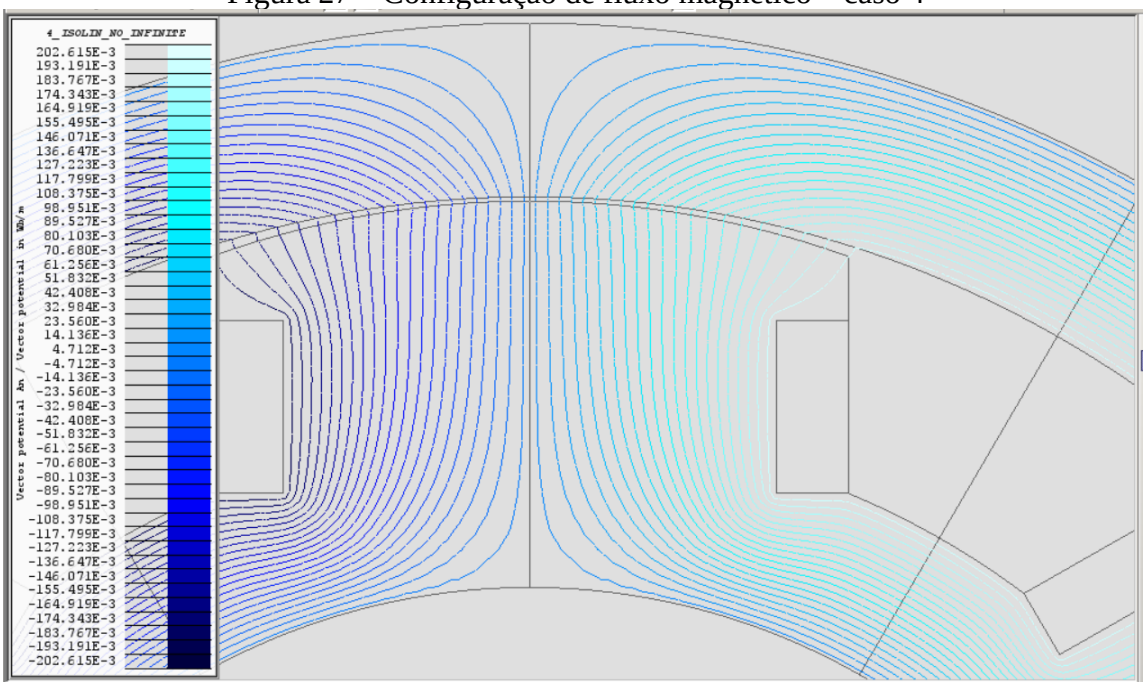


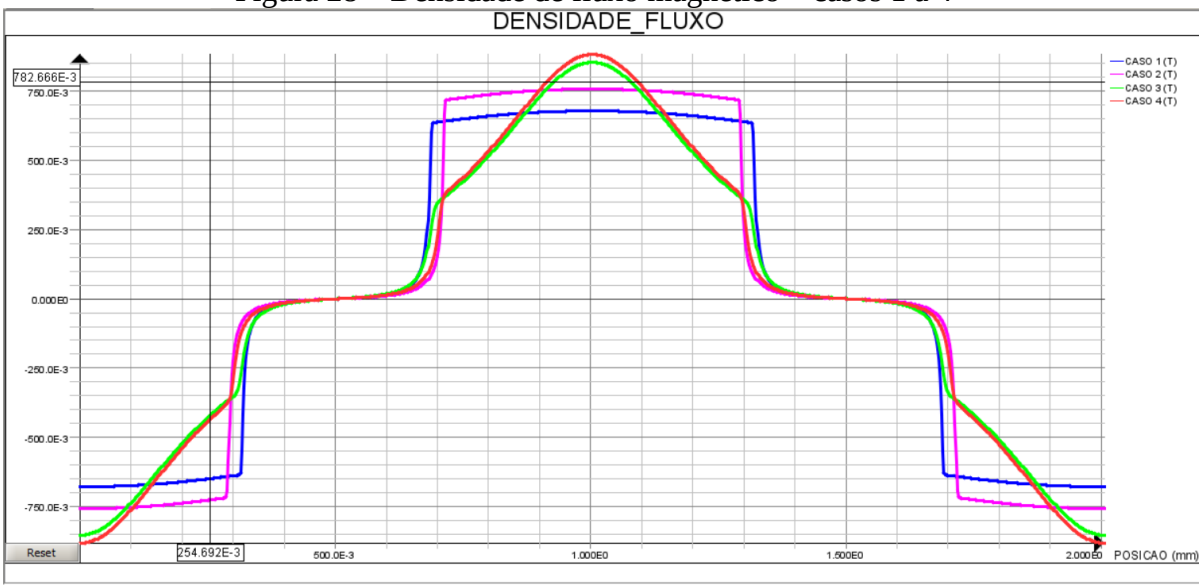
Figura 27 – Configuração de fluxo magnético – caso 4



4.3.5 Densidade de fluxo magnético de entreferro

Os gráficos da densidade de fluxo magnético de entreferro para cada caso, conforme os valores indicados na Tabela 11 são mostrados na Figura 28. É possível observar a forte influência dos parâmetros na forma de onda da densidade de fluxo.

Figura 28 – Densidade de fluxo magnético – casos 1 a 4



A Tabela 13 mostra os dados característicos das ondas da Figura 28. Estes dados são o fluxo magnético de pico e a densidade de fluxo média sob um polo.

Tabela 11 - Fluxo e densidade de fluxo para os gráficos da Figura 28

Caso	Fluxo magnético de pico (Wb)	Densidade de fluxo média (T)
1	0,219	0,434
2	0,226	0,448
3	0,201	0,397
4	0,203	0,400

Assim, demonstra-se a grande capacidade de obtenção de resultados do Flux2D quando se utiliza o recurso de programação com a linguagem *PyFlux*.

O próximo capítulo propõe utilizar a forma automatizada de obter os resultados para determinar a melhor geometria a ser escolhida para a presente máquina síncrona em estudo.

5 ESTUDO DE MELHORIA DA GEOMETRIA

No presente capítulo se propõe a verificação comparativa e qualitativa de diferentes geometrias da máquina em estudo, com vistas a escolher uma geometria que apresenta um desempenho mais adequado, segundo o critério da distorção harmônica.

5.1 A distorção harmônica

Para a análise dos resultados, o parâmetro de qualidade escolhido foi o fator de distorção harmônica, obtido pela análise do espectro de Fourier da curva de densidade de fluxo magnético no entreferro gerada para cada cenário de simulação.

Para além dos casos estudados no capítulo 4, foram realizadas simulações com a seguinte parametrização:

- Variação do raio da cabeça polar entre 800 e 955 mm, com passo de variação de 5 mm, num total de 32 valores.
- Variação da largura da cabeça polar entre 600 e 770 mm, com passo de variação de 5 mm, num total de 35 valores.

A parametrização assumida resulta num total de 1120 casos de simulação (32 valores do raio da cabeça polar vezes 35 valores da largura da cabeça polar).

O *Flux2D* permite a determinação do espectro de Fourier da densidade de fluxo magnético de entreferro, de acordo com o especificado no Apêndice A. Assim, é possível determinar individualmente cada componente harmônico espacial da curva espacial em questão.

Como exemplo, para o Caso 1 do capítulo 4, apresenta-se a densidade de fluxo magnético de entreferro na Figura 30. O espectro harmônico é mostrado na Tabela 13.

Figura 29 – Densidade de fluxo magnético – caso 1

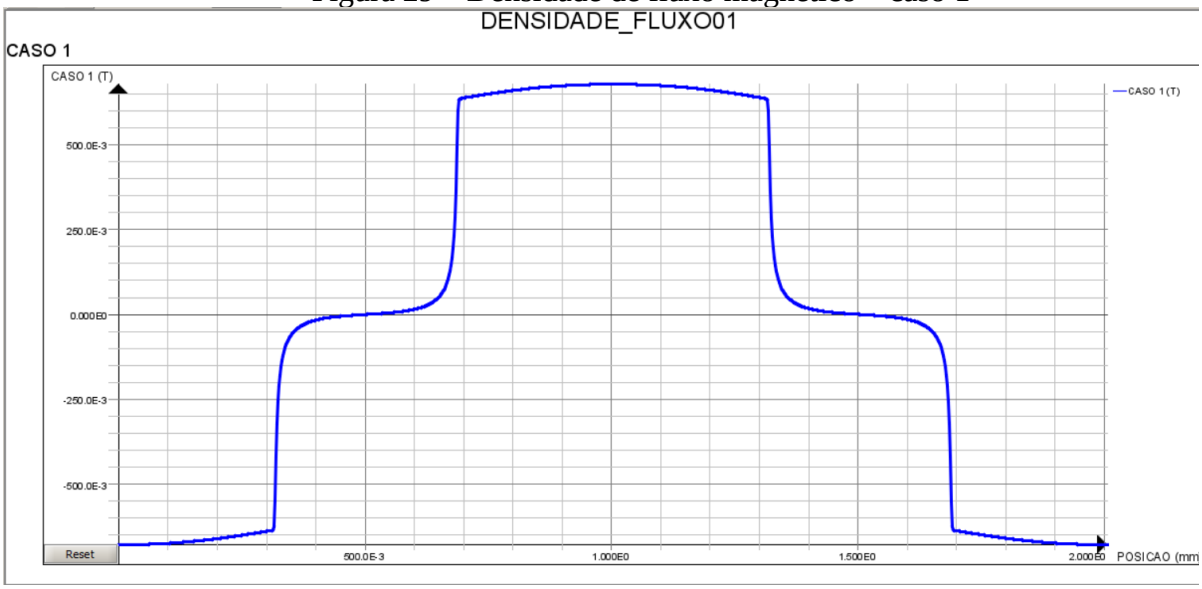


Tabela 12 - Valores dos harmônicos para o espectro da curva da Figura 29

Ordem harmônica	Densidade de fluxo (T)
1	0,7234
2	0,0000
3	0,0290
4	0,0000
5	0,1423
6	0,0000
7	0,0783
8	0,0000
9	0,0223
10	0,0000
11	0,0628

Assim, para avaliar a "qualidade" da onda de densidade de fluxo magnético obtida, propõe-se usar a expressão (2) a seguir para o cálculo da distorção harmônica $DH_{\%}$, na qual N é o número de harmônicos escolhido [12].

$$DH_{\%} = \left(1 - \frac{1}{\mathrm{B}_1} \cdot \sqrt{\sum_{n=1}^{N} B_n^2}\right) \times 100 \qquad (2)$$

Como exemplo, utilizando os dados da Tabela 13, determinamos $DH = 2{,}98\%$, para o caso 1, considerando os 11 primeiros harmônicos.

5.2 Resultados

De acordo com o procedimento do item 5.1, utilizando as variações de raio e largura da cabeça polar conforme estabelecido, obtemos o seguinte resultado para o raio e o ângulo da cabeça polar. O valor da distorção harmônica obtida é o menor valor obtido entre todas as simulações.

- Raio da cabeça polar: 900 mm.
- Largura da cabeça polar: 745 mm.
- Ângulo da cabeça polar: 48,9 graus.
- Distorção harmônica: 0,37%

As Figuras 30 e 31 mostram, respectivamente, o mapa da densidade de fluxo magnético e a configuração de fluxo magnético para o caso obtido com a melhor distorção harmônica

Figura 30 – Mapa de densidade de fluxo magnético – melhor caso

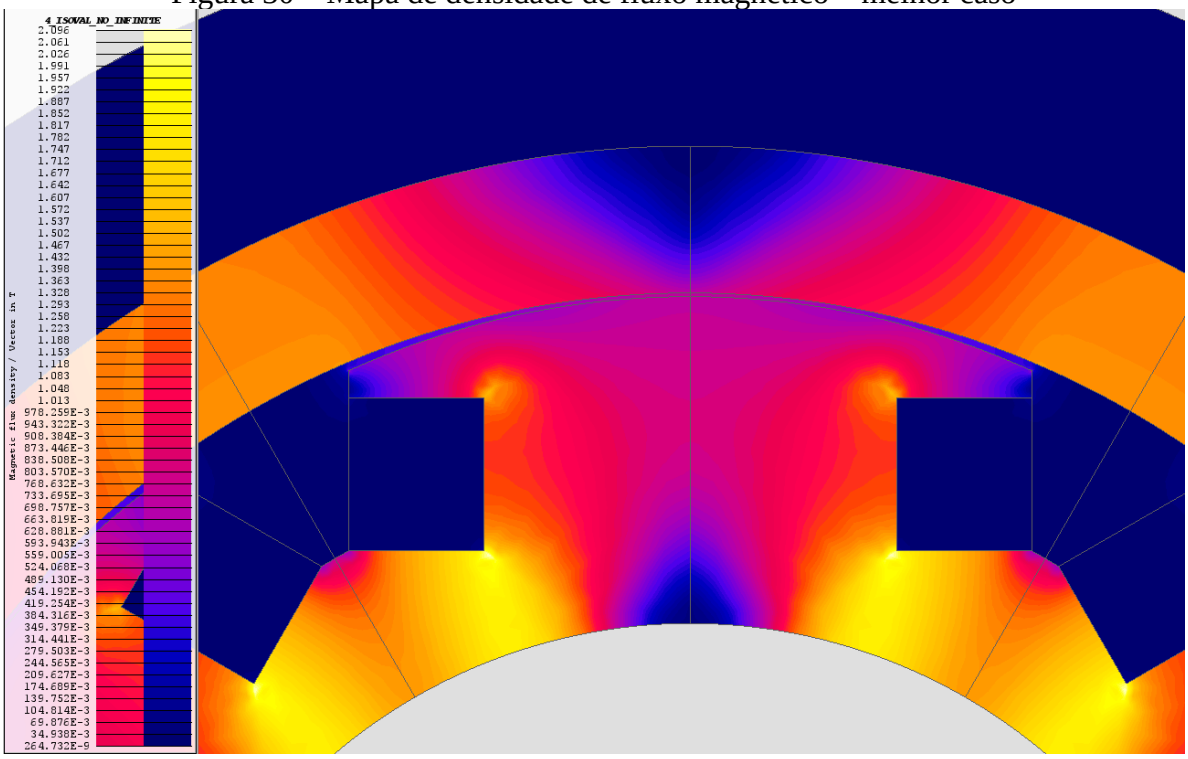
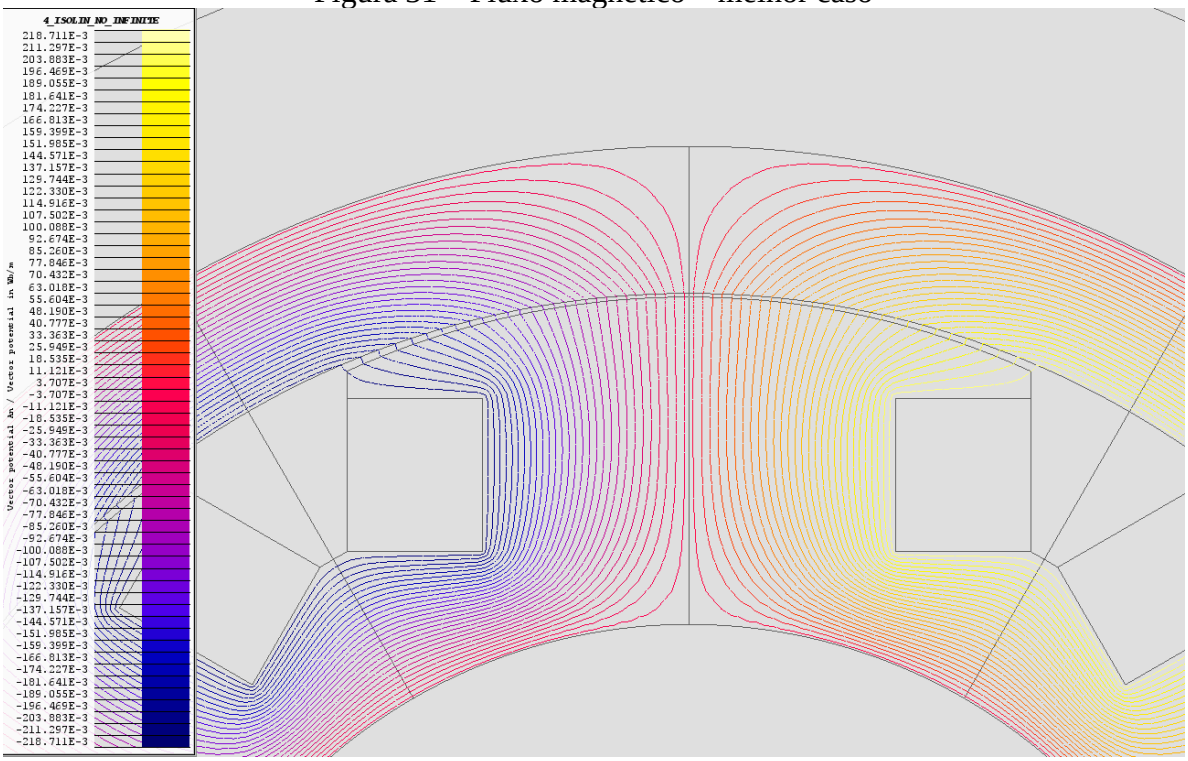


Figura 31 – Fluxo magnético – melhor caso



Finalmente, as Figuras 32 e 33 mostram, respectivamente, a densidade de fluxo magnético calculada ao longo da linha média do entreferro para dois passos polares e o espectro harmônico da mesma onda.

Figura 32 – Densidade de fluxo magnético de entreferro – melhor caso

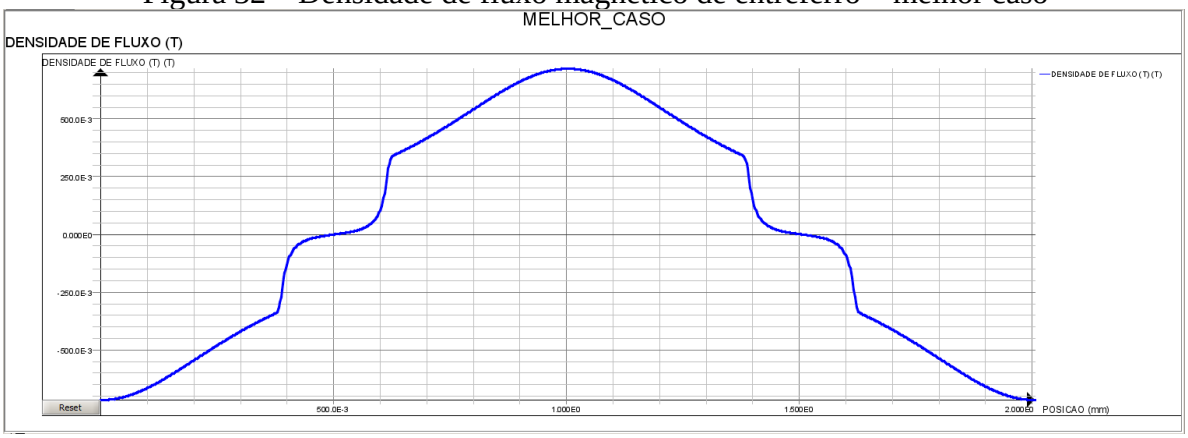
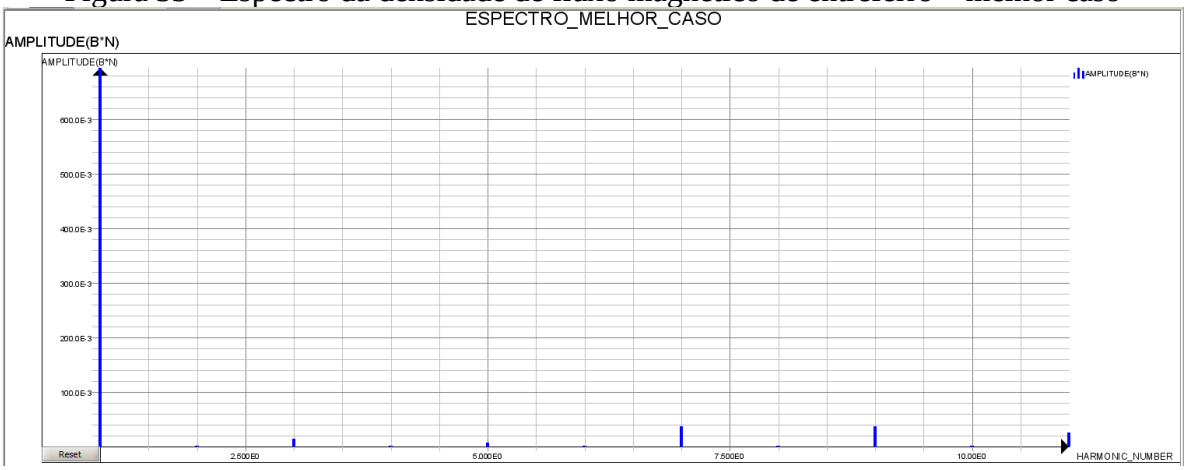


Figura 33 – Espectro da densidade de fluxo magnético de entreferro – melhor caso



6 CONCLUSÃO

Este trabalho teve como foco a modelagem e simulação das formas polares das máquinas síncronas utilizando o *software Flux2D* e a linguagem de programação *Python*. Os objetivos principais foram construir a geometria dessas máquinas, determinar a densidade de fluxo magnético de entreferro e analisar os efeitos da variação de parâmetros geométricos de forma automática.

O expressivo aumento do número de cenários de simulação para a avaliação de qualidade e geração de cada resultado mostra como a integração de *Python* com *Flux2D* auxilia a realização de uma análise mais precisa, mas observa-se limitações do uso da ferramenta: (I) tempo de simulação, que pode crescer consideravelmente com a quantidade de variáveis no processo de busca estabelecido; e (II) necessidade física de memória.

Como continuidade, sugere-se desenvolver os códigos Python para geometrias de dois e três arcos e aplicar processos de otimização.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] AGÊNCIA NACIONAL DE ENERGIA ELÉTRICA (ANEEL). Matriz elétrica brasileira alcança 200 GW. Disponível em: https://www.gov.br/aneel/pt-br/assuntos/noticias/2024/matriz-eletrica-brasileira-alcanca-200-gw.

[2] ALTAIR. Flux. Disponível em: https://altair.com/flux.

[3] ALTAIR. Importation of CAD files. Disponível em: https://2022.help.altair.com/2022/flux/Flux/Help/english/UserGuide/English/topics/ImportDeFichiersCAO.htm.

[4] ALTAIR. Import formats. Disponível em: https://help.altair.com/flux/flux/help/english/UserGuide/English/topics/PrincipeDeConversionEtOptionsDeConversion.htm.

[5] ALTAIR. 2D Examples. Disponível em: https://help.altair.com/flux/flux/help/english/UserGuide/English/topics/Examples2D_en.htm.

[6] ALTAIR. Reduction of the study domain: symmetries and periodicities. Disponível em: https://help.altair.com/flux/Flux/Help/english/UserGuide/English/topics/ReductionDuDomaineDetudeSymetriesEtPeriodicites.htm.

[7] ALTAIR. Shape Optimization. Disponível em: https://2021.help.altair.com/2021.2/flux/Flux/Help/english/UserGuide/English/topics/ShapeOptimization.htm.

[8] ALTAIR. PyFlux and Python languages. Disponível em: https://help.altair.com/flux/Flux/Help/english/UserGuide/English/topics/LangagesPyFluxEtPython.htm.

[9] ALTAIR. How to experiment with NumPy in Flux. Disponível em: https://2022.help.altair.com/2022/flux/Flux/Help/english/HowTo/English/HowTo_ExperimentWithNumPyInFlux/topics/experimentwithnumpyinflux_r.htm.

[10] IBM. Watson Studio: Language - Python e Jython. Disponível em: https://www.ibm.com/docs/pt-br/watsonx/saas?topic=language-python-jython.

[11] ALSTOM. Electrical Pole Geometry. Disponível em: https://drive.google.com/file/d/1R5DHcrt-QOs78Ss2oUseDGtzlLOcCWwZ/view?usp=sharing.

[12] KREYSZIG, Erwin. Advanced Engineering Mathematics. 9. ed. Hoboken: John Wiley & Sons, 2006.

GLOSSÁRIO

Software: Conjunto de instruções escritas em uma linguagem de programação.

Flux2D: Software de simulação de campo eletromagnético da Altair.

Python: Linguagem de programação de alto nível.

Log: Registro de um evento relevante no sistema.

Grid Search: Técnica de machine learning para encontrar a melhor combinação de parâmetros pelo teste de todas as combinações possíveis de parâmetros especificados pelo usuário.

Script: Similar a software. Programa escrito em uma linguagem de script específica.

Jython: implementação da linguagem de script Python que é escrita na linguagem Java e integrada com a plataforma Java

CPython: implementação da linguagem Python em C.

PyFlux: Implementação do Python utilizando o Jython dentro do ambiente do Flux2D.

Numpy: biblioteca Python especializada em cálculos numéricos.

Pandas: biblioteca Python especializada na manipulação de dados.

Matplotlib: biblioteca Python de visualização gráfica.

Seaborn: biblioteca Python construída sobre o Matplotlib para fornecer análises estatísticas.

APÊNDICE A – Script PyFlux para modelagem geométrica da máquina de um arco

```
#! Flux2D 23.0

import math

#-------------------------------------------------------------------------------
# 1. Definindo parâmetros geométricos dentro no Flux
#-------------------------------------------------------------------------------

ParameterGeom(name='P : numero de pares de polos',
              expression='3.0')
ParameterGeom(name='WPS : largura da sapata',
                expression='615.0')
ParameterGeom(name='WP : largura do nucleo polar',
                expression='450.0')
ParameterGeom(name='HPS : altura da sapata',
                expression='110.0')
ParameterGeom(name='HR : altura do nucleo polar',
                expression='167.0')
ParameterGeom(name='DELTA_0 : valor minimo do entreferro',
                expression='4.0')
ParameterGeom(name='R0 : raio interno da coroa do rotor',
                expression='600.0')
ParameterGeom(name='R1 : raio da sapata polar no entreferro',
                expression='900.0')
ParameterGeom(name='R4 : raio interno do estator',
                expression='960.0')
ParameterGeom(name='R5 : raio externo do estator',
                expression='1120.0')
ParameterGeom(name='R6 : raio externo da coroa do rotor',
                expression='Sqrt((WPS/2)^2+(R4-DELTA_0-HPS-HR)^2)')
ParameterGeom(name='Y1',
              expression='R4-DELTA_0-R1')
ParameterGeom(name='ALPHA_P',
                expression='360/(2*P)')
ParameterGeom(name='ALPHA_1',
                expression='2*Asind(WPS/(2*R1))')
ParameterGeom(name='ALPHA_P2',
                expression='Asind(WPS/(2*R6))')

#-------------------------------------------------------------------------------
# 2. Definindo sistemas de coordenadas
#-------------------------------------------------------------------------------

CoordSysCartesian(name='ROTOR',
                  parentCoordSys=GlobalUnits(lengthUnit=LengthUnit['MILLIMETER'],
angleUnit=AngleUnit['DEGREE']),
                  origin=['0', '0'],
                  rotationAngles=RotationAngles(angleZ='0'),
                  visibility=Visibility['VISIBLE'])
CoordSysCartesian(name='ROTOR_CABECA_POLAR',
                  parentCoordSys=Local(coordSys=CoordSys['ROTOR']),
                  origin=['0', 'Y1'],
                  rotationAngles=RotationAngles(angleZ='0'),
                  visibility=Visibility['VISIBLE'])
CoordSysCartesian(name='ESTATOR',
                  parentCoordSys=GlobalUnits(lengthUnit=LengthUnit['MILLIMETER'],
angleUnit=AngleUnit['DEGREE']),
                  origin=['0', '0'],
                  rotationAngles=RotationAngles(angleZ='0'),
                  visibility=Visibility['VISIBLE'])

#-------------------------------------------------------------------------------
# 3. Definindo opções de linhas para a malha
#-------------------------------------------------------------------------------
```

```
MeshLineArithmetic(name='M_01',
                   color=Color['Grey'],
                   number=280)
MeshLineArithmetic(name='M_02',
                   color=Color['Grey'],
                   number=200)
MeshLineArithmetic(name='M_03',
                   color=Color['Grey'],
                   number=3)
MeshLineGeometricNumber(name='M_04',
                        color=Color['Grey'],
                        number=30,
                        concentrationFactor='1.12')
MeshLineGeometricNumber(name='M_05',
                        color=Color['Grey'],
                        number=13,
                        concentrationFactor='1.15')
MeshLineGeometricNumber(name='M_06',
                        color=Color['Grey'],
                        number=12,
                        concentrationFactor='1.3')
MeshLineGeometricNumber(name='M_07',
                        color=Color['Grey'],
                        number=20,
                        concentrationFactor='1.16')
MeshLineArithmetic(name='M_08',
                   color=Color['Grey'],
                   number=12)
MeshLineGeometricNumber(name='M_09',
                        color=Color['Grey'],
                        number=3,
                        concentrationFactor='1.2')
MeshLineArithmetic(name='M_10',
                   color=Color['Grey'],
                   number=20)
MeshLineArithmetic(name='M_11',
                   color=Color['Grey'],
                   number=8)
MeshLineArithmetic(name='M_12',
                color=Color['Grey'],
                number=8)
MeshLineArithmetic(name='M_13',
                color=Color['Grey'],
                number=10)
MeshLineArithmetic(name='M_14',
                color=Color['Grey'],
                number=4)
MeshLineGeometricNumber(name='M_15',
                        color=Color['Grey'],
                        number=6,
                        concentrationFactor='1.2')

# Definindo malha periodica para simular simetria/periodicidade
PeriodicityNumberZaxis(physicalType=PeriodicityCyclicType(),
                       repetitionNumber='3',
                       domainOriginAngle='0')
TransfRotation3AnglesPivotCoord(name='T_PERIODICA',
                                coordSys=CoordSys['XY1'],
                                pivotCoord=['0',
                                            '0'],
                                rotationAngles=RotationAngles(angleZ='120'))
MeshLineLinked(name='M_PERIODICA',
               color=Color['Grey'],
               linked=Transf['T_PERIODICA'])

#-------------------------------------------------------------------------------
# 4. Definindo pontos
```

```
#--------------------------------------------------------------------------------

PointCoordinates(color=Color['Grey'],
                 visibility=Visibility['VISIBLE'],
                 coordSys=CoordSys['ROTOR_CABECA_POLAR'],
                 uvw=['0', 'R1'],
                 nature=Nature['STANDARD'])
PointCoordinates(color=Color['Grey'],
                 visibility=Visibility['VISIBLE'],
                 coordSys=CoordSys['ROTOR_CABECA_POLAR'],
                 uvw=['R1*Sind(ALPHA_1/2)', 'R1*Cosd(ALPHA_1/2)'],
                 nature=Nature['STANDARD'])
PointCoordinates(color=Color['Grey'],
                 visibility=Visibility['VISIBLE'],
                 coordSys=CoordSys['ROTOR'],
                 uvw=['WPS/2', 'R4-DELTA_0-HPS'],
                 nature=Nature['STANDARD'])
PointCoordinates(color=Color['Grey'],
                 visibility=Visibility['VISIBLE'],
                 coordSys=CoordSys['ROTOR'],
                 uvw=['WP/2', 'R4-DELTA_0-HPS'],
                 nature=Nature['STANDARD'])
PointCoordinates(color=Color['Grey'],
                 visibility=Visibility['VISIBLE'],
                 coordSys=CoordSys['ROTOR'],
                 uvw=['WP/2', 'R6*Cosd(ALPHA_P2)'],
                 nature=Nature['STANDARD'])
PointCoordinates(color=Color['Grey'],
                 visibility=Visibility['VISIBLE'],
                 coordSys=CoordSys['ROTOR'],
                 uvw=['R6*Sind(ALPHA_P2)', 'R6*Cosd(ALPHA_P2)'],
                 nature=Nature['STANDARD'])
PointCoordinates(color=Color['Grey'],
                 visibility=Visibility['VISIBLE'],
                 coordSys=CoordSys['ROTOR'],
                 uvw=['R6*Sind(ALPHA_P/2)', 'R6*Cosd(ALPHA_P/2)'],
                 nature=Nature['STANDARD'])
PointCoordinates(color=Color['Grey'],
                 visibility=Visibility['VISIBLE'],
                 coordSys=CoordSys['ROTOR'],
                 uvw=['R0*Sind(ALPHA_P/2)', 'R0*Cosd(ALPHA_P/2)'],
                 nature=Nature['STANDARD'])
PointCoordinates(color=Color['Grey'],
                 visibility=Visibility['VISIBLE'],
                 coordSys=CoordSys['ROTOR'],
                 uvw=['0', 'R0'],
                 nature=Nature['STANDARD'])
PointCoordinates(color=Color['Grey'],
                 visibility=Visibility['VISIBLE'],
                 coordSys=CoordSys['ESTATOR'],
                 uvw=['0', 'R4'],
                 nature=Nature['STANDARD'])
PointCoordinates(color=Color['Grey'],
                 visibility=Visibility['VISIBLE'],
                 coordSys=CoordSys['ESTATOR'],
                 uvw=['R4*Sind(ALPHA_P/2)', 'R4*Cosd(ALPHA_P/2)'],
                 nature=Nature['STANDARD'])
PointCoordinates(color=Color['Grey'],
                 visibility=Visibility['VISIBLE'],
                 coordSys=CoordSys['ESTATOR'],
                 uvw=['R5*Sind(ALPHA_P/2)', 'R5*Cosd(ALPHA_P/2)'],
                 nature=Nature['STANDARD'])
PointCoordinates(color=Color['Grey'],
                 visibility=Visibility['VISIBLE'],
                 coordSys=CoordSys['ESTATOR'],
                 uvw=['0', 'R5'],
                 nature=Nature['STANDARD'])
```

```
#-------------------------------------------------------------------------------
# 6. Definindo linhas
#-------------------------------------------------------------------------------

# Encontrando pontos pelas coordenadas para definir as linhas sem depender da ordem
de criação dos pontos
p_cabeca_polar_1 = Point.selectByCoordinates(coordinates=[0,
ParameterGeom['R1'].value], coordSys=CoordSys['ROTOR_CABECA_POLAR'])

p_cabeca_polar_2 =
Point.selectByCoordinates(coordinates=[ParameterGeom['R1'].value*math.sin(math.radi
ans(ParameterGeom['ALPHA_1'].value/2)),
ParameterGeom['R1'].value*math.cos(math.radians(ParameterGeom['ALPHA_1'].value/2))]
, coordSys=CoordSys['ROTOR_CABECA_POLAR'])

p_cabeca_polar_3 =
Point.selectByCoordinates(coordinates=[ParameterGeom['WPS'].value/2,
ParameterGeom['R4'].value-ParameterGeom['DELTA_0'].value-
ParameterGeom['HPS'].value], coordSys=CoordSys['ROTOR'])

p_cabeca_polar_4 =
Point.selectByCoordinates(coordinates=[ParameterGeom['WP'].value/2,
ParameterGeom['R4'].value-ParameterGeom['DELTA_0'].value-
ParameterGeom['HPS'].value], coordSys=CoordSys['ROTOR'])

p_cabeca_polar_5 =
Point.selectByCoordinates(coordinates=[ParameterGeom['WP'].value/2,
ParameterGeom['R6'].value*math.cos(math.radians(ParameterGeom['ALPHA_P2'].value))],
coordSys=CoordSys['ROTOR'])

p_cabeca_polar_6 =
Point.selectByCoordinates(coordinates=[ParameterGeom['R6'].value*math.sin(math.radi
ans(ParameterGeom['ALPHA_P2'].value)),
ParameterGeom['R6'].value*math.cos(math.radians(ParameterGeom['ALPHA_P2'].value))],
coordSys=CoordSys['ROTOR'])

p_cabeca_polar_7 =
Point.selectByCoordinates(coordinates=[ParameterGeom['R6'].value*math.sin(math.radi
ans(ParameterGeom['ALPHA_P'].value/2)),
ParameterGeom['R6'].value*math.cos(math.radians(ParameterGeom['ALPHA_P'].value/2))]
, coordSys=CoordSys['ROTOR'])

p_cabeca_polar_8 =
Point.selectByCoordinates(coordinates=[ParameterGeom['R0'].value*math.sin(math.radi
ans(ParameterGeom['ALPHA_P'].value/2)),
ParameterGeom['R0'].value*math.cos(math.radians(ParameterGeom['ALPHA_P'].value/2))]
, coordSys=CoordSys['ROTOR'])

p_cabeca_polar_9 = Point.selectByCoordinates(coordinates=[0,
ParameterGeom['R0'].value], coordSys=CoordSys['ROTOR'])

p_estator_1 = Point.selectByCoordinates(coordinates=[0, ParameterGeom['R4'].value],
coordSys=CoordSys['ESTATOR'])

p_estator_2 =
Point.selectByCoordinates(coordinates=[ParameterGeom['R4'].value*math.sin(math.radi
ans(ParameterGeom['ALPHA_P'].value/2)),
ParameterGeom['R4'].value*math.cos(math.radians(ParameterGeom['ALPHA_P'].value/2))]
, coordSys=CoordSys['ESTATOR'])

p_estator_3 =
Point.selectByCoordinates(coordinates=[ParameterGeom['R5'].value*math.sin(math.radi
ans(ParameterGeom['ALPHA_P'].value/2)),
ParameterGeom['R5'].value*math.cos(math.radians(ParameterGeom['ALPHA_P'].value/2))]
, coordSys=CoordSys['ESTATOR'])

p_estator_4 = Point.selectByCoordinates(coordinates=[0, ParameterGeom['R5'].value],
coordSys=CoordSys['ESTATOR'])
```

```
# Linhas do rotor
LineArcPivotCoord(color=Color['Grey'],
                  visibility=Visibility['VISIBLE'],
                  coordSys=CoordSys['ROTOR_CABECA_POLAR'],
                  pivotCoord=['0', '0'],
                  defPoint=[p_cabeca_polar_2, p_cabeca_polar_1],
                  nature=Nature['STANDARD'],
                  mesh=MeshLine['M_02'])
LineSegment(color=Color['Grey'],
            visibility=Visibility['VISIBLE'],
            defPoint=[p_cabeca_polar_2,  p_cabeca_polar_3],
            nature=Nature['STANDARD'],
            mesh=MeshLine['M_05'])
LineSegment(color=Color['Grey'],
            visibility=Visibility['VISIBLE'],
            defPoint=[p_cabeca_polar_3, p_cabeca_polar_4],
            nature=Nature['STANDARD'],
        mesh=MeshLine['M_11'])
LineSegment(color=Color['Grey'],
            visibility=Visibility['VISIBLE'],
            defPoint=[p_cabeca_polar_4, p_cabeca_polar_5],
            nature=Nature['STANDARD'],
            mesh=MeshLine['M_13'])
LineSegment(color=Color['Grey'],
            visibility=Visibility['VISIBLE'],
            defPoint=[p_cabeca_polar_5, p_cabeca_polar_6],
            nature=Nature['STANDARD'],
            mesh=MeshLine['M_11'])
LineSegment(color=Color['Grey'],
            visibility=Visibility['VISIBLE'],
            defPoint=[p_cabeca_polar_6, p_cabeca_polar_3],
            nature=Nature['STANDARD'],
            mesh=MeshLine['M_13'])
LineArcPivotCoord(color=Color['Grey'],
                  visibility=Visibility['VISIBLE'],
                  coordSys=CoordSys['ROTOR'],
                  pivotCoord=['0', '0'],
                  defPoint=[p_cabeca_polar_7, p_cabeca_polar_6],
                  nature=Nature['STANDARD'])
LineSegment(color=Color['Grey'],
            visibility=Visibility['VISIBLE'],
            defPoint=[p_cabeca_polar_8, p_cabeca_polar_7],
            nature=Nature['STANDARD'],
            mesh=MeshLine['M_15'])
Line.selectByCoordinates(coordinates=[ParameterGeom['R6'].value*math.sin(math.radia
ns(ParameterGeom['ALPHA_P'].value/2)) - 1,
ParameterGeom['R6'].value*math.cos(math.radians(ParameterGeom['ALPHA_P'].value/2))
- 1], coordSys=CoordSys['ROTOR']).orientMesh()
LineArcPivotCoord(color=Color['Grey'],
                  visibility=Visibility['VISIBLE'],
                  coordSys=CoordSys['ROTOR'],
                  pivotCoord=['0', '0'],
                  defPoint=[p_cabeca_polar_8, p_cabeca_polar_9],
                  nature=Nature['STANDARD'],
                  mesh=MeshLine['M_12'])
LineSegment(color=Color['Grey'],
            visibility=Visibility['VISIBLE'],
            defPoint=[p_cabeca_polar_9, p_cabeca_polar_1],
            nature=Nature['STANDARD'],
            mesh=MeshLine['M_04'])
Line.selectByCoordinates(coordinates=[0,
(ParameterGeom['R0'].value+ParameterGeom['R1'].value)/2],
coordSys=CoordSys['ROTOR']).orientMesh()

# Linhas do estator
LineArcPivotCoord(color=Color['Grey'],
                  visibility=Visibility['VISIBLE'],
```

```
                  coordSys=CoordSys['ESTATOR'],
                  pivotCoord=['0', '0'],
                  defPoint=[p_estator_2, p_estator_1],
                  nature=Nature['STANDARD'],
                  mesh=MeshLine['M_01'])
LineArcPivotCoord(color=Color['Grey'],
                visibility=Visibility['VISIBLE'],
                coordSys=CoordSys['ESTATOR'],
                pivotCoord=['0', '0'],
                defPoint=[p_estator_3, p_estator_4],
                nature=Nature['STANDARD'],
                mesh=MeshLine['M_08'])
LineSegment(color=Color['Grey'],
            visibility=Visibility['VISIBLE'],
            defPoint=[p_estator_1, p_estator_4],
            nature=Nature['STANDARD'],
            mesh=MeshLine['M_06'])
LineSegment(color=Color['Grey'],
            visibility=Visibility['VISIBLE'],
            defPoint=[p_estator_2, p_estator_3],
            nature=Nature['STANDARD'],
            mesh=MeshLine['M_06'])

# Fechando o domínio
LineSegment(color=Color['Grey'],
            visibility=Visibility['VISIBLE'],
            defPoint=[p_estator_1, p_cabeca_polar_1],
            nature=Nature['STANDARD'],
            relaxation=RelaxLine['AIDED_RELAXLINE'],
            mesh=MeshLine['M_03'])
LineSegment(color=Color['Grey'],
            visibility=Visibility['VISIBLE'],
            defPoint=[p_cabeca_polar_7, p_estator_2],
            nature=Nature['STANDARD'],
            mesh=MeshLine['M_07'])
Line.selectByCoordinates(coordinates=[(ParameterGeom['R6'].value*math.sin(math.radi
ans(ParameterGeom['ALPHA_P'].value/2)) +
ParameterGeom['R4'].value*math.sin(math.radians(ParameterGeom['ALPHA_P'].value/2)))
/ 2,

(ParameterGeom['R6'].value*math.cos(math.radians(ParameterGeom['ALPHA_P'].value/2))
+
ParameterGeom['R4'].value*math.cos(math.radians(ParameterGeom['ALPHA_P'].value/2)))
/2],
                        coordSys=CoordSys['ROTOR']).orientMesh()


#-------------------------------------------------------------------------------
# 7. Definindo as faces, regiões e periodicidade
#-------------------------------------------------------------------------------

# Definindo caso magnético
ApplicationMagneticDC2D(domain2D=Domain2DPlane(lengthUnit=LengthUnit['MILLIMETER'],
                                               depth='900'),
                        coilCoefficient=CoilCoefficientAutomatic())

# Importando material para definição das faces
Material(name='SSTEEL_EQUA : SILICON STEEL EQUACIONAL',
         propertyBH=PropertyBhNonlinearSpline(splinePoints=[
                           BHPoint(h=0.0, b=0.0),
                           BHPoint(h=41.0, b=0.05),
                           BHPoint(h=46.0, b=0.1),
                           BHPoint(h=51.0, b=0.15),
                           BHPoint(h=56.0, b=0.2),
                           BHPoint(h=61.0, b=0.25),
                           BHPoint(h=66.0, b=0.3),
                           BHPoint(h=75.0, b=0.35),
                           BHPoint(h=81.0, b=0.4),
```

```
                                BHPoint(h=85.0, b=0.45),
                                BHPoint(h=89.0, b=0.5),
                                BHPoint(h=91.0, b=0.55),
                                BHPoint(h=96.0, b=0.6),
                                BHPoint(h=100.0, b=0.65),
                                BHPoint(h=103.0, b=0.7),
                                BHPoint(h=110.0, b=0.75),
                                BHPoint(h=113.0, b=0.8),
                                BHPoint(h=121.0, b=0.85),
                                BHPoint(h=126.0, b=0.9),
                                BHPoint(h=133.0, b=0.95),
                                BHPoint(h=140.0, b=1.0),
                                BHPoint(h=150.0, b=1.05),
                                BHPoint(h=160.0, b=1.1),
                                BHPoint(h=175.0, b=1.15),
                                BHPoint(h=190.0, b=1.2),
                                BHPoint(h=210.0, b=1.25),
                                BHPoint(h=240.0, b=1.3),
                                BHPoint(h=281.0, b=1.35),
                                BHPoint(h=373.0, b=1.4),
                                BHPoint(h=520.0, b=1.45),
                                BHPoint(h=750.0, b=1.5),
                                BHPoint(h=1160.0, b=1.55),
                                BHPoint(h=1800.0, b=1.6),
                                BHPoint(h=2670.0, b=1.65),
                                BHPoint(h=4000.0, b=1.7),
                                BHPoint(h=5500.0, b=1.75),
                                BHPoint(h=7500.0, b=1.8),
                                BHPoint(h=10300.0, b=1.85),
                                BHPoint(h=12300.0, b=1.9),
                                BHPoint(h=16000.0, b=1.95),
                                BHPoint(h=21000.0, b=2.0),
                                BHPoint(h=27000.0, b=2.05),
                                BHPoint(h=35000.0, b=2.1),
                                BHPoint(h=45000.0, b=2.15),
                                BHPoint(h=58000.0, b=2.2),
                                BHPoint(h=76000.0, b=2.25),
                                BHPoint(h=98000.0, b=2.3),
                                BHPoint(h=128000.0, b=2.35),
                                BHPoint(h=168000.0, b=2.4),
                                BHPoint(h=218000.0, b=2.45),
                                BHPoint(h=280000.0, b=2.5),
                                BHPoint(h=360000.0, b=2.55),
                                BHPoint(h=460000.0, b=2.6),
                                BHPoint(h=526314.55962, b=2.65),
                                BHPoint(h=566103.29539, b=2.7),
                                BHPoint(h=605892.03117, b=2.75)]))


# Definição automática das faces
buildFaces()

# Encontrando as faces geradas automáticamente independente da ordem
face_rotor =
Face.selectByCoordinates(coordinates=[1,0.5*(ParameterGeom['R0'].value+ParameterGeo
m['R1'].value)],coordSys=CoordSys['ROTOR'])
face_estator =
Face.selectByCoordinates(coordinates=[1,0.5*(ParameterGeom['R4'].value+ParameterGeo
m['R5'].value)],coordSys=CoordSys['ESTATOR'])
face_bobinas = Face.selectByCoordinates(coordinates=[ParameterGeom['WPS'].value/2 -
1,ParameterGeom['R4'].value-ParameterGeom['DELTA_0'].value-
ParameterGeom['HPS'].value-ParameterGeom['HR'].value/2],coordSys=CoordSys['ROTOR'])
face_entreferro =
Face.selectByCoordinates(coordinates=[1,ParameterGeom['R4'].value-
ParameterGeom['DELTA_0'].value/2],coordSys=CoordSys['ROTOR'])

# Atribuindo gerador de malha mapped para face perfeitamente retangular
face_bobinas.meshGenerator=MeshGenerator['MAPPED']
```

```
#Definindo as regiões
RegionFace(name='ESTATOR',
           magneticDC2D=MagneticDC2DFaceMagnetic(material=Material['SSTEEL_EQUA']),
            visibility=Visibility['VISIBLE'],
            color=Color['Grey_Steel'])
face_estator.assignRegion(region=RegionFace['ESTATOR'])
RegionFace(name='ROTOR',
           magneticDC2D=MagneticDC2DFaceMagnetic(material=Material['SSTEEL_EQUA']),
            visibility=Visibility['VISIBLE'],
            color=Color['Grey_Steel'])
face_rotor.assignRegion(region=RegionFace['ROTOR'])
RegionFace(name='ENTREFERRO',
           magneticDC2D=MagneticDC2DFaceVacuum(),
           visibility=Visibility['VISIBLE'],
           color=Color['Black'])
face_entreferro.assignRegion(region=RegionFace['ENTREFERRO'])
RegionFace(name='AR',
           magneticDC2D=MagneticDC2DFaceVacuum(),
           visibility=Visibility['VISIBLE'],
           color=Color['Black'])

# Propagando faces
TransfAffineLine2PT(name='Transf_1',
                    point=[p_cabeca_polar_7, p_cabeca_polar_8],
                    factor='-1')
FaceAutomatic[ALL].propagate(transformation=Transf['TRANSF_1'],
          repetitionNumber=1,
          buildingOption='FacesWithMeshGenerator',
          regionPropagation='Same')

TransfAffineLine2PT(name='Transf_2',
                    point=[p_estator_1, p_cabeca_polar_1],
                    factor='-1')
FaceAutomatic[ALL].propagate(transformation=Transf['TRANSF_2'],
          repetitionNumber=1,
          buildingOption='FacesWithMeshGenerator',
          regionPropagation='Same',
          regionPeriodicity=1)


# Criando camada de ar externa (Poderia ter criado antes da propagação de faces e
depois ter propagado a face criada)
PointCoordinates(color=Color['Grey'],
                visibility=Visibility['VISIBLE'],
                coordSys=CoordSys['ESTATOR'],
                uvw=['1.2*R5*Sind(ALPHA_P)', '1.2*R5*Cosd(ALPHA_P)'],
                nature=Nature['STANDARD'])
PointCoordinates(color=Color['Grey'],
                visibility=Visibility['VISIBLE'],
                coordSys=CoordSys['ESTATOR'],
                uvw=['1.2*R5*Sind(-ALPHA_P)', '1.2*R5*Cosd(-ALPHA_P)'],
                nature=Nature['STANDARD'])
LineArcPivotCoord(color=Color['Grey'],
                  visibility=Visibility['VISIBLE'],
                  coordSys=CoordSys['ROTOR_CABECA_POLAR'],
                  pivotCoord=['0', '0'],

defPoint=[Point.selectByCoordinates(coordinates=[1.2*ParameterGeom['R5'].value*math
.sin(math.radians(ParameterGeom['ALPHA_P'].value)),
1.2*ParameterGeom['R5'].value*math.cos(math.radians(ParameterGeom['ALPHA_P'].value)
)], coordSys=CoordSys['ESTATOR']),

Point.selectByCoordinates(coordinates=[1.2*ParameterGeom['R5'].value*math.sin(math.
radians(-ParameterGeom['ALPHA_P'].value)),
1.2*ParameterGeom['R5'].value*math.cos(math.radians(-
ParameterGeom['ALPHA_P'].value))], coordSys=CoordSys['ESTATOR'])],
                  nature=Nature['STANDARD'],
```

```
                mesh=MeshLine['M_10'])
LineSegment(color=Color['Grey'],
            visibility=Visibility['VISIBLE'],

defPoint=[Point.selectByCoordinates(coordinates=[1.2*ParameterGeom['R5'].value*math
.sin(math.radians(ParameterGeom['ALPHA_P'].value)),
1.2*ParameterGeom['R5'].value*math.cos(math.radians(ParameterGeom['ALPHA_P'].value)
)], coordSys=CoordSys['ESTATOR']),

Point.selectByCoordinates(coordinates=[ParameterGeom['R5'].value*math.sin(math.radi
ans(ParameterGeom['ALPHA_P'].value)),
ParameterGeom['R5'].value*math.cos(math.radians(ParameterGeom['ALPHA_P'].value))],
coordSys=CoordSys['ESTATOR'])],
            nature=Nature['STANDARD'],
            mesh=MeshLine['M_09'])
Line.selectByCoordinates(coordinates=[1.1*ParameterGeom['R5'].value*math.sin(math.r
adians(ParameterGeom['ALPHA_P'].value)),
1.1*ParameterGeom['R5'].value*math.cos(math.radians(ParameterGeom['ALPHA_P'].value)
)], coordSys=CoordSys['ESTATOR']).orientMesh()
LineSegment(color=Color['Grey'],
            visibility=Visibility['VISIBLE'],

defPoint=[Point.selectByCoordinates(coordinates=[1.2*ParameterGeom['R5'].value*math
.sin(math.radians(-ParameterGeom['ALPHA_P'].value)),
1.2*ParameterGeom['R5'].value*math.cos(math.radians(-
ParameterGeom['ALPHA_P'].value))], coordSys=CoordSys['ESTATOR']),

Point.selectByCoordinates(coordinates=[ParameterGeom['R5'].value*math.sin(math.radi
ans(-ParameterGeom['ALPHA_P'].value)),
ParameterGeom['R5'].value*math.cos(math.radians(-ParameterGeom['ALPHA_P'].value))],
coordSys=CoordSys['ESTATOR'])],
            nature=Nature['STANDARD'],
            mesh=MeshLine['M_PERIODICA'])
buildFaces()
face_camada_de_ar = Face.selectByCoordinates(coordinates=[0,
(1.2*ParameterGeom['R5'].value +
ParameterGeom['R5'].value)/2],coordSys=CoordSys['ESTATOR'])
face_camada_de_ar.assignRegion(region=RegionFace['AR'])

# Criando bobinas
CoilConductorImposedCurrent(name='BOBINA', rmsModulus='31')

RegionFace(name='BOBINA_POSITIVA',

magneticDC2D=MagneticDC2DFaceCoilConductor(coilConductor=CoilConductor2DPositive(tu
rnNumber='200', seriesParallel=AllInSeries(),
electricComponent=CoilConductor['BOBINA'])),
           visibility=Visibility['VISIBLE'],
           color=Color['Red'])
RegionFace(name='BOBINA_NEGATIVA',

magneticDC2D=MagneticDC2DFaceCoilConductor(coilConductor=CoilConductor2DNegative(tu
rnNumber='200', seriesParallel=AllInSeries(),
electricComponent=CoilConductor['BOBINA'])),
           visibility=Visibility['VISIBLE'],
           color=Color['Blue'])

x1 = ParameterGeom['WPS'].value/2 - 1
y1= ParameterGeom['R4'].value-ParameterGeom['DELTA_0'].value-
ParameterGeom['HPS'].value-ParameterGeom['HR'].value/2
teta = math.atan(x1/y1)
gama = math.pi/3 - 2*teta
R = math.sqrt(x1**2 + y1**2)

face_bobina_1 = Face.selectByCoordinates(coordinates=[x1,
y1],coordSys=CoordSys['ROTOR'])
face_bobina_2 = Face.selectByCoordinates(coordinates=[R*math.sin(teta+gama),
R*math.cos(teta+gama)],coordSys=CoordSys['ROTOR']) # Desenho 5
```

```
face_bobina_3 = Face.selectByCoordinates(coordinates=[-x1,
y1],coordSys=CoordSys['ROTOR'])
face_bobina_4 = Face.selectByCoordinates(coordinates=[-R*math.sin(teta+gama),
R*math.cos(teta+gama)],coordSys=CoordSys['ROTOR'])

face_bobina_1.assignRegion(region=RegionFace['BOBINA_POSITIVA'])
face_bobina_2.assignRegion(region=RegionFace['BOBINA_POSITIVA'])
face_bobina_3.assignRegion(region=RegionFace['BOBINA_NEGATIVA'])
face_bobina_4.assignRegion(region=RegionFace['BOBINA_NEGATIVA'])

# Atribuindo malha periódica para as as linhas nas bordas laterais para simular
simetria/periodicidade
Line.selectByCoordinates(coordinates=[(ParameterGeom['R0'].value+(ParameterGeom['R1
'].value+6))/2*math.sin(math.radians(-ParameterGeom['ALPHA_P'].value)),
(ParameterGeom['R0'].value+(ParameterGeom['R1'].value+6))/2*math.cos(math.radians(-
ParameterGeom['ALPHA_P'].value))],
coordSys=CoordSys['ROTOR']).assignMeshLine(meshLine=MeshLine['M_PERIODICA'])
Line.selectByCoordinates(coordinates=[(ParameterGeom['R4'].value-
ParameterGeom['DELTA_0'].value/2)*math.sin(math.radians(-
ParameterGeom['ALPHA_P'].value)), (ParameterGeom['R4'].value-
ParameterGeom['DELTA_0'].value/2)*math.cos(math.radians(-
ParameterGeom['ALPHA_P'].value))],
coordSys=CoordSys['ESTATOR']).assignMeshLine(meshLine=MeshLine['M_PERIODICA'])
Line.selectByCoordinates(coordinates=[(ParameterGeom['R4'].value+ParameterGeom['R5'
].value)/2*math.sin(math.radians(-ParameterGeom['ALPHA_P'].value)),
(ParameterGeom['R4'].value+ParameterGeom['R5'].value)/2*math.cos(math.radians(-
ParameterGeom['ALPHA_P'].value))],
coordSys=CoordSys['ESTATOR']).assignMeshLine(meshLine=MeshLine['M_PERIODICA'])


#-------------------------------------------------------------------------------
# 10. Condições de contorno
#-------------------------------------------------------------------------------

# Condição de contorno: Campo tangencial na região mais interna do rotor
RegionLine(name='INTERNA : LINHA INTERNA PARA O POTENCIAL',
           magneticDC2D=MagneticDC2DLineTangentField(),
           visibility=Visibility['VISIBLE'])
Line.selectByCoordinates(coordinates=[1, ParameterGeom['R0'].value],
coordSys=CoordSys['ROTOR']).assignRegion(region=RegionLine['INTERNA'])
Line.selectByCoordinates(coordinates=[-1, ParameterGeom['R0'].value],
coordSys=CoordSys['ROTOR']).assignRegion(region=RegionLine['INTERNA'])
Line.selectByCoordinates(coordinates=[ParameterGeom['R0'].value*math.sin(math.radia
ns(ParameterGeom['ALPHA_P'].value/2)) + 1,
ParameterGeom['R0'].value*math.cos(math.radians(ParameterGeom['ALPHA_P'].value/2))]
, coordSys=CoordSys['ROTOR']).assignRegion(region=RegionLine['INTERNA'])
Line.selectByCoordinates(coordinates=[ParameterGeom['R0'].value*math.sin(math.radia
ns(-ParameterGeom['ALPHA_P'].value/2)) - 1,
ParameterGeom['R0'].value*math.cos(math.radians(-
ParameterGeom['ALPHA_P'].value/2))],
coordSys=CoordSys['ROTOR']).assignRegion(region=RegionLine['INTERNA'])

# Condição de contorno: Campo nulo na região após a região de ar externa
RegionLine(name='EXTERNA : POTENCIAL DE REFERENCIA NULO',
           magneticDC2D=MagneticDC2DLineImposedFluxConstant(value='0'),
           visibility=Visibility['VISIBLE'])
Line.selectByCoordinates(coordinates=[0, 1.2*ParameterGeom['R5'].value],
coordSys=CoordSys['ROTOR']).assignRegion(region=RegionLine['EXTERNA'])

# Cria a malha de acordo com as configurações definidas
meshDomain()
checkPhysic()
# Geração das curvas de cada cenário de simulação e extração de resultados
dict_resultados = {}
lista_resultados = {'r1': [], 'wps': [], 'distorcao': [], 'x': [], 'y': [],
'meanValue': [], 'rectifiedMeanValue': [], 'integral': [], 'minimalValues': [],
'maximalValues': [], 'harmonicosEspectro': [], 'amplitudeEspectro': [],
```

```
'magnitudeEspectro': [], 'phaseEspectro': [], 'frequencyEspectro': [],
'fatorDeDistorcaoHarmonico': []}
menorDistorcao = {'R1': 0, 'WPS': 0, 'distorcao': 1000}

for r1 in range(minValueR1, maxValueR1 + 1, stepValue):
       dict_resultados[r1] = {}
       for wps in range(minValueWPS, maxValueWPS + 1, stepValue):
              # Selecionando variação de parâmetros do cenário de simulação
              selectCurrentStep(activeScenario=Scenario['VAR_R1_WPS'],
                                parameterValue=['R1='+str(r1), 'WPS='+str(wps)])

              # Gerando curva
              nome = 'INDUCAO_' + str(r1) + '_' + str(wps)
              SpatialCurve(name=nome,
                        compoundPath=CompoundPath['ENTREFERRO'],
                        formula=['B*N'])

              # Gerando espectro da curva
              nomeEspectro = 'ESPECTRO_' + str(r1) + '_' + str(wps)
              SpectrumAnalysis(curve=Curve2d[nome],
                        component='B*N',
                        cycle=FullCycle(),
                        numberHarmonics=11,
                        computeDCComponent=NonContinue(),
                        scaleDecibel=FirstHarmonic(firstHarmonicValue=100.0),
                        nameRebuiltCurve='CURVA_' + nomeEspectro,
                        nameFourierTransform=nomeEspectro)

              # Extraindo dados para dicionário de forma de fácil acesso a valores
              de R1 e WPS específicos. Exemplo:
              dict_resultados[900][550]['meanValue']
              dict_resultados[r1][wps] = {}
              dict_resultados[r1][wps]['x'] = Curve2d[nome].xAxis.x
              dict_resultados[r1][wps]['y'] = Curve2d[nome].y[0].values
              dict_resultados[r1][wps]['meanValue'] =
              Curve2d[nome].y[0].meanValues[0]
              dict_resultados[r1][wps]['rectifiedMeanValue']=Curve2d[nome].y[0].rect
              ifiedMeanValues[0]
              dict_resultados[r1][wps]['integral'] = Curve2d[nome].y[0].integral[0]
              dict_resultados[r1][wps]['minimalValues'] =
              Curve2d[nome].y[0].minimalValues[0]
              dict_resultados[r1][wps]['maximalValues'] =
              Curve2d[nome].y[0].maximalValues[0]
              dict_resultados[r1][wps]['harmonicosEspectro'] =
              Curve2d[nomeEspectro].xAxis.x
              dict_resultados[r1][wps]['amplitudeEspectro'] =
              Curve2d[nomeEspectro].y[0].values
              dict_resultados[r1][wps]['magnitudeEspectro'] =
              Curve2d[nomeEspectro].y[1].values
              dict_resultados[r1][wps]['phaseEspectro'] =
              Curve2d[nomeEspectro].y[2].values
              dict_resultados[r1][wps]['frequencyEspectro'] =
              Curve2d[nomeEspectro].y[3].values
              dict_resultados[r1][wps]['fatorDeDistorcaoHarmonico'] =
              math.sqrt(sum([x**2 for x in
              dict_resultados[r1][wps]['amplitudeEspectro']])) /
              dict_resultados[r1][wps]['amplitudeEspectro'][0]

              # Extraindo resultados para lista de forma de fácil a facilitar a
              passagem de parâmetros para visualizações gráficas.
              lista_resultados['r1'].append(r1)
              lista_resultados['wps'].append(wps)
              lista_resultados['x'].append(dict_resultados[r1][wps]['x'])
              lista_resultados['y'].append(dict_resultados[r1][wps]['y'])
              lista_resultados['meanValue'].append(dict_resultados[r1][wps]['meanVal
              ue'])
              lista_resultados['rectifiedMeanValue'].append(dict_resultados[r1][wps]
              ['rectifiedMeanValue'])
```

```
            lista_resultados['integral'].append(dict_resultados[r1][wps]['integral
            '])
            lista_resultados['minimalValues'].append(dict_resultados[r1][wps]['min
            imalValues'])
            lista_resultados['maximalValues'].append(dict_resultados[r1][wps]['max
            imalValues'])
            lista_resultados['harmonicosEspectro'].append(dict_resultados[r1][wps]
            ['harmonicosEspectro'])
            lista_resultados['amplitudeEspectro'].append(dict_resultados[r1][wps][
            'amplitudeEspectro'])
            lista_resultados['magnitudeEspectro'].append(dict_resultados[r1][wps][
            'magnitudeEspectro'])
            lista_resultados['phaseEspectro'].append(dict_resultados[r1][wps]['pha
            seEspectro'])
            lista_resultados['frequencyEspectro'].append(dict_resultados[r1][wps][
            'frequencyEspectro'])
            lista_resultados['fatorDeDistorcaoHarmonico'].append(dict_resultados[r
            1][wps]['fatorDeDistorcaoHarmonico'])

            # Comparando resultados
            if dict_resultados[r1][wps]['fatorDeDistorcaoHarmonico'] <
      menorDistorcao['distorcao']:
            menorDistorcao['R1'] = r1
            menorDistorcao['WPS'] = wps
            menorDistorcao['distorcao'] =
      dict_resultados[r1][wps]['fatorDeDistorcaoHarmonico']

# Exporta resultados
import json
with open("r1.json", "w") as json_file:
    json.dump(lista_resultados['r1'], json_file)
with open("wps.json", "w") as json_file:
    json.dump(lista_resultados['wps'], json_file)
with open("qualidade.json", "w") as json_file:
    json.dump(lista_resultados['fatorDeDistorcaoHarmonico'], json_file)

# Imprimindo resultados da comparação na tela
print('Menor distorção: R1 = ' + str(menorDistorcao['R1']) + ', WPS = ' +
str(menorDistorcao['WPS']) + ', distorção = ' + str(menorDistorcao['distorcao']))
```