**UNIVERSIDADE FEDERAL DE SANTA CATARINA**
**CENTRO TECNOLÓGICO**
**DEPARTAMENTO DE ENGENHARIA ELÉTRICA E ELETRÔNICA**
**CURSO DE ENGENHARIA ELÉTRICA**

**Artur Nunes Pires Lopes**

# Implementação de Ferramenta Computacional de Cálculo Analítico da Reatância de Dispersão de Transformadores pelos Métodos de Roth e Rabins

**Florianópolis**
**2021**

Artur Nunes Pires Lopes

# Implementação de Ferramenta Computacional de Cálculo Analítico da Reatância de Dispersão de Transformadores pelos Métodos de Roth e Rabins

**Trabalho Conclusão do Curso de Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Bacharel em Engenharia Elétrica.**
**Orientador: Prof. Mauricio Valencia Ferreira da Luz, Dr.**

Florianópolis, Setembro de 2021

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Lopes, Artur Nunes Pires
Implementação de ferramenta computacional de cálculo analítico da reatância de dispersão de transformadores pelos métodos de Roth e Rabins / Artur Nunes Pires Lopes ; orientador, Mauricio Valencia Ferreira da Luz, 2021.
95 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Elétrica, Florianópolis, 2021.

Inclui referências.

1. Engenharia Elétrica. 2. Método analítico de resolução de EDP. 3. Reatância de dispersão de transformador. I. Ferreira da Luz, Mauricio Valencia. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Elétrica. III. Título.

Artur Nunes Pires Lopes

**Implementação de Ferramenta Computacional de Cálculo Analítico da Reatância de Dispersão de Transformadores pelos Métodos de Roth e Rabins**

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Engenharia Elétrica" e aceito, em sua forma final, pelo Curso de Graduação em Engenharia Elétrica.

Florianópolis, 29 de setembro de 2021.


Documento assinado digitalmente
Jean Vianei Leite
Data: 29/09/2021 14:04:19-0300
CPF: 003.474.909-80
Verifique as assinaturas em https://v.ufsc.br

---

Prof. Jean Vianei Leite, Dr.
Coordenador do Curso de Graduação em Engenharia Elétrica

**Banca Examinadora:**


Documento assinado digitalmente
Mauricio Valencia Ferreira da Luz
Data: 29/09/2021 13:04:40-0300
CPF: 960.926.969-91
Verifique as assinaturas em https://v.ufsc.br

---

Prof. Mauricio Valencia Ferreira da Luz, Dr.
Orientador
Universidade Federal de Santa Catarina


Documento assinado digitalmente
Walter Pereira Carpes Junior
Data: 29/09/2021 13:39:38-0300
CPF: 572.566.599-20
Verifique as assinaturas em https://v.ufsc.br

---

Prof. Walter Pereira Carpes Júnior, Dr.
Universidade Federal de Santa Catarina

Documento assinado digitalmente
geovane.ribeiro
Data: 29/09/2021 12:51:35-0300
CPF: 066.838.499-94

---

Prof. Geovane Romeu Ribeiro.
Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina

## AGRADECIMENTOS

Agradeço a minha família, minha mãe Sylvia Nunes Pires Lopes, meu pai Rodrigo Lopes e minha irmã Ana Paula Nunes Pires Lopes pelo apoio incondicional, dedicação, conselhos e pela educação que me proporcionaram. Aos meus demais familiares, avós, tios e primos pelo apoio e companhia.

À minha namorada Jéssica Alves pelo companheirismo. Nestes últimos três anos, nos apoiamos em todos os momentos difíceis e desfrutamos juntos de vários momentos felizes.

Aos meus amigos que conheci na universidade, que sempre fizeram dos meus dias, melhores e mais divertidos. Aos meus amigos do ensino técnico que nunca esquecerei.

A esta instituição de ensino superior, corpo docente, servidores e colaboradores que contribuem todos os dias para a formação dos cidadãos que aqui estudam. Aos meus professores do curso técnico integrado no IFSC, em especial o meu pai, sem vocês certamente eu teria mais dificuldade em escolher e trilhar o meu caminho no ensino superior.

Ao meu orientador Mauricio Valencia Ferreira da Luz por seus conselhos desde a disciplina de Eletromagnetismo e durante o restante da graduação, pelas diversas disciplinas ministradas e pelas orientações que levaram à Redação deste trabalho.

Aos colegas e amigos da Reivax, em especial ao Alécio José Grzybowski Júnior e ao Marcelo Schmidt Jacobsen, juntos compartilhamos diversas risadas e momentos de aprendizado que contribuíram para o meu desenvolvimento social, técnico e profissional.

E a todos que direta ou indiretamente fizeram parte do meu crescimento e formação.

## RESUMO

O presente trabalho descreve a programação de uma biblioteca computacional que executa o cálculo da indutância de dispersão de transformadores para uma geometria qualquer de entrada. É revisado o estado da arte dos métodos analíticos, como se desenvolveram os métodos de Kapp, Rogowski, Roth e Rabins. São comentadas as deduções de cada pesquisador com foco nas escolhas de simplificações que tornam o problema de campo magnético disperso solucionável analiticamente. São programados os métodos analíticos de Roth e Rabins para geometrias genéricas, bem como uma interface com o *software* de elementos finitos *Femm* com fim de comparação com os métodos analíticos. A interface com o *Femm* é programada de forma que o usuário possa utilizar as simplificações propostas por Roth, por Rabins ou uma combinação entre elas para a resolução do problema numericamente. Além das indutâncias de dispersão, a ferramenta computacional permite a confecção de gráficos dos campos de densidade de corrente, vetor potencial magnético e indução magnética dentro do domínio da janela do transformador conforme os cálculos analíticos e numéricos. Ao fim deste trabalho, foram testadas as vantagens e desvantagens de cada um dos métodos através da construção de geometrias que geram problemas para as suposições de Rabins e Roth. O método de Roth apresenta problemas quando o campo magnético se espalha por grande parte da janela, por desprezar a curvatura da geometria. A teoria de Rabins é extremamente robusta a qualquer geometria de transformador com que trata, porém sua implementação numérica apresenta problemas de convergência ao se deparar com descrições complexas de corrente em função da posição $y$. Este trabalho também compara os resultados analíticos e numéricos com o ensaio de um transformador real fornecido por um fabricante brasileiro de transformador, nesta comparação todos os métodos apresentam erros relativos à medida inferiores a 2,5%, um resultados extremamente satisfatório quando comparado ao erro máximo permitido por norma de 10%.



**Palavras-chave**: campo magnético disperso; transformador; indutância de dispersão; reatância de dispersão; método de elementos finitos; método de Roth; método de Rabins; linguagem *python*.

## ABSTRACT

This papper describes the programming of a computational library that calculates the leakage inductance of transformers for any input geometry. The state of the art analytical methods is reviewed, the development of Kapp's, Rogowski's, Roth's and Rabins's studied, focusing on the simplification choices that make the stray magnetic field problem analytically solvable. The Roth's and Rabins's analytical methods are programmed for generic geometries, as well as an interface with the finite element software *Femm* in order to compare them to the analytical methods. The interface with *Femm* is programmed so that the user can use the simplifications proposed by Roth, by Rabins or a combination of them to solve the problem numerically. In addition to the leakage inductances, the computational tool allows for plotting of current density, magnetic potential vector and magnetic induction fields within the transformer window domain, according to the analytical and numerical methods. At the end of this work, the advantages and disadvantages of each method were tested through the construction of geometries that generate problems for the Rabins' and Roth's assumptions. Roth's method has problems when the magnetic field spreads over a large part of the window, because it neglects the curvature of the geometry. Rabins' theory is extremely robust to any transformer geometry it deals with, but its numerical implementation presents convergence problems when faced with complex descriptions of current as a function of the $y$ position. This work also compares the analytical and numerical results with the test of a real transformer supplied by a Brazilian transformer manufacturer, in this comparison all methods present measurement errors below 2.5%, an extremely satisfactory result when compared to the maximum error allowed in the industry standard of 10%.



**Keywords**: stray magnetic field; transformer; leakage inductance; leakage reactance; finite element method; Roth's method; Rabins's method; python programming language.

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE SÍMBOLOS

**A** : Vetor potencial magnético [volt segundo por metro].

**B** : Vetor indução magnética [tesla].

**D** : Vetor deslocamento elétrico [coulomb por metro quadrado].

**E** : Vetor campo elétrico [volt por metro].

**H** : Vetor campo magnético [ampere por metro].

$i$ : Corrente elétrica [ampere].

**J** : Vetor densidade superficial de corrente [ampere por metro quadrado].

$L$ : Indutância [henry].

**M** : Vetor magnetização [tesla].

$p$ : Potência elétrica [watt].

**P** : Vetor polarização [coulomb por metro quadrado].

$t$ : Tempo [segundo].

$v$ : Tensão elétrica [volt].

$W$ : Energia armazenada no campo magnético [joule].

$X_L$ : Reatância indutiva [ohm].

$\epsilon$ : Permissividade elétrica do meio [farad por metro].

$\mu$ : Permeabilidade magnética do meio [henry por metro].

$\mu_0$ : Permeabilidade magnética do ar [henry por metro].

$\pi$ : Constante matemática pi.

$\rho$ : Densidade volumétrica de carga [coulomb por metro cúbico].

$\omega$ : Frequência angular [radianos por segundo].

$\partial$ : Operador matemático diferencial.

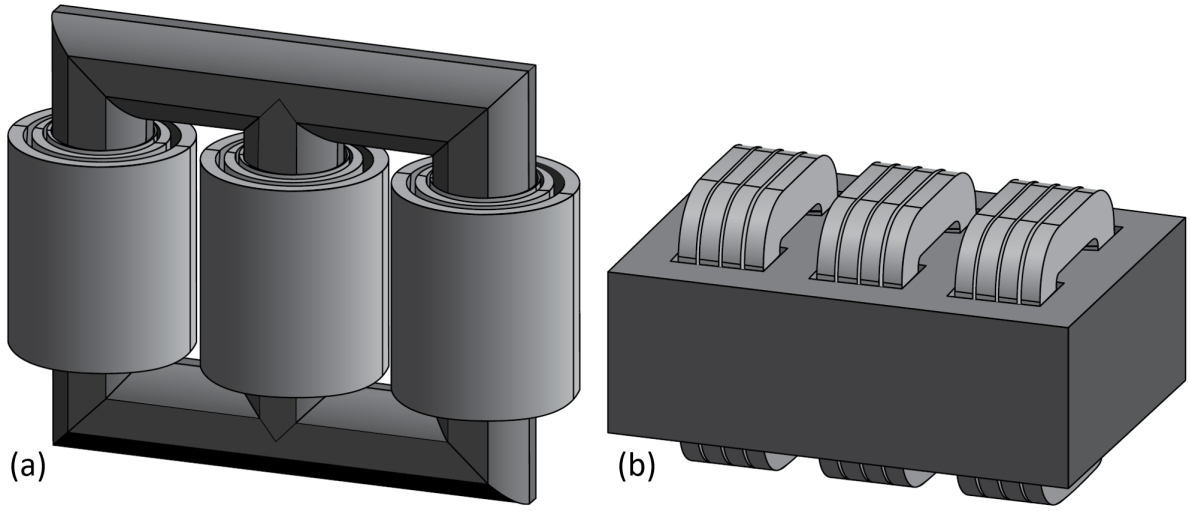
$\nabla$ : Operador matemático Nabla.

SUMÁRIO

# 1 INTRODUÇÃO

A partir da descoberta do princípio da indução por Michael Faraday, transformadores passaram a se tornar cada vez mais importantes nos sistemas elétricos de potência. Transformadores são máquinas elétricas estáticas que realizam a conversão de nível de tensão entre dois circuitos. Na figura 1 estão representados os dois tipos de transformadores no que se refere à construção do núcleo ferromagnético. Transformadores de núcleo envolvido são os transformadores em que o material condutor é enrolado em torno do núcleo. Transformadores de núcleo envolvente são transformadores nos quais o núcleo se fecha por fora dos enrolamentos condutores. A figura 1 também demonstra os dois tipos de enrolamento usuais. Enrolamentos tipo sanduíche são enrolados um em torno do outro como camadas ou cascas cilíndricas. Enrolamentos do tipo panqueca são enrolados em formato de disco e empilhados em torno do núcleo.

Figura 1 – Desenho de transformadores.
(a) transformador de núcleo envolvido com enrolamentos do tipo sanduíche.
(b) transformador de núcleo envolvente com enrolamentos do tipo panqueca.

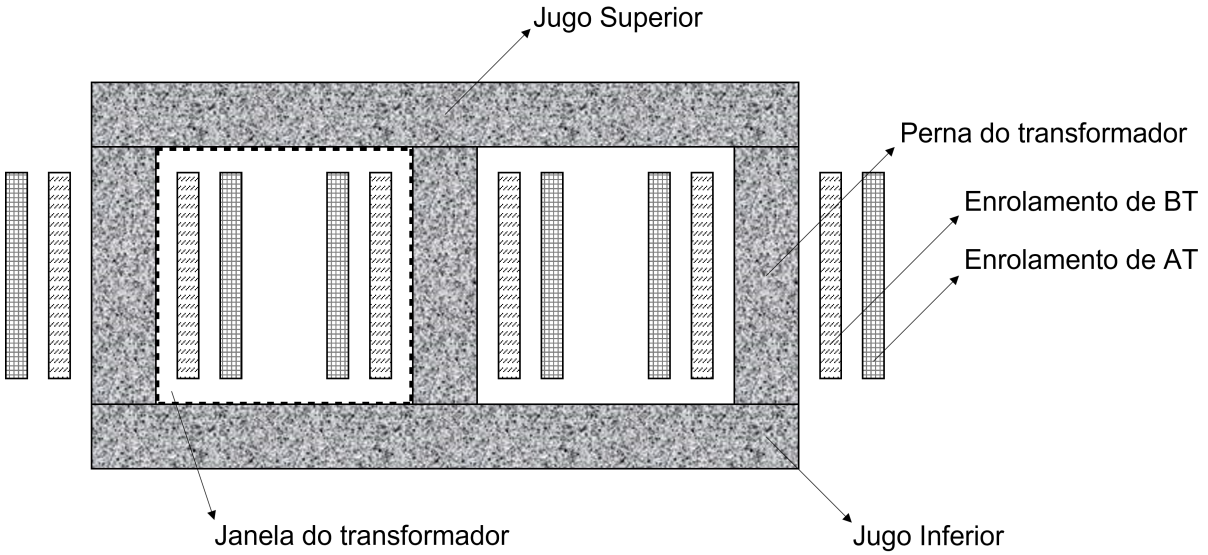



Fonte: Vecchio et al. (2002).

Antes de dar seguimento ao trabalho, é necessário comentar sobre a nomenclatura dada aos componentes do transformador. Aos condutores que são enrolados em torno do núcleo é dado o nome de enrolamento ou bobina. Transformadores elevadores e rebaixadores têm enrolamentos de alta tensão (AT) e baixa tensão (BT) fazendo referência aos níveis de potencial a que estão submetidos. Em geral, os enrolamentos sanduíche de baixa tensão são construídos mais próximos ao núcleo e os de alta tensão mais afastados dele. Às massas de ferro em que os condutores estão enrolados, são dados os nomes de pernas do transformador. Em núcleos envolventes, também são pernas as colunas de ferro paralelas às pernas envolvidas pelos enrolamentos. Ao restante

da massa de ferro é dado o nome de jugo. Jugo superior é a barra que liga as pontas superiores das pernas e jugo inferior é a barra que liga as pontas inferiores. Por fim, dá-se o nome de janela do transformador aos retângulos cujos lados são formados por duas pernas e os dois jugos. Os componentes do transformador estão identificados na vista em corte da figura 2.

Figura 2 – Vista em corte do transformado com seus principais componentes identificados.




Fonte: O autor (2021).

A reatância de dispersão, reatância série ou reatância de curto de um transformador é um parâmetro cujo cálculo preciso é extremamente importante para a modelagem dinâmica dos sistemas em que o transformador se encontra. Por estar em série com a carga, o seu valor é determinante na modelagem de transitórios em sistemas de potência (REYNDERS, 2003) e no cálculo de sobretensão causadas por descontinuidades de corrente em cargas não lineares (ZAKAREVICIUS, 1977).

O valor medido de reatância de dispersão de um transformador de dois enrolamentos não deve exceder uma tolerância de ±7,5% ou ±10% do valor definido na especificação técnica, dependendo se for superior ou inferior a 0,025 por unidade [pu-ohms] (IEEE. . . , 2010). Por consequência, o objetivo de um modelo analítico que pode ser utilizado na indústria é resultar em uma precisão melhor que ±7,5%.

Métodos numéricos como o método de elementos finitos são utilizados para cálculo de reatâncias, porém é necessário um tempo de processamento muito superior ao tempo de processamento de uma formulação analítica. Por este motivo, modelos analíticos são amplamente utilizados em projetos de transformadores.

O engenheiro eletricista formado na Universidade Federal de Santa Catarina

(UFSC) Geovane Romeu Ribeiro (RIBEIRO, 2014) abordou em seu trabalho de conclusão de curso (TCC) o método de Rabins para o cálculo analítico de reatâncias de dispersão de transformadores. Seu TCC teve como foco o estudo da dedução do método de Rabins e apresenta como resultados finais a aplicação do método para uma geometria com enrolamentos do tipo sanduíche de mesma altura variando a altura dos enrolamentos e da janela.

Dando sequência aos estudos realizados na UFSC, este TCC tem como objetivo a escrita de uma biblioteca computacional que permita ao usuário executar o cálculo da reatância de dispersão aplicando os métodos analíticos de Roth e Rabins a partir da entrada de uma geometria arbitrária de transformador, bem como confeccionar gráficos do tipo mapa da densidade de corrente, do vetor potencial magnético e da indução magnética no domínio da janela. Também é um objetivo deste trabalho apresentar as limitações dos métodos analíticos implementados através da análise quantitativa do desvio do cálculo da indutância e qualitativas em desvios de configurações de campos magnéticos ao comparar as soluções analíticas com simulações de elementos finitos. A biblioteca permitirá que o usuário realize o cálculo da indutância, também, por elementos finitos através da programação de uma interface com o *software Femm*. Todos os objetivos do trabalho foram cumpridos, a ferramenta computacional e os métodos analíticos de Roth e Rabins ofereceram erros relativos de 0,36% e 1,93% quando comparados a ensaios realizados em um transformador com reatância medida de 6,16%, a exatidão de ambos os métodos para o caso estudado apresenta melhoria superior a cinco vezes a exatidão prevista por norma.

A ferramenta computacional será desenvolvida na linguagem de programação *python* com a qual o autor possui afinidade devido a desenvolvimento de projetos anteriores durante o curso de disciplinas da graduação. As bibliotecas de *python* a serem utilizadas são a biblioteca *numpy* para manipulação de matrizes e cálculos matemáticos, a biblioteca *scipy* utilizada para computar as funções de Bessel e Struve durante a resolução do método de Rabins, a biblioteca *pyfemm* utilizada para integrar o código em *python* com o programa de elementos finitos *Femm* e a biblioteca *matplotlib* será utilizada para programar a plotagem dos campos de interesse.

O trabalho é dividido em cinco capítulos e um apêndice. O primeiro capítulo fornece o objetivo de estudo dentro do ponto de vista do autor, citando os trabalhos anteriores, problemas, justificativas, comentando sobre o os objetivos pretendidos e citando resultados. O segundo capítulo revisa a bibliografia estudada para realizar o trabalho, discorrendo sobre a teoria do eletromagnetismo e pelo histórico dos métodos desenvolvidos para cálculo analítico da reatância de dispersão de transformadores. O terceiro capítulo explica como um usuário deve fazer uso da ferramenta computacional desenvolvida, bem como detalha o código escrito mostrando as escolhas feitas e soluções

para os problemas encontrados durante a sua implementação. O quarto capítulo tem como objetivo testar os modelos implementados, avaliando-os frente a comparações com simulações de elementos finitos. O quinto e último capítulo é uma conclusão do trabalho desenvolvido, descreve o cumprimento dos objetivos e propõe possíveis temas para trabalhos futuros. O Apêndice A apresenta o código fonte da ferramenta desenvolvida.

# 2 REVISÃO BIBLIOGRÁFICA

## 2.1 ELETROMAGNETISMO

O eletromagnetismo é o ramo da física que estuda a relação entre grandezas elétricas e magnéticas. Essa relação foi primeiro observada pelo físico Hans Christian Ørsted que percebeu uma influência da corrente elétrica na deflexão da agulha de uma bússola. A observação levou a um grande desenvolvimento da teoria eletromagnética a partir do século XIX. Quando as contribuições de André-Marie Ampère, Michael Faraday, James Clerk, Oliver Heaviside e Heinrich Hertz deram forma ao que hoje se conhece como teoria eletromagnética clássica.

### 2.1.1 Equações de Maxwell

As equações de Maxwell são formadas pela lei de Gauss, lei da conservação do fluxo magnético, lei de Faraday e lei de Ampère, apresentadas nas equações (2.1), (2.2), (2.3), (2.4). Publicadas por James Maxwell (1873), as equações são a fundação do eletromagnetismo clássico. Descrevendo, em uma escala macroscópica, as relações entre campos elétricos, magnéticos e cargas em função do tempo e posição, tem-se.

$$\nabla \cdot \mathbf{D} = \rho \tag{2.1}$$

$$\nabla \cdot \mathbf{B} = 0 \tag{2.2}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{2.3}$$

$$\nabla \times \mathbf{H} = \mathbf{J} + \frac{\partial \mathbf{D}}{\partial t} \tag{2.4}$$

As equações são complementadas pelas relações constitutivas do magnetismo (2.5), do campo elétrico (2.6) e pela lei de Ohm (2.7). Em diversas deduções, também, é necessária a introdução do conceito de potencial vetor magnético $\mathbf{A}$, apresentado em (2.8).

$$\mathbf{B} = \mu\mathbf{H} + \mathbf{M} \tag{2.5}$$

$$\mathbf{D} = \epsilon\mathbf{E} + \mathbf{P} \tag{2.6}$$

$$\mathbf{J} = \sigma\mathbf{E} \tag{2.7}$$

$$\nabla \times \mathbf{A} = \mathbf{B} \tag{2.8}$$

Durante operação normal, o transformador em estudo está sujeito a frequências baixas, de forma que despreza-se o termo do deslocamento na lei de Ampère. O que resulta em (2.9).

$$\nabla \times \mathbf{H} = \mathbf{J} \tag{2.9}$$

Substituindo (2.5), (2.8) em (2.9) chega-se à lei de Ampère magnetostática em termos do potencial vetor magnético (2.10) que toma a forma da equação de Poisson.

$$\nabla^2 \mathbf{A} = \mu \mathbf{J} \tag{2.10}$$

### 2.1.2 Indutância

O indutor é o componente em circuitos elétricos que se opõe à variação da corrente que o percorre. Esta oposição se dá através da indução de uma tensão que contribui para a manutenção da corrente. À capacidade do indutor de se opor a esta variação se dá o nome de indutância que é uma grandeza física medida em Henry. A unidade é definida como a indutância necessária para causar a indução de um volt entre os terminais de um indutor sob a variação de corrente de um ampere por segundo. Seu nome foi dado em homenagem ao físico Joseph Henry.

Para indutores inseridos em circuitos elétricos, reescreve-se a lei de indução de Faraday como (2.11). Onde $v(t)$ é a tensão em volts sob os terminais do componente, $i(t)$ a corrente em amperes sob a qual ele está submetido e $L$ é a sua indutância. A indução elétrica é um processo conservativo e um componente puramente indutivo não dissipa potência. A potência instantânea $p(t)$ (2.12) de um indutor é, então, interpretada como a variação da energia magnética $W_m$ (2.13) armazenada por ele.

$$v(t) = L\frac{di(t)}{dt} \tag{2.11}$$

$$p(t) = v(t) \cdot i(t) = Li(t)\frac{di(t)}{dt} \tag{2.12}$$

$$W_m(t) = \int_0^t p(\tau)\, d\tau = \frac{Li^2(t)}{2} \tag{2.13}$$

A partir de (2.11) calcula-se a tensão entre os terminais de um indutor percorrido por uma corrente senoidal pura de amplitude $I$, frequência $\omega$ e fase $\phi$ resultando em (2.14). Por este motivo, define-se a reatância indutiva $X_L[\Omega]$ (2.15) como um análogo na lei de indução do papel da resistência na lei de Ohm.

$$v(t) = \omega\, L \cdot I\, sin\left(\omega\, t + \frac{\pi}{2} + \phi\right) \tag{2.14}$$

$$X_L = \omega\, L \tag{2.15}$$

A energia armazenada no campo magnético é, portanto uma ferramenta importante no cálculo de indutâncias. A densidade de energia magnética é calculada através da equação (2.16). Para o cálculo da energia armazenada em um volume deve-se integrar esta densidade no espaço, resultando em (2.17).

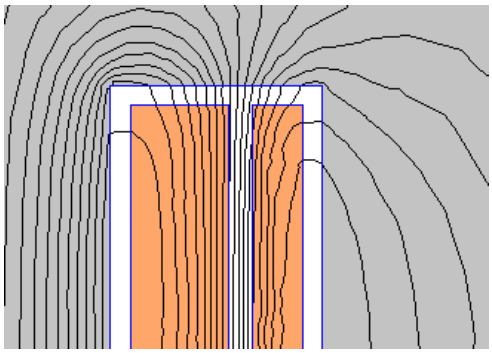
$$w = \frac{\mathbf{B} \cdot \mathbf{H}}{2} \tag{2.16}$$

$$W_m = \frac{1}{2} \int_{volume} \mathbf{B} \cdot \mathbf{H} \, dV \tag{2.17}$$

## 2.2 REATÂNCIA DE DISPERSÃO EM TRANSFORMADORES

Em um transformador, a maior parte das linhas de campo magnético seguem uma trajetória de alta permeabilidade. Porém parte do campo gerado pelos enrolamentos segue trajetórias que passam por meios não ferromagnéticos. A esta fração do campo magnético é dado o nome de campo disperso. A reatância de dispersão está relacionada ao campo disperso que permeia os enrolamentos, conforme mostrado na figura 3.

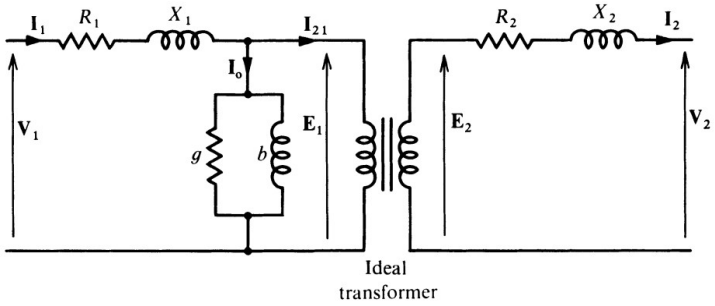
Figura 3 – Desenho das linhas de campo disperso de um transformador. Somente a metade superior da janela é mostrada. Em alaranjado o material condutor dos enrolamentos, em cinza o núcleo ferromagnético.



Fonte: O autor (2021).

A modelagem de um transformador em regime permanente é feita classicamente através do circuito equivalente da figura 4. No circuito equivalente, as reatâncias $X_1$ e $X_2$ são as reatâncias de dispersão. Também são denominadas reatâncias série ou reatâncias de curto-circuito, respectivamente por aparecerem no modelo em série com a carga e representarem aproximadamente a reatância do transformador se o enrolamento secundário estiver em curto. A indutância de dispersão se relaciona com a reatância de dispersão conforme a equação (2.15). O valor da indutância de dispersão é determinante na modelagem transitória de sistemas de potência e de circuitos eletrônicos, permitindo o cálculo da resposta desses sistemas a curto-circuitos e chaveamentos.

Figura 4 – Circuito equivalente do transformador.




Fonte: Daniels (1976).

### 2.2.1 Método de Kapp

Com o objetivo de desenvolver um dos primeiros modelos analíticos de indutância de dispersão, Kapp (1900) aproxima as janelas de transformador por problemas unidimensionais. Esta suposição, para enrolamentos tipo sanduíche (figura 5), não considera a componente radial dos campos e sua variação em $y$. Para enrolamentos tipo panqueca (figura 6), o campo é tomado como somente radial e não varia radialmente. A corrente que gera o campo é pressuposta contínua, desconsiderando o aparecimento de correntes parasitas nos condutores. A última aproximação não gera erro expressivo em baixas frequências.

O método resolve a lei de Ampère na forma integral, equação (2.18). Escolhendo uma linha amperiana que varre os enrolamentos e se fecha no núcleo, a dedução de Kapp chega à equação (2.19) para a soma das indutâncias dos enrolamentos primário e secundário $S_1 + S_2$, onde $K$ é um fator multiplicativo experimental, $L$ é o comprimento médio dos enrolamentos e $l$ é a altura do sanduíche ou largura da panqueca.

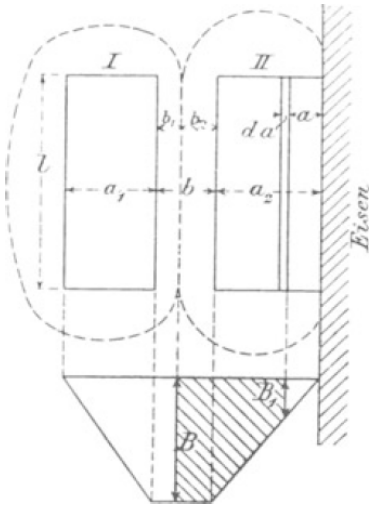
$$\oint_C \mathbf{H} \cdot \mathbf{dl} = \int \int \mathbf{J} \cdot \mathbf{dS} \tag{2.18}$$

$$S_1 + S_2 = K \frac{2\pi}{l} \left( b + \frac{a_1 + a_2}{6} \right) n^2 L \tag{2.19}$$

Experimentalmente, verifica-se que para o disco de enrolamento mais próximo ao ferro, $K$ deve assumir valor de $0,1$. Para cada outro disco, deve assumir $0,05$. Dado um transformador com $u$ discos, o fator K do enrolamento completo pode ser escrito através da equação (2.20)

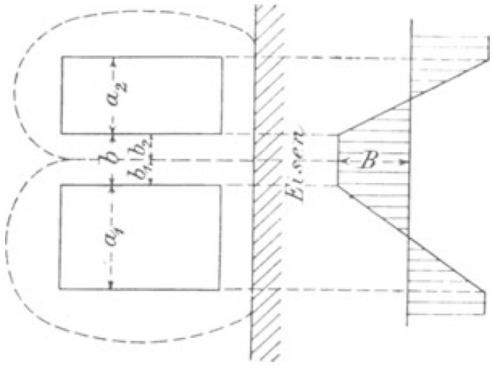
$$K = 0,05 \frac{u+1}{u} \tag{2.20}$$

Figura 5 – Janela do transformador com enrolamentos em camadas.




Fonte: Kapp (1900).

Figura 6 – Janela do transformador com enrolamentos tipo panqueca.

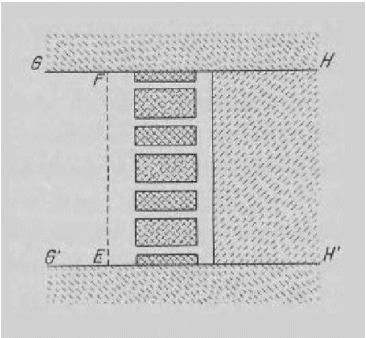



Fonte: Kapp (1900).

### 2.2.2 Método de Rogowski

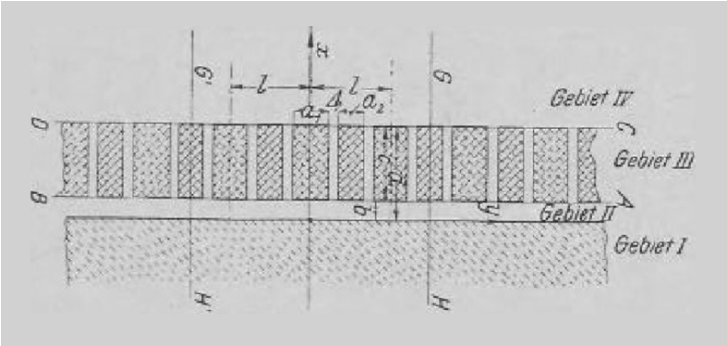
O método discutido nesta seção teve como motivação remover a componente experimental do método de Kapp (1900). O método de Rogowski (1908) divide a janela mostrada na figura 7 em quatro regiões uniformes conforme a figura 8. O desenho repete periodicamente as fontes de corrente na janela com o intuito de, na sequência, representá-las através de uma série de Fourier. Na figura 8, a Região I é composta por material magnético de permeabilidade relativa $\mu_r$, as Regiões II e IV são permeadas por ar e a Região III é composta por discos de enrolamento primário de altura $a_1$, intercalados com discos de ar de altura $\Delta$ e discos de enrolamento secundário de altura $a_2$.

Figura 7 – Janela do transformador.




Fonte: Rogowski (1908).

Figura 8 – Janela do transformador dividida em regiões.




Fonte: Rogowski (1908).

O modelo resolve o campo magnético na janela em um plano $xy$. O condutor é tomado como infinito em $z$. Desta forma, as equações de Maxwell são resolvidas em coordenadas retangulares. A corrente que gera o campo é tomada como contínua. Na janela do transformador (figura 7), consideram-se preenchidos por ferro, os espaços entre o topo do enrolamento e o jugo superior e entre a base do enrolamento e o jugo inferior são considerados preenchidos por ferro.

O autor resolve a lei de conservação do fluxo magnético e a lei de Ampère em cada uma das regiões, acoplando-as nas fronteiras. Ao estabelecer uma formulação prática, o método assume que:

$$\begin{aligned} 1 < \frac{a_1 + a_2 + 2\Delta}{a_1 + a_2} < 2 \\ kc = \frac{2\pi c}{a_1 + 2\Delta + a_2} > 1,5 \end{aligned} \tag{2.21}$$

Onde $c$ denota a espessura dos enrolamentos. Após a resolução, o método retorna a equação (2.22) para as indutâncias, onde $q$ é o número de discos. O fato que surpreende até mesmo o autor do trabalho é que a fórmula é igual à equação (2.19) de Kapp sendo

descoberto um equacionamento teórico para o fator $K$, equação (2.23). Por fim, o autor argumenta que a exponencial de $-2\,b\,k$ é muito próxima de zero em casos práticos, por $b$ assumir um valor mínimo da distância necessária para isolar os condutores do núcleo. O fator de Rogowski $K_R$ é então definido como o fator $K$ desconsiderando a influência do núcleo ferromagnético. A equação (2.24) define o fator.

$$S = K\frac{2\pi n^2}{c}Lq\left(\Delta + \frac{a_1 + a_2}{6}\right) \tag{2.22}$$

$$K = 1 - \frac{1 - e^{-kc}}{kc}\left(1 - \frac{1}{2}\frac{\mu_r - 1}{\mu_r + 1}e^{-2kb}\left(1 - e^{-kc}\right)\right) \tag{2.23}$$

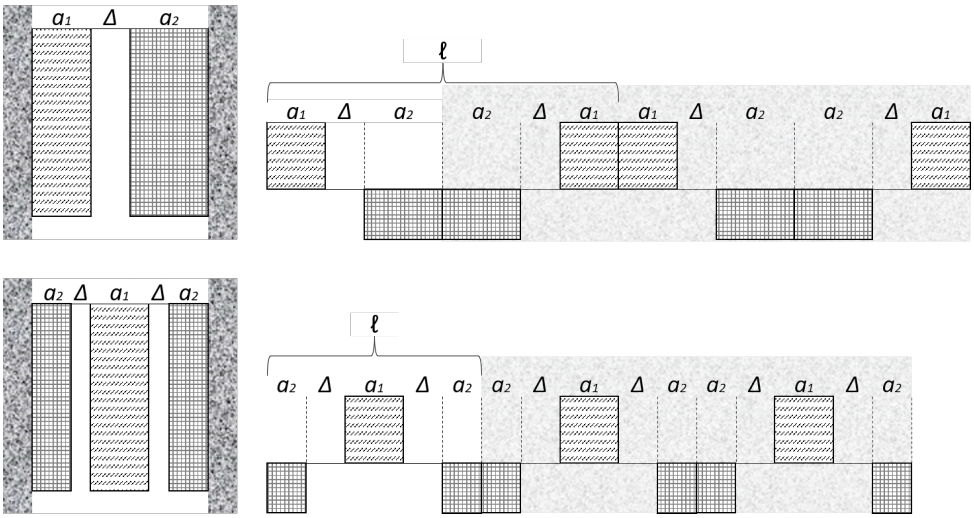
$$K_R = 1 - \frac{1 - e^{-kc}}{kc} \tag{2.24}$$

Onde:

$$k = \frac{2\pi}{l} \tag{2.25}$$

E $l$ representa a unidade de periodicidade da corrente quando refletida infinitas vezes no núcleo. Observa-se na figura 9 exemplos do período $l$ para duas diferentes configurações de enrolamentos.

Figura 9 – Ilustração do período $l$ do fator de Rogowski.




Fonte: O autor (2021).

### 2.2.3 Método de Roth

Em vez da divisão do problema em regiões, Roth (1927) propõe uma descrição da densidade de corrente na janela do transformador através da série dupla de Fourier conforme a equação (2.28). Um breve desenvolvimento do método está disponível em Boyajian (1954). Anteriormente ao método, para que se facilite o cálculo dos coeficientes $J_{mn}$, é proposto um mapeamento das coordenadas da janela a um quadrado de dimensões $\pi \times \pi$. Este mapeamento é feito através das mudanças de variável das equações theta$_x$e$theta_y, onde$x$_0$ e $y_0$ são as dimensões da janela.

$$\theta_x = \pi \frac{x}{x_0} \tag{2.26}$$

$$\theta_y = \pi \frac{y}{y_0} \tag{2.27}$$

$$J\left(x,y\right) = \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} J_{mn} cos\left(m\theta_x\right) cos\left(n\theta_y\right) \tag{2.28}$$

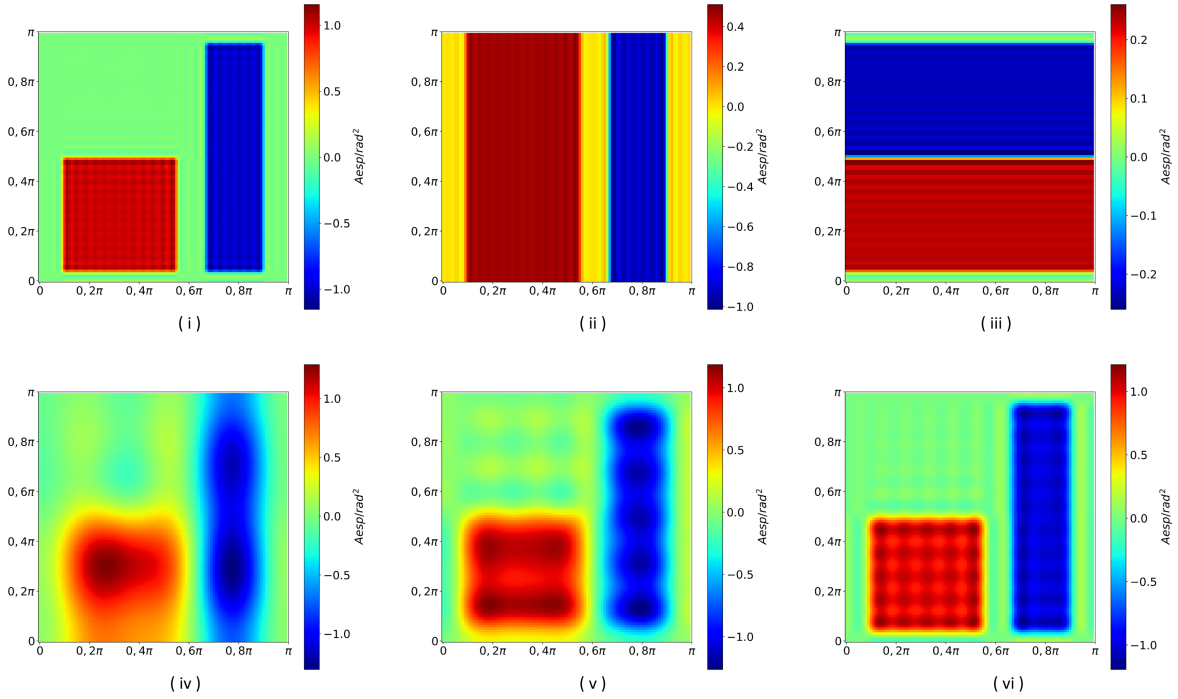
Em um transformador ou reator com $k$ blocos retangulares de condutores, calcula-se $J_{mn}$ através da equação (2.29), onde $x_1^{(i)}$ e $y_1^{(i)}$ são as coordenadas do vértice inferior esquerdo e $x_2^{(i)}$ e $y_2^{(i)}$ as do superior direito do i-ésimo bloco retangular. $J^{(i)}$ é a densidade de corrente do i-ésimo bloco $[A \cdot esp/rad^2]$ após o mapeamento da janela. Para transformadores, espera-se que a componente independente de cossenos $J_{00}$ seja nula, pois se parte de um princípio de equilíbrio de Ampère-espiras nos enrolamentos. A figura 10 plota a função $J(x,y)$ truncada em diferentes valores de $m$ e $n$.

$$J_{mn} = \begin{cases} \frac{1}{\pi^2} \sum_{i=1}^{k} J^{(i)} \left(\theta_{x_2}^{(i)} - \theta_{x_1}^{(i)}\right) \left(\theta_{y_2}^{(i)} - \theta_{y_1}^{(i)}\right) & \begin{matrix} m=0 \\ n=0 \end{matrix} \\ \frac{2}{n\pi^2} \sum_{i=1}^{k} J^{(i)} \left(\theta_{x_2}^{(i)} - \theta_{x_1}^{(i)}\right) \left(\sin n\theta_{y_2}^{(i)} - \sin n\theta_{y_1}^{(i)}\right) & \begin{matrix} m=0 \\ n=1,2,... \end{matrix} \\ \frac{2}{m\pi^2} \sum_{i=1}^{k} J^{(i)} \left(\sin m\theta_{x_2}^{(i)} - \sin m\theta_{x_2}^{(i)}\right) \left(\theta_{y_2}^{(i)} - \theta_{y_1}^{(i)}\right) & \begin{matrix} m=1,2,... \\ n=0 \end{matrix} \\ \frac{4}{mn\pi^2} \sum_{i=1}^{k} J^{(i)} \left(\sin m\theta_{x_2}^{(i)} - \sin m\theta_{x_1}^{(i)}\right) \left(\sin n\theta_{y_2}^{(i)} - \sin n\theta_{y_1}^{(i)}\right) & \begin{matrix} m=1,2,... \\ n=1,2,... \end{matrix} \end{cases} \tag{2.29}$$

A resolução da equação de Poisson (2.10) é feita novamente em coordenadas retangulares resultando no potencial magnético a seguir.

$$A(x,y) = \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} A_{mn} \cos\left(m\theta_x\right) \cos\left(n\theta_y\right) \tag{2.30}$$

Figura 10 – Ilustração da dupla série de Fourier em um enrolamento de transformador. As figuras se referem aos seguintes truncamentos. i: $n = 50$ e $m = 50$; ii: $n = 0$ e $m = 50$; iii: $n = 50$ e $m = 0$; iv: $n = 5$ e $m = 5$; v: $n = 10$ e $m = 10$; vi: $n = 20$ e $m = 20$.




Fonte: O autor (2021).

Onde:

$$A_{mn} = \mu_0 \frac{J_{mn} x_0 y_0}{m^2 y_0^2 + n^2 x_0^2} \tag{2.31}$$

Esta proposta permite através da equação (2.32) que se calcule a indutância de um transformador ou reator.

$$L = \frac{\pi^2 p}{I^2} \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \frac{J_{mn}\, A_{mn}}{h} \tag{2.32}$$

Onde $I$ é a corrente nominal do equipamento, $p$ é o comprimento médio dos enrolamentos e $h$ está definido a seguir:

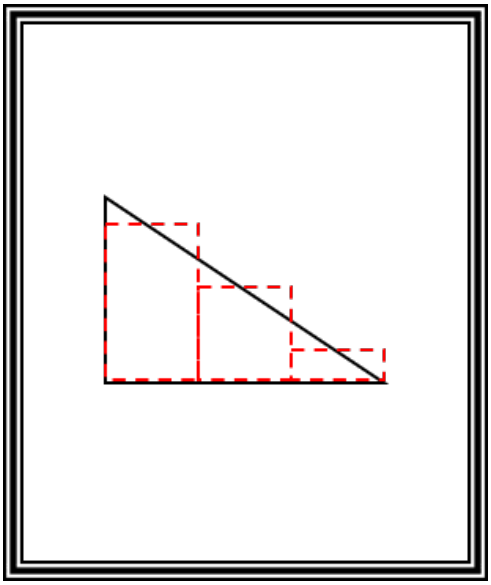
$$\begin{aligned} h &= 1, \quad m = 0 \quad n = 0 \\ h &= 2, \quad m = 0 \quad n = 1, 2, \ldots \\ h &= 2, \quad m = 1, 2, \ldots \quad n = 0 \\ h &= 4, \quad m = 1, 2, \ldots \quad n = 1, 2, \ldots \end{aligned} \tag{2.33}$$

O método também permite o cálculo da indução magnética $\mathbf{B}(x,y)$. Conforme a equação (2.8), basta calcular o rotacional do potencial vetor magnético em (2.30) para que se chegue à equação da indução magnética (2.34).

$$\mathbf{B}(x,y) = \sum_{m=0}^{\infty}\sum_{n=0}^{\infty} A_{mn}\left(-\frac{n\,\pi}{y_0}\,cos\left(m\theta_x\right)\,sin\left(n\theta_y\right)\,\hat{\imath} + \frac{m\,\pi}{x_0}\,sin\left(m\theta_x\right)\,cos\left(n\theta_y\right)\,\hat{\jmath}\right) \tag{2.34}$$

A vantagem do método é que possibilita o projeto de uma geometria genérica composta por $k$ blocos condutores de seção retangular. Diferentemente do método de Rogowski que foi desenvolvido para que a corrente esteja compreendida entre um par de segmentos de reta paralelos. Até mesmo no caso da necessidade de condutores não retangulares, é possível aproximá-los pela soma de retângulos como exemplificado na figura 11.

Figura 11 – Ilustração da aproximação de um enrolamento triangular por retângulos.



Fonte: O autor (2021).

### 2.2.4 Método de Rabins

Os métodos de Rogowski e Roth possuem alta acurácia nas configurações usuais de bobinas. Normalmente, os enrolamentos possuem alturas semelhantes, ficam próximos e estão posicionados a uma mesma altura na janela. A precisão dos métodos diminui quanto mais a geometria do transformador se afasta do usual. Se houver necessidade de contemplar transformadores de três enrolamentos ou transformadores de tap variável, muitas vezes, será necessário que a resolução analítica se dê em coordenadas cilíndricas. A resolução das equações de Maxwell em coordenadas cilíndricas liberta o

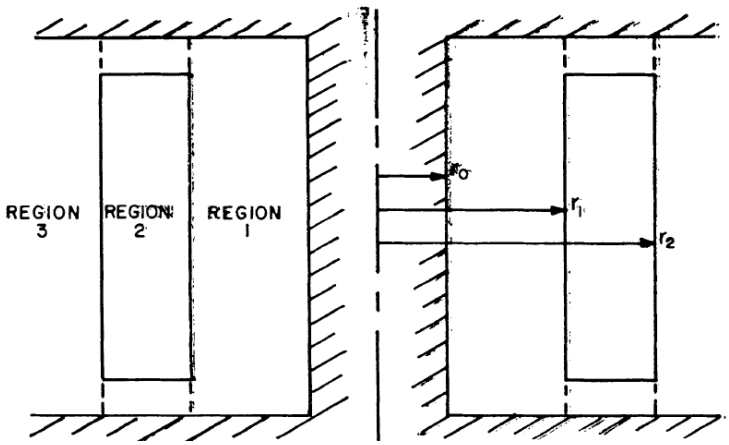
modelo da suposição de condutor infinito em profundidade e, então, passa-se a resolver o problema como de natureza axissimétrica.

Roth aplicou a mesma série de Fourier dupla ao problema em coordenadas cilíndricas, resultando na necessidade da solução de funções transcendentes para a resolução do problema de valor de contorno. Rabins (1956) retorna a um ponto de partida mais simples, assim como Rogowski dividindo a janela do transformador em regiões e utilizando a série simples de Fourier para cada região. A equação de Poisson em coordenadas cilíndricas pode ser escrita como (2.35) no caso em que a corrente tem direção unicamente azimutal. Rabins considera em sua dedução que à direita da janela, se afastando da perna do transformador, os jugos superior e inferior se estendem ao infinito, assim como a largura da própria janela de ar.

$$\frac{\partial^2 \mathbf{A}}{\partial r^2} + \frac{1}{r}\frac{\partial \mathbf{A}}{\partial r} - \frac{\mathbf{A}}{r^2} + \frac{\partial^2 \mathbf{A}}{\partial z^2} = \mu \mathbf{J} \tag{2.35}$$

Rabins calcula a contribuição de cada enrolamento para o potencial vetor magnético, através da geometria da figura 12, onde as regiões 1 e 3 são permeadas por ar e a região 2 contém o enrolamento estudado. A densidade de corrente da região 2 pode ser escrita através da série de Fourier da equação (2.36).

Figura 12 – Regiões do método de Rabins.




Fonte: Rabins (1956).

$$J(z) = J_0 + \sum_{n=0}^{\infty} J_n \cos(mz) \tag{2.36}$$

Onde $L$ é a altura da janela do transformador e $m$ depende de $n$ conforme a equação (2.37). Durante a solução da equação diferencial parcial, a mudança de variável de $r$ para $x$ em (2.38) se faz útil. Através desta mudança, a equação de Poisson toma a forma da equação de Bessel modificada cujo resultado é conhecido. Após garantir a satisfação das condições de contorno refração da indução entre o ferro e o ar, a

continuidade do potencial vetor nas interfaces regiões diferentes dentro da janela e campo nulo em raios infinitos, Rabins chega ao equacionamento (2.39) do potencial vetor magnético para cada uma das regiões da figura 12.

$$m = \frac{n\,\pi}{L} \tag{2.37}$$

$$x = m\,r \tag{2.38}$$

$$\begin{aligned}
A^I &= \tfrac{\mu_0 J_0 (r_2 - r_1)}{2} r + \mu_0 \textstyle\sum_{n=1}^{\infty} \tfrac{J_n}{m^2} \left[C_n I_1(x) + D_n K_1(x)\right] cos\,(mz) \\
A^{II} &= \mu_0 J_0 \left(\tfrac{r_2 r}{2} - \tfrac{r_1^3}{6r} - \tfrac{r^2}{3}\right) + \mu_0 \textstyle\sum_{n=1}^{\infty} \tfrac{J_n}{m^2} \left[E_n I_1(x) + F_n K_1(x) - \tfrac{\pi}{2} L_1(x)\right] cos\,(mz) \\
A^{III} &= \tfrac{\mu_0 J_0 \left(r_2^3 - r_1^3\right)}{6r} + \mu_0 \textstyle\sum_{n=1}^{\infty} \tfrac{J_n}{m^2} G_n K_1(x)\, cos\,(mz)
\end{aligned} \tag{2.39}$$

Onde $I_n(x)$ e $K_n(x)$ são funções modificadas de Bessel do primeiro e segundo tipo e n-ésima ordem. $L_n(x)$ é a função modificada de Struve de n-ésima ordem e as constantes $C_n$, $D_n$, $E_n$, $F_n$ e $G_n$ podem ser calculadas conforme a equação (2.40).

$$\begin{aligned}
C_n &= \textstyle\int_{x_1}^{x_2} t\,K_1(t)\,dt\,, & D_n &= \tfrac{I_0(x_c)}{K_0(x_c)} C_n \\
E_n &= \textstyle\int_{0}^{x_2} t\,K_1(t)\,dt\,, & F_n &= D_n - \textstyle\int_{0}^{x_1} t\,I_1(t)\,dt \\
G_n &= D_n + \textstyle\int_{x_1}^{x_2} t\,I_1(t)\,dt
\end{aligned} \tag{2.40}$$

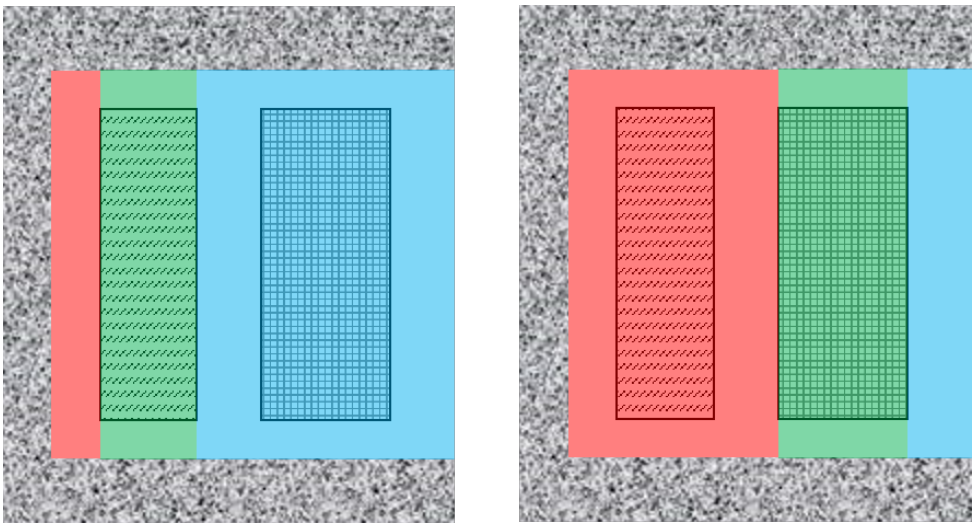
Onde $x_1 = m\,r_1$, $x_2 = m\,r_2$, $x_c = m\,r_c$ e $r_c$ é o raio do núcleo do transformador.

Para o estudo do transformador, calcula-se então o potencial vetor magnético que cada um dos enrolamentos gera. As contribuições são sobrepostas levando em conta as posições relativas dos enrolamentos, permitindo o cálculo do potencial vetor magnético **A** em todo o domínio.

Na figura 13 verifica-se que a bobina de alta tensão está na Região III da divisão do domínio para a solução do potencial vetor magnético gerado pela corrente do enrolamento de baixa tensão. A bobina de BT se encontra na Região I durante a análise do enrolamento de AT e cada enrolamento está localizado na região II durante a análise do próprio potencial magnético.

De maneira genérica, cada enrolamento está permeado por um somatório dos potenciais vetor magnético do tipo $A^{III}$ gerados por enrolamentos mais internos que ele com o de tipo $A^{II}$ que ele mesmo gera e os de tipo $A^I$ gerados pelos enrolamentos externos a ele.

Figura 13 – Regiões vistas em cada enrolamento no método de Rabins. À esquerda a divisão do domínio na análise do enrolamento de baixa tensão, à direita as regiões na análise do enrolamento de alta tensão. Em vermelho a região I, em verde a II e em azul a III.



Fonte: O autor (2021).

Para chegar ao valor da reatância de dispersão, calcula-se a energia armazenada no campo magnético através da equação (2.41). Com o valor de $W$ calculado, é trivial isolar a indutância na equação (2.13) e calculá-la. O valor de $\mathbf{J}$ é diferente de zero somente no volume dos enrolamentos, permitindo que (2.41) seja reescrita como (2.42) para um transformador de $k$ enrolamentos.

$$W = \frac{1}{2} \int_{volume} \mathbf{A} \cdot \mathbf{J} \, dV \tag{2.41}$$

$$W = \frac{1}{2} \sum_{i=1}^{k} \int_{V_i} \mathbf{A} \cdot \mathbf{J_i} \, dV \tag{2.42}$$

Onde $V_i$ é o volume do i-ésimo enrolamento.

Numerando-se os enrolamentos de 1 a $k$ do mais interno ao mais externo, desmembra-se o potencial vetor magnético nas contribuições de cada fonte de corrente chegando-se à equação (2.43). A notação de $\mathbf{A}_j^{\lambda}$ significa que deve-se utilizar a equação de (2.39) referente à região $\lambda$ no cálculo da contribuição da j-ésima fonte de corrente para o potencial vetor magnético no domínio.

$$W = \frac{1}{2} \sum_{i=1}^{k} \int_{V_i} \left[ \left( \sum_{j=1}^{i-1} \mathbf{A}_j^{III} \right) + \mathbf{A}_i^{II} + \left( \sum_{j=i+1}^{k} \mathbf{A}_j^{I} \right) \right] \cdot \mathbf{J_i} \, dV \tag{2.43}$$

Expandindo o termo entre colchetes, tem-se:

$$\begin{aligned}\sum_{j=1}^{i-1}\mathbf{A}_j^{III}+\mathbf{A}_i^{II}+\sum_{j=i+1}^{k}\mathbf{A}_j^{I} = &-\mu_0\frac{J_0^{(i)}}{3}r^2+\mu_0\left(\frac{J_0^{(i)}\,r_e^{(i)}}{2}+\sum_{j=i+1}^{k}\frac{J_0^{(j)}\left(r_e^{(j)}-r_i^{(j)}\right)}{2}\right)r\\ &+\mu_0\left(\sum_{j=1}^{i-1}\frac{J_0^{(j)}\left(r_e^{3(j)}-r_i^{3(j)}\right)}{6}-\frac{J_0^{(i)}r_i^{3(i)}}{6}\right)\frac{1}{r}\\ &+\mu_0\sum_{n=1}^{\infty}\frac{1}{m^2}\left[\left(J_n^{(i)}\,E_n^{(i)}+\sum_{j=i+1}^{k}J_n^{(j)}\,C_n^{(j)}\right)I_1(x)\right.\\ &+\left(\sum_{j=1}^{i-1}J_n^{(j)}\,G_n^{(j)}+J_n^{(i)}\,F_n^{(i)}+\sum_{j=i+1}^{k}J_n^{(j)}\,D_n^{(j)}\right)K_1(x)\\ &\left.-\frac{\pi}{2}J_n^{(i)}\,L_1(x)\right]cos\,(m\,z)\end{aligned} \tag{2.44}$$

Onde os subscritos se referem à componente harmônica da série de Fourier, os sobrescritos se referem ao número do enrolamento e os parâmetros $r_i^{(i)}$ e $r_e^{(i)}$ denotam raio interno e externo do i-ésimo enrolamento. Definindo na equação (2.45) as constantes $P$, $Q$, $R$, $S_n$, $T_n$ e $U_n$, reescreve-se a energia magnética conforme a equação (2.46).

$$\begin{aligned}P^{(i)} &= -\mu_0\frac{J_0^{(i)}}{3}\\ Q^{(i)} &= \mu_0\left(\frac{J_0^{(i)}\,r_e^{(i)}}{2}+\sum_{j=i+1}^{k}\frac{J_0^{(j)}\left(r_e^{(j)}-r_i^{(j)}\right)}{2}\right)\\ R^{(i)} &= \mu_0\left(\sum_{j=1}^{i-1}\frac{J_0^{(j)}\left(r_e^{3(j)}-r_i^{3(j)}\right)}{6}-\frac{J_0^{(i)}r_i^{3(i)}}{6}\right)\\ S_n^{(i)} &= \mu_0\frac{1}{m^2}\left(J_n^{(i)}\,E_n^{(i)}+\sum_{j=i+1}^{k}J_n^{(j)}\,C_n^{(j)}\right)\\ T_n^{(i)} &= \mu_0\frac{1}{m^2}\left(\sum_{j=1}^{i-1}J_n^{(j)}\,G_n^{(j)}+J_n^{(i)}\,F_n^{(i)}+\sum_{j=i+1}^{k}J_n^{(j)}\,D_n^{(j)}\right)\\ U_n^{(i)} &= -\mu_0\frac{1}{m^2}\,\frac{\pi}{2}J_n^{(i)}\end{aligned} \tag{2.45}$$

$$\begin{aligned}W = \frac{1}{2}\sum_{i=1}^{k}\int_{V_i}&\left[\left(P^{(i)}\,r^2+Q^{(i)}\,r+R^{(i)}\,\frac{1}{r}\right)+\right.\\ &\left.\sum_{n=1}^{\infty}\left(S_n^{(i)}\,I_1(x)+T_n^{(i)}\,K_1(x)+U_n^{(i)}\,L_1(x)\right)\,cos\,(m\,z)\right]\mathbf{J^{(i)}}\,dV\end{aligned} \tag{2.46}$$

Formulando a integral em coordenadas cilíndricas e substituindo $\mathbf{J_i}$ pela sua expansão em série de Fourier tem-se:

$$W = \frac{1}{2}\sum_{i=1}^{k}\left[J_0^{(i)}\int_0^{2\pi}\int_0^{L}\int_{r_i^{(i)}}^{r_e^{(i)}}\left(P^{(i)}\,r^2 + Q^{(i)}\,r + R^{(i)}\,\frac{1}{r}\right)\,r\,dr\,dz\,d\theta + \sum_{n=1}^{\infty} J_n^{(i)}\int_0^{2\pi}\int_0^{L}\int_{r_i^{(i)}}^{r_e^{(i)}}\left(S_n^{(i)}\,I_1(x) + T_n^{(i)}\,K_1(x) + U_n^{(i)}\,L_1(x)\right)\,cos^2\,(m\,z)\,r\,dr\,dz\,d\theta\right] \tag{2.47}$$

Resolvendo as integrais em $\theta$ e $L$, chega-se a:

$$W = \frac{1}{2}\sum_{i=1}^{k}\left[J_0^{(i)}\,2\,\pi\,L\int_{r_i^{(i)}}^{r_e^{(i)}}\left(P^{(i)}\,r^3 + Q^{(i)}\,r^2 + R^{(i)}\right)\,dr + \pi\,L\sum_{n=1}^{\infty} J_n^{(i)}\int_{r_i^{(i)}}^{r_e^{(i)}}\left(S_n^{(i)}\,r\,I_1(x) + T_n^{(i)}\,r\,K_1(x) + U_n^{(i)}\,r\,L_1(x)\right)\,dr\right] \tag{2.48}$$

Na segunda integral, faz-se a substituição de $r$ por $x$:

$$W = \frac{1}{2}\sum_{i=1}^{k}\left[J_0^{(i)}\,2\,\pi\,L\int_{r_i^{(i)}}^{r_e^{(i)}}\left(P^{(i)}\,r^3 + Q^{(i)}\,r^2 + R^{(i)}\right)\,dr + \pi\,L\sum_{n=1}^{\infty}\frac{J_n^{(i)}}{m^2}\int_{x_i^{(i)}}^{x_e^{(i)}}\left(S_n^{(i)}\,x\,I_1(x) + T_n^{(i)}\,x\,K_1(x) + U_n^{(i)}\,x\,L_1(x)\right)\,dx\right] \tag{2.49}$$

Resolvendo separadamente cada uma das integrais tem-se:

$$\begin{aligned}\int_{r_i^{(i)}}^{r_e^{(i)}} r^3\,dr &= \frac{r_e^{4(i)} - r_i^{4(i)}}{4}\\ \int_{r_i^{(i)}}^{r_e^{(i)}} r^2\,dr &= \frac{r_e^{3(i)} - r_i^{3(i)}}{3}\\ \int_{r_i^{(i)}}^{r_e^{(i)}} dr &= r_e^{(i)} - r_i^{(i)}\end{aligned} \tag{2.50}$$

$$\begin{aligned}
\int_{x_i^{(i)}}^{x_e^{(i)}} x\, I_1(x)\, dx &= \Big( x\, I_0\,(x) \Big) \Big|_{x=x_i^{(i)}}^{x_e^{(i)}} - \int_{x_i^{(i)}}^{x_e^{(i)}} I_0(x)\, dx \\
\int_{x_i^{(i)}}^{x_e^{(i)}} x\, K_1(x)\, dx &= \Big( -x\, K_0\,(x) \Big) \Big|_{x=x_i^{(i)}}^{x_e^{(i)}} + \int_{x_i^{(i)}}^{x_e^{(i)}} K_0(x)\, dx \\
\int_{x_i^{(i)}}^{x_e^{(i)}} x\, L_1(x)\, dx &= \left( x\, L_0\,(x) - \frac{x^2}{\pi} \right) \Big|_{x=x_i^{(i)}}^{x_e^{(i)}} - \int_{x_i^{(i)}}^{x_e^{(i)}} L_0(x)\, dx
\end{aligned} \tag{2.51}$$

Para as funções de Bessel e para a função de Struve, a construção da integração por partes em (2.51) é suficiente para a implementação em *python*, visto que a biblioteca que será utilizada, *Scipy Special*, possui implementações das funções $I_n$, $K_n$ e $L_n$ e suas integrais. O uso dessa biblioteca foi fundamental na implementação do método de Rabins. Devido aos problemas de convergência das funções de Bessel e Struve, a biblioteca seleciona o método mais adequado para computar os valores das funções conforme o valor argumento $x$.

O último passo para calcular o valor da energia na equação (2.49) é escolher um limite em $n$ para truncar o somatório das componentes harmônicas da série de Fourier. Existe um compromisso na escolha do número de componentes, quanto mais harmônicas, maior a precisão da descrição na série de Fourier da corrente. Porém, além do aumento do tempo de computação, o valor de $x$ aumenta com $n$. Quanto maior o argumento $x$ nas integrações em (2.40) e (2.51), maior o erro numérico no seu cálculo. Observou-se que para as geometrias estudadas, a implementação do *scipy* das funções de Bessel e Struve levam o método à divergência caso se utilize argumentos $x$ maiores que 25. Desta maneira, sugere-se que o número de componentes harmônicas utilizado nas simulações não ultrapasse oito vezes a razão entre a altura da janela e o raio externo do enrolamento mais distante do núcleo, conforme a inequação em (2.52).

$$\frac{n\pi}{L}\, r_{max} < 25 \Rightarrow n < \frac{25}{\pi}\, \frac{L}{r_{max}} \simeq 8\, \frac{L}{r_{max}} \tag{2.52}$$

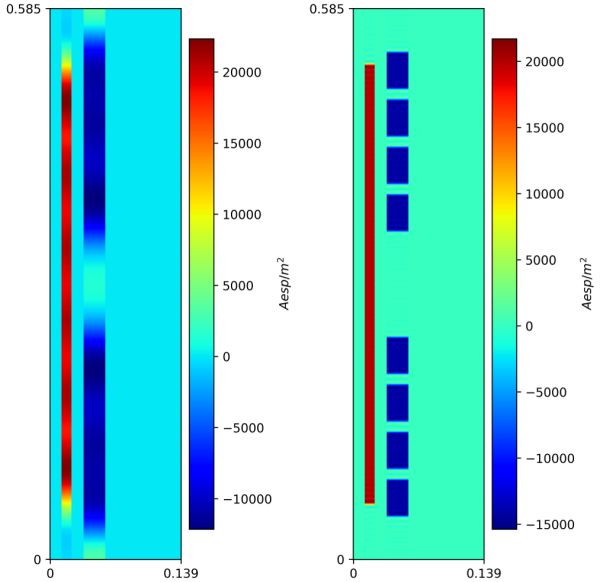
Observa-se na figura 14 uma representação do quanto somente 15 harmônicas afasta a geometria da realidade em comparação a uma representação com mais harmônicas. Porém, na representação da esquerda, a energia magnética do método de Rabins desvia menos de $0,4\%$ da simulação por elementos finitos. Para o caso $n = 200$, o método diverge e o desvio ultrapassa $3 \cdot 10^{31}\%$. A figura 14 ilustra o impacto da adição, além da regra prática, de componentes na série de fourier da corrente no método de Rabins. Pode-se observar um ruído numérico aparecendo no mapa da indução para os valores de raio próximos ao raio externo do último enrolamento.

Tendo calculado a energia magnética, a equação (2.53) deve ser utilizada para o cálculo da indutância de dispersão do transformador, onde $I$ é a corrente no enrolamento de referência. A reatância de dispersão é, então, calculada através da equação (2.15).

$$L = \frac{2\,W}{I^2} \tag{2.53}$$

Definidas as componentes $r$ e $z$ da equação (2.54), o método permite o cálculo da indução magnética através do rotacional em coordenadas cilíndricas de $\mathbf{A}$ conforme as equações (2.55) e (2.56).

Figura 14 – Geometria do transformador recuperada das séries de Fourier das fontes de corrente. À esquerda, a série foi truncada em $n = 15$, à direita, em $n = 200$.



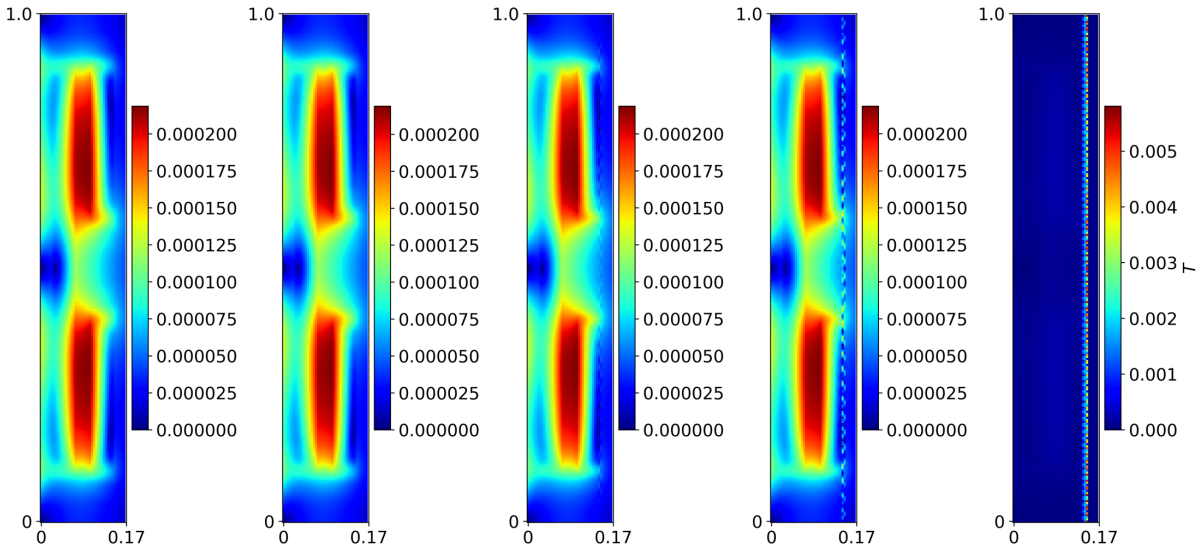

Fonte: O autor (2021).

$$\mathbf{B}(r,z) = B_r\,\hat{\mathbf{r}} + B_z\,\hat{\mathbf{z}} \tag{2.54}$$

$$B_r = \sum_{n=1}^{\infty} m\,\left(S_n^{(i)}\,I_1(x) + T_n^{(i)}\,K_1(x) + U_n^{(i)}\,L_1(x)\right)\,sin\,(m\,z) \tag{2.55}$$

$$\begin{aligned} B_z =& 3\,P^{(i)}\,r \\ &+ 2\,Q^{(i)} \\ &+ \sum_{n=1}^{\infty}\left(S_n^{(i)}\,I_1(x) + S_n^{(i)}\frac{x}{2}\,(I_0(x) + I_2(x))\right. \\ &+ T_n^{(i)}\,K_1(x) - T_n^{(i)}\frac{x}{2}\,(K_0(x) + K_2(x)) \\ &\left.+ U_n^{(i)}\,L_1(x) + U_n^{(i)}\frac{x}{2}\left(L_0(x) + L_2(x) + \frac{2\,x}{3\pi}\right)\right)\,cos\,(m\,z)\,\frac{1}{r} \end{aligned} \tag{2.56}$$

Figura 15 – Mapa da indução magnética exemplificando a divergência das funções de Bessel e Struve com $n = 50, 55, 65, 70\,e\,80$, para este caso, a regra prática sugere que seja utilizado $n < 53$.




Fonte: O autor (2021).

Com este item, finaliza-se a revisão bibliográfica do TCC. O próximo capítulo aborda a ferramenta computacional desenvolvida para aplicação dos métodos estudados, uma documentação das decisões tomadas durante o seu desenvolvimento, um breve tutorial de como utilizá-la, bem como comentários sobre seus recursos, limitações e oportunidades de expansão.

# 3 FERRAMENTA COMPUTACIONAL DESENVOLVIDA

A ferramenta computacional desenvolvida durante este trabalho *transformer* é um módulo em *Python* que permite que o usuário calcule a indutância de dispersão de transformadores. A ferramenta permite que se opte pelos métodos analíticos de Roth e Rabins ou o método de elementos finitos em duas dimensões e em axissimétrico através do *software* livre *Femm* e sua extensão para *Python pyfemm* (MEEKER, 2018). Esta seção do trabalho se destina a comentar sobre a implementação do código e do uso da biblioteca desenvolvida.

## 3.1 COMO UTILIZAR A BIBLIOTECA

### 3.1.1 Dependências

A ferramenta desenvolvida depende da instalação dos módulos *python numpy* (HARRIS et al., 2020), *scipy* (VIRTANEN et al., 2020), *matplotlib* (HUNTER, 2007) e *pyfemm* bem como da instalação do *software* livre de elementos finitos em duas dimensões e em axissimétrico *Femm*. As bibliotecas de *python* utilizadas podem ser adquiridas via o instalador *pip* através dos comandos *"pip install numpy"*, *"pip install scipy"*, *"python -m pip install -U matplotlib"* e *"pip install pyfemm"*. O *software* de elementos finitos *Femm* pode ser facilmente adquirido em repositório *online*.

### 3.1.2 Inicialização

Para importar o módulo *transformer* basta rodar a linha de código abaixo.

```
>>> from myfilename import transformer
```

Onde *myfilename.py* é o arquivo que contém o código descrito no Apêndice A. Em seguida, deve-se alocar uma variável para a classe *transformer*, inicializada através das variáveis de largura e altura da janela $x_0$ e $y_0$ e do raio do núcleo $r_{core}$. O código abaixo instancia *T1* como um transformador cuja janela tem dimensões de 100 $mm$ por 200 $mm$ e cujo núcleo tem raio de 50 $mm$.

```
>>> T1 = transformer(x0=0.1, y0=0.2, r_core=0.05)
```

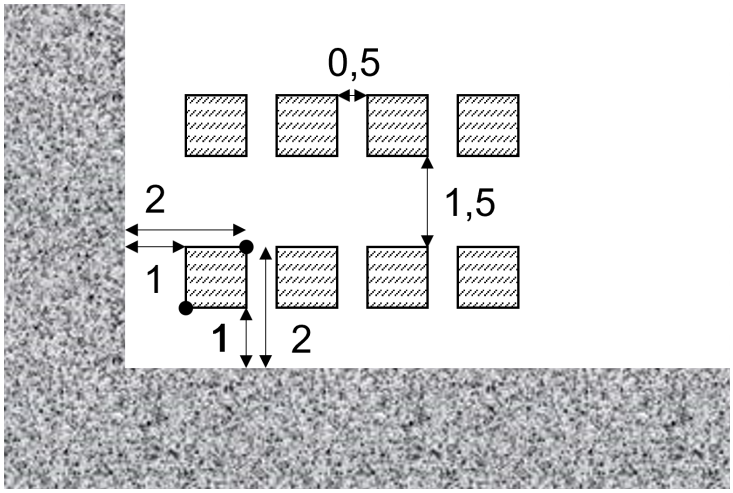
Na sequência da inicialização do transformador, deve-se acrescentar os enrolamentos do transformador um a um. Assim como nos modelos analíticos implementados, o *software* está preparado para tratar enrolamentos com seção retangular e densidade superficial de corrente uniforme. Para adicionar o enrolamento, deve-se descrever as coordenadas dos vértices inferior esquerdo e superior direito do retângulo e sua densidade de corrente. Caso a geometria requerida tenha enrolamentos com as mesmas características espaçados igualmente entre si, o usuário tem a opção de gerar repetições

dele para a direita ou para cima espaçados por distâncias definidas. Para as coordenadas, é tido como a origem o vértice inferior esquerdo da janela do transformador. Como exemplo, o comando abaixo gera o enrolamento da Figura 16.

```
>>> T1.add_winding(x1=0.01, y1=0.01, x2=0.02, y2=0.02, J=1000,
                   repet_x=4, Delta_x=0.005,
                   repet_y=2, Delta_y=0.015)
```

Onde as distâncias estão em metros e $J$ em ampere por metro quadrado.

Figura 16 – Exemplo de entrada de enrolamento no programa, as cotas estão em centímetros.



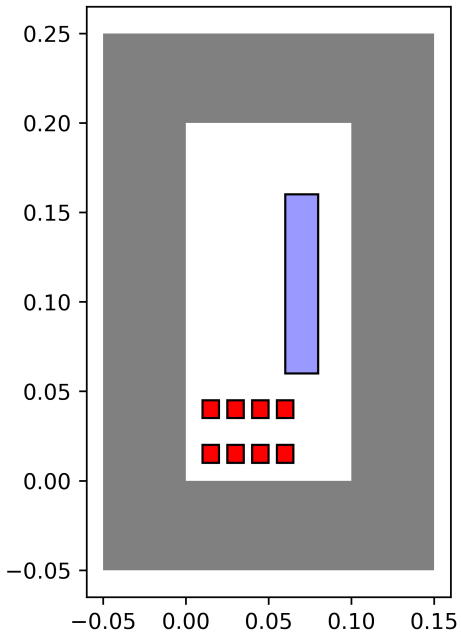

Fonte: O autor (2021).

Após acrescentar os demais enrolamentos, utiliza-se o método *draw_window* para esboçar a janela e inspecionar o resultado das entradas. Outra conferência importante antes de prosseguir é verificar se as fontes de corrente da janela estão balanceadas, como é requisito dos modelos analíticos. A conferência pode ser feita através do objeto *I_balance* que indica o saldo de correntes na janela em amperes. Se *I_balance* apresentar valor nulo, os enrolamentos estão balanceados. Após a impressão do saldo de correntes, o código gera a imagem da figura 17.

```
>>> T1.add_winding(x1=0.06, y1=0.06, x2=0.08, y2=0.16, J=-400)
>>> T1.draw_window()
>>> print('Valor de I_balance: '+str(T1.I_balance))
Valor de I_balance: 0
```

13 errors10 warnings

Figura 17 – Esboço da janela introduzida no programa.




Fonte: O autor (2021).

### 3.1.3 Elementos finitos em duas dimensões e em axissimétrico.

Agora que tem-se a geometria desejada instanciada em *T1*, a biblioteca permite que o usuário utilize o *software Femm* para calcular a indutância pelo método de elementos finitos em 2D e em axissimétrico. A biblioteca permite a simulação numérica das geometrias supostas por Roth e por Rabins. Roth considerou o problema como planar, em coordenadas cartesianas e considerou que à direita da janela, uma perna de ferro a fechava. Rabins por sua vez, tratou o problema como axissimétrico e considerou que exceto pelo material condutor dos enrolamentos, o espaço do início da janela até o infinito é preenchido por ar. Como, até o momento, não foi definido o número de espiras de cada enrolamento, é necessário definir qual a corrente que percorre o enrolamento de referência. Este parâmetro é o único parâmetro não opcional do método *run_FEMM* como pode-se visualizar a seguir.

```
>>> run_FEMM(input_I, simmetry = 'axi', p = 0,
              right_leg = 'steel', close_FEMM = True,
              plot_generation = False):
```

Onde *input_I* é a corrente que percorre o enrolamento para o qual a indutância estará referida. A variável *simmetry* deve ser definida como *'axi'* para geometria axissimétrica ou *'planar'* para que se resolva o método no plano cartesiano. Caso seja planar, deve-se definir o perímetro médio *p* em metros. Como já discutido, o programa permite que a perna à direita da janela seja preenchida por material ferromagnético ou ar. Para

selecionar o material deve-se definir a variável *right_leg* respectivamente como *'steel'* ou *'air'*.

O programa *Femm* possui interface gráfica, caso ao fim do método, seja necessário explorar a interface do *Femm*, fazer ajustes ou acessar variáveis que o módulo *transformer* não contempla será necessário que a aplicação não seja encerrada após a coleta de dados. Para que não se encerre o aplicativo, deve-se dar valor booleano *False* à variável *close_FEMM*.

O último parâmetro de entrada do método é o booleano *plot_generation* o qual define se é de interesse do usuário a coleta dos valores de $J(x,y)$, $A(x,y)$, das componentes $B_x(x,y)$, $B_y(x,y)$ da indução e do seu módulo $B(x,y)$ para plotar posteriormente. Caso o parâmetro tenha valor *True*, os objetos *J_xy_FEMM*, *A_xy_FEMM*, *Bx_xy_FEMM*, *By_xy_FEMM* e *B_xy_FEMM* passarão a armazenar matrizes 2D de tipo *numpy.ndarray* contendo os valores que as grandezas assumem em cada ponto do domínio. A coleta dos campos é um processo computacionalmente demorado e, então, se o objetivo do usuário for somente o cálculo da indutância, sugere-se manter o parâmetro em falso.

A seguir demonstra-se como utilizar o método para, respectivamente, as suposições de Roth e de Rabins para uma corrente de entrada de $1\,A$.

```
>>> T1.run_FEMM(input_I = 1, simmetry = 'planar',
                p = 0.6, right_leg = 'steel')

>>> T1.run_FEMM(input_I = 1, simmetry = 'axi',
                right_leg = 'air')
```

Ao final do processo, o valor calculado para a indutância em Henry estará armazenado no objeto *L_FEMM* e poderá ser acessado através de uma impressão na tela, conforme a linha abaixo.

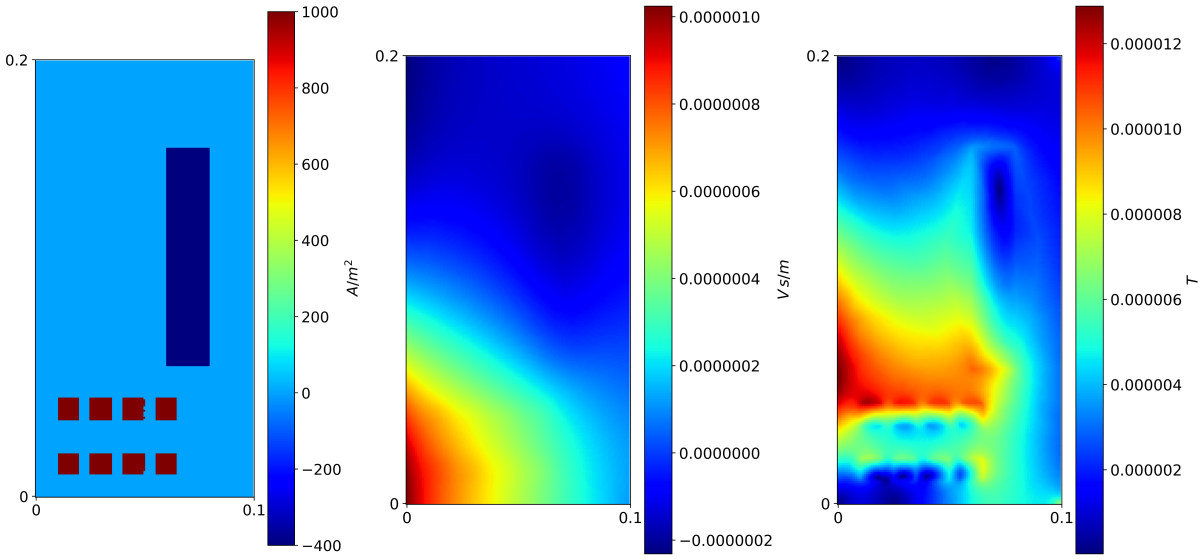
```
>>> print(str(T1.L_FEMM)+' H')
2.931933788834268e-07 H
```

Caso o usuário deseje plotar as variáveis densidade de corrente, potencial vetor magnético ou o módulo da indução magnética no domínio da janela, a biblioteca possibilita *plots* dessas variáveis. Os gráficos são do tipo mapa da variável e são gerados utilizando ferramentas da biblioteca *matplotlib*. Cada função tem como entrada uma variável *save_name*, caso se tenha interesse em salvar o *plot* como imagem, nomeia-se o arquivo através dela. Seguem as linhas de código que resultam nas imagens da Figura 18.

```
>>> T1.plot_heatmap_FEMM_J(save_name = 'FEMM_J')
>>> T1.plot_heatmap_FEMM_A(save_name = 'FEMM_A')
```

```
>>> T1.plot_heatmap_FEMM_B(save_name = 'FEMM_B')
```

Figura 18 – Mapas da densidade de corrente (à esquerda), do potencial vetor magnético (ao centro) e do módulo da indução magnética (à direita) referentes à geometria exemplo resolvida sob as suposições de Rabins.




Fonte: O autor (2021).

### 3.1.4 O método de Roth

Após a programação dos enrolamentos, a biblioteca permite o cálculo da indutância de dispersão através do método de Roth. No método *run_Roth* estão implementadas as equações da seção 2.2.3. Os números de harmônicas em ambas as direções $x$ e $y$ é 500 por padrão, mas podem ser alterados através das variáveis *m_max* e *n_max* do método. Caso o usuário deseje plotar os campos $J$, $A$ e $B$, sugere-se utilizar menos harmônicas para que não haja problemas de alocação de memória. Para cálculos de indutância, o valor de 500 harmônicas está adequado. Além destas entradas opcionais, duas variáveis são obrigatórias no método abaixo.

```
run_Roth(self, input_I, p, m_max = 500, n_max = 500)
```

Deve ser dado o valor da corrente no enrolamento de referência à variável *input_I* em amperes e deve ser definido o perímetro médio dos enrolamentos $p$. Ao fim do método, a indutância de dispersão calculada estará armazenada no objeto *L_Roth*. Abaixo a aplicação do método de Roth e a impressão da indutância a que ele resulta.

```
>>> T1.run_Roth(input_I = 1, p = 0.6, m_max = 20, n_max = 50)
>>> print(str(T1.L_Roth)+' H')
2.899631036035509e-07 H
```

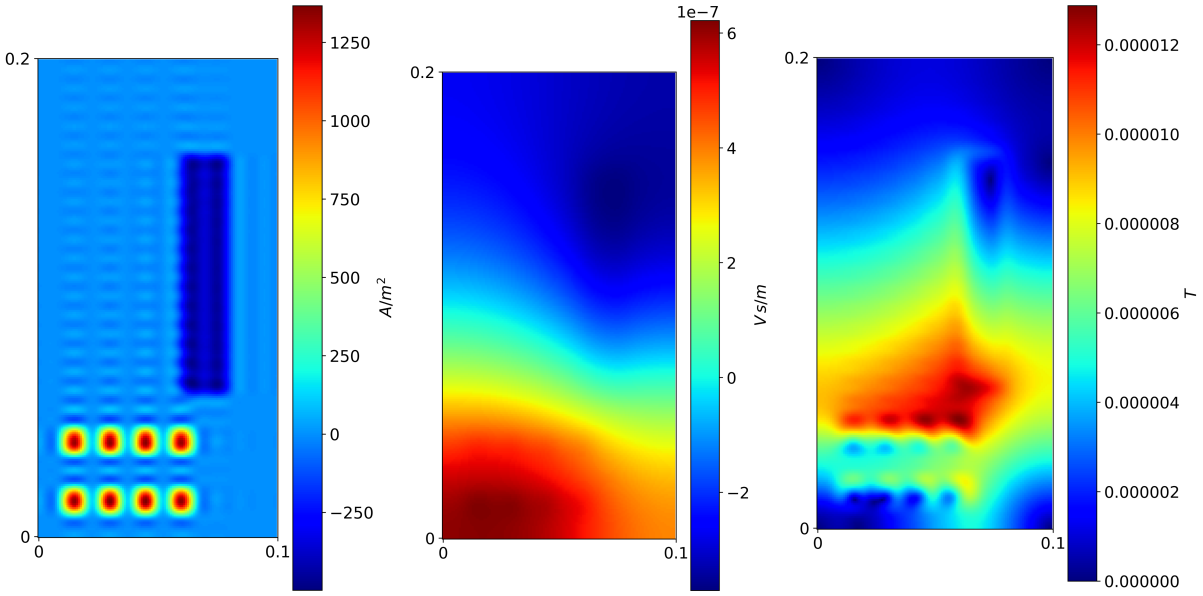
Assim como no método de elementos finitos, a biblioteca permite a confecção de gráficos de mapa das variáveis $J$, $A$ e módulo de $B$. O cálculo de $J(x,y)$ e $A(x,y)$ se dá

através das equações (2.28) e (2.30), enquanto o módulo de $B(x, y)$ foi calculado através do módulo do vetor **B** descrito pela equação (2.34). Os métodos de plotagem de cada grandeza geram matrizes com os valores de $J$, $A$, $B_x$, $B_y$ e $B$ no espaço e as armazena nos objetos *J_xy_Roth*, *A_xy_Roth*, *Bx_xy_Roth*, *By_xy_Roth* e *B_xy_Roth*.

Os métodos abaixo geram as imagens da figura 19. A variável *save_name* segue a mesma lógica da variável de mesmo nome nos mapas do método de elementos finitos.

```
>>> T1.plot_heatmap_Roth_J(save_name = 'Roth_J')
>>> T1.plot_heatmap_Roth_A(save_name = 'Roth_A')
>>> T1.plot_heatmap_Roth_B(save_name = 'Roth_B')
```

Figura 19 – Mapas da densidade de corrente (à esquerda), do potencial vetor magnético (ao centro) e do módulo da indução magnética (à direita) conforme resolução pelo método de Roth.




Fonte: O autor (2021).

### 3.1.5 O método de Rabins

A terceira opção que a biblioteca oferece para o cálculo da indutância de dispersão é o método de Rabins. O método *run_Rabins* realiza o seccionamento do domínio em quantas regiões de Rabins forem necessárias e computa as equações descritas na seção 2.2.4. O método exige somente duas variáveis, conforme a definição abaixo.

```
run_Rabins(input_I, n)
```

A variável *input_I* de entrada do método é a corrente que percorra o enrolamento ao qual a indutância de dispersão estará referida. Deve-se fornecer, também, ao método o número de harmônicas $n$ utilizada na série de Fourier da corrente em cada região. Ao fim da resolução do método, o valor da indutância de dispersão em henry estará

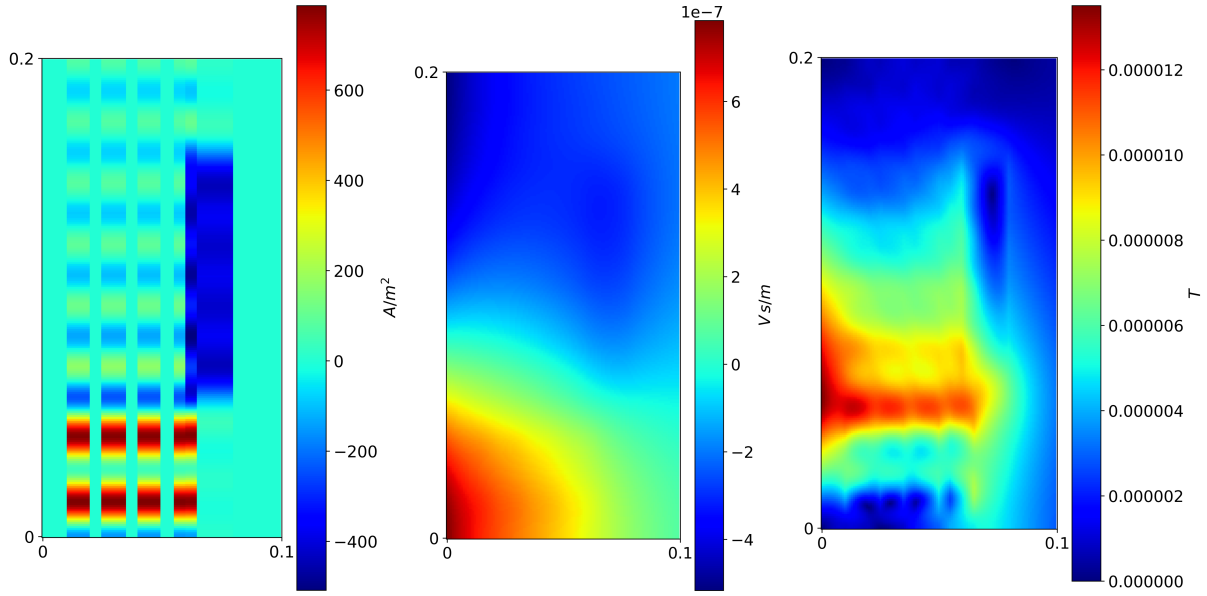
armazenado no objeto *L_Rabins*. As linhas de código que seguem rodam o método de Rabins para o transformador exemplo *T1* e imprimem na tela a indutância de dispersão calculada pelo método.

```
>>> T1.run_Rabins(input_I = 1, n = 15)
>>> print(str(T1.L_Rabins)+' H')
3.0298670301955364e-07 H
```

A biblioteca permite a confecção de gráficos de mapa das variáveis $J(r,z)$, $A(r,z)$ e $B(r,z)$. Os gráficos são impressos na tela e, opcionalmente, salvos no formato *png* através da variável *save_name*. As linhas de código abaixo plotam as variáveis $J$, $A$ e $B$ e salvam os gráficos resultantes. A figura 20 mostra os gráficos impressos na tela para o transformador exemplo *T1*. Durante os métodos de plotagem, são armazenadas nos objetos *J_xy_Roth*, *A_xy_Roth*, *Bx_xy_Roth*, *By_xy_Roth* e *B_xy_Roth* matrizes contendo os valores de cada campo em todo o domínio.

```
>>> T1.plot_heatmap_Rabins_J(save_name = 'Rabins_J')
>>> T1.plot_heatmap_Rabins_A(save_name = 'Rabins_A')
>>> T1.plot_heatmap_Rabins_B(save_name = 'Rabins_B')
```

Figura 20 – Mapas da densidade de corrente (à esquerda), do potencial magnético (ao centro) e do módulo da indução magnética (à direita) conforme resolução pelo método de Rabins.




Fonte: O autor (2021).

### 3.1.6 Gráficos Adicionais

O usuário pode ter interesse em confeccionar gráficos diferentes dos oferecidos diretamente pela biblioteca. É possível efetuar operações nas matrizes que contêm as

grandezas e na sequência plotá-las através do método a seguir.

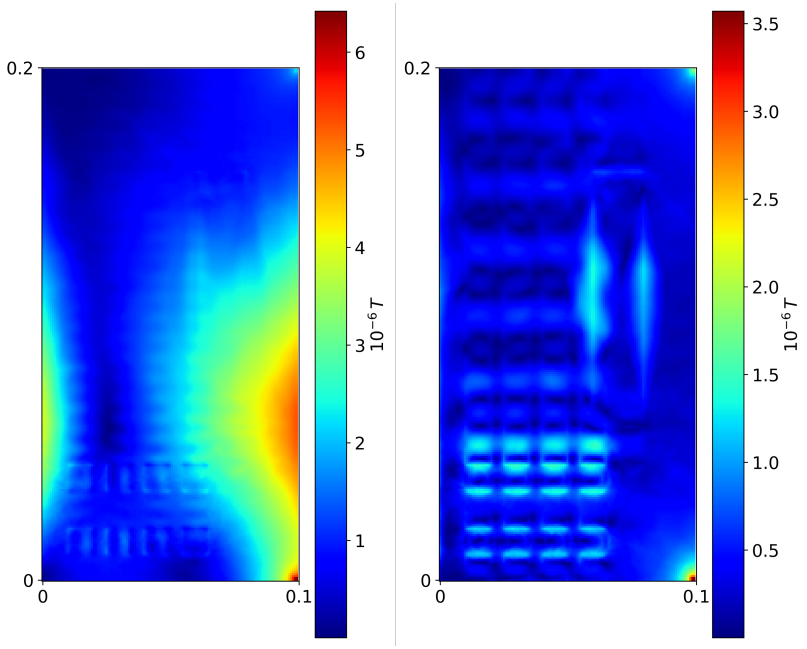
```
plot_heatmap_generic(field_xy, label = '', save_name = False)
```

Onde deve-se inserir a matriz do campo a ser plotado na variável *field_xy*. As variáveis *label* e *save_name* são opcionais. A variável *label* será a legenda da barra do mapa de calor, podendo ser utilizada para inserir as unidades da grandeza graficada. A variável *save_name* deve ser preenchida com o nome do arquivo destino, caso se queira salvar a imagem plotada.

As linhas de código a seguir oferecem exemplos da manipulação das matrizes no cálculo da norma do erro absoluto em $B$ em função do espaço nos cálculos analíticos, tomando como base o valor da indução calculada através do método de elementos finitos. Como se pode visualizar, calculam-se as diferenças nas componentes $x$ e $y$ do vetor $B$ e, em seguida, a sua norma. A geometria escolhida para entrada no *Femm* é a que condiz com as suposições de Rabins, simetria axial e ar preenchendo o espaço à direita da janela. As imagens geradas pelo código abaixo compõem a Figura 21.

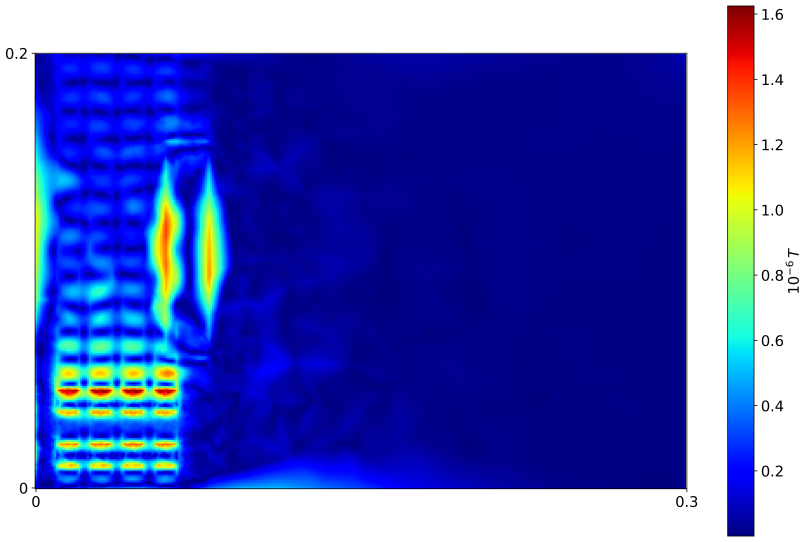
```
>>> residual_Bx_Roth = T1.Bx_xy_Roth - T1.Bx_xy_FEMM
>>> residual_By_Roth = T1.By_xy_Roth - T1.By_xy_FEMM
>>> residual_abs_B_Roth = 1e6*((residual_Bx_Roth**2 +\
                               residual_By_Roth**2)**(1/2))
>>> T1.plot_heatmap_generic(field_xy = residual_abs_B_Roth,
                            label = '$10^{-6}\,T$',
                            save_name = 'Roth_residual_B')
>>> residual_Bx_Rabins = T1.Bx_xy_Rabins - T1.Bx_xy_FEMM
>>> residual_By_Rabins = T1.By_xy_Rabins - T1.By_xy_FEMM
>>> residual_abs_B_Rabins = 1e6*((residual_Bx_Rabins**2 +\
                                  residual_By_Rabins**2)**(1/2))
>>> T1.plot_heatmap_generic(field_xy = residual_abs_B_Rabins,
                            label = '$10^{-6}\,T$',
                            save_name = 'Rabins_residual_B')
```

Figura 21 – Mapas dos módulos dos desvios na indução magnética calculados pelo método de Roth (à esquerda) e Rabins (à direita) quando comparados às simulações por elementos finitos.




Fonte: O autor (2021).

Figura 22 – Mapas dos módulos dos desvios na indução magnética calculados pelo método de Rabins quando comparado à simulação por elementos finitos aumentando a largura da janela.




Fonte: O autor (2021).

Verifica-se, como esperado, que o método de Rabins desvia muito menos da simulação que o de Roth. O método de Roth apresenta problemas para valores pequenos de raio para os quais a sua suposição de comprimento de circunferência muito superior ao raio está inadequada e à direita da janela onde o método de Roth espera que exista

material ferromagnético. O método de Rabins desvia da simulação nos vértices direitos da janela, onde supõe que os jugos superior e inferior se estendam até o infinito. Na simulação os jugos são interrompidos ao fim da janela. Caso a largura $x_0$ da janela fosse aumentada e, por consequência, o comprimento dos jugos na simulação de elementos finitos, o desvio do método de Rabins diminuiria. A Figura 22 demonstra como ficaria o resíduo caso a largura da janela fosse alterada para 300 $mm$.

## 3.2 IMPLEMENTAÇÃO DA FERRAMENTA COMPUTACIONAL

Esta seção se dedica a explicar o código da ferramenta computacional e quais escolhas foram feitas durante o seu desenvolvimento. Este trecho do trabalho permite a expansão do projeto ou o desenvolvimento de projetos semelhantes por próximos estudantes.

### 3.2.1 Inicialização do código

São importadas as bibliotecas *numpy*, *matplotlib.pyplot* e *Femm* nas linhas iniciais do código, bem como a constante *numpy.pi*, as funções trigonométricas *numpy.cos* e *numpy.sin*, a função exponencial *numpy.exp*, as funções de Bessel $K$ de ordem zero *scipy.special.k0*, ordem um *scipy.special.k1* e ordem $n$ *scipy.special.kn*, as funções de Bessel $I$ de ordem zero *scipy.special.i0*, ordem um *scipy.special.i1* e ordem $n$ *scipy.special.iv*, as integrais de zero a $x$ das funções de Bessel $K_0$ e $I_0$ *scipy.special.iti0k0*, a função de Struve $L$ de ordem $n$ *scipy.special.modstruve* e a integral de zero a $x$ da função de Struve $L_0$ *scipy.special.itmodstruve0*. Foi importada também a função *Rectangle* da *matplotlib*. Na sua sequência, define-se a constante $\mu_0$ e, para facilitar a leitura do código, as funções $K_2$, $I_2$, $L_0$, $L_1$ e $L_2$ a partir das funções de ordem $n$ genérica.

Para a implementação do método de Rabins, é necessário o cálculo das integrais das funções de Bessel e Struve multiplicadas por $x$, reescritas pela integração por partes na equação (2.51). Não estão implementadas na biblioteca as integrais das funções de Bessel e Struve em função dos dois limites de integração, mas sim as integrais de zero a $x$ com o zero fixado. Logo, durante a implementação, foi necessário dividir as integrais conforme a equação (3.1) e resolvê-las fracionadas conforme a equação (3.2) através das funções importadas.

$$\begin{aligned}
\int_{x_1}^{x_2} x\, I_1(x)\, dx &= \int_0^{x_2} x\, I_1(x)\, dx - \int_0^{x_1} x\, I_1(x)\, dx \\
\int_{x_1}^{x_2} x\, K_1(x)\, dx &= \int_0^{x_2} x\, K_1(x)\, dx - \int_0^{x_1} x\, K_1(x)\, dx \\
\int_{x_1}^{x_2} x\, L_1(x)\, dx &= \int_0^{x_2} x\, L_1(x)\, dx - \int_0^{x_1} x\, L_1(x)\, dx
\end{aligned} \tag{3.1}$$

$$\begin{aligned}
\int_0^x t\, I_1(t)\, dt &= x\, I_0(x) - \int_0^x I_0(x)\, dx \\
\int_0^x t\, K_1(t)\, dt &= -x\, K_0(x) + \int_0^x K_0(x)\, dx \\
\int_0^x t\, L_1(t)\, dt &= x\, L_0(x) - \frac{x^2}{\pi} - \int_0^x L_0(x)\, dx
\end{aligned} \tag{3.2}$$

Na implementação da segunda integral da equação (3.2) foi necessário o cuidado de tratar o argumento $x = 0$ separadamente de outros números reais. É evidente que é nula uma integral cujos limites de integração são de zero a zero, porém o limite de $K_0(x)$ com $x$ tendendo a zero é infinito. Este é o valor que a função *scipy.special.k0(0)* fornece, gerando um erro ao computar a integral. Criou-se, então, um teste lógico se a entrada $x$ era zero na função *integral_t_K1_0(x)*. Para que se mantivesse a característica da função poder operar tanto em matrizes quanto em números, foi necessário um trecho de código que realiza a detecção do zero em matrizes e outro que verifica se um número é zero.

A implementação dessas integrais foi um dos grandes desafios do trabalho. Não foi possível utilizar as equações sugeridas por livros que tratam do assunto do método de Rabins para transformadores por conta de problemas de convergência. A implementação acabou por ser simples, pois contou com a ajuda de uma biblioteca que toma diversos cuidados para a convergência de funções, dentre eles utiliza equações diferentes para as funções de Bessel e Struve dependendo do seu argumento.

### 3.2.2 Inicialização da classe *transformer*

A inicialização do transformador tem como função fornecer as dimensões básicas do transformador, dimensões da janela e raio do núcleo. A largura e a altura da janela são guardadas nos objetos *self.x0* e *self.y0* e o raio é armazenado em *self.r_core*. Caso se resolva o problema em coordenadas cartesianas, não é necessário especificar o raio ao programa, caso o usuário não o especifique, *self.r_core* passa a ser igual a um quinto da altura da janela.

Durante a inicialização da classe, também são criados vetores vazios que armazenarão as coordenadas e as densidades de corrente dos enrolamentos do transformador. Estes vetores são armazenados nos objetos *self.x1*, *self.x2*, *self.y1*, *self.y2* e *self.J*. Além dos vetores, é necessário iniciar o contador de enrolamentos *self.number_of_windings* e o saldo de corrente na janela *self.I_balance* ambos em zero.

### 3.2.3 Método de inclusão de enrolamento *add_winding*

A inclusão de novos enrolamentos se dá pelo método *add_winding*. Cada enrolamento possui seção retangular e é percorrido por correntes uniformes. Logo somente é necessário que o programa guarde de alguma forma a posição na janela e as dimensões

do retângulo, bem como a densidade da corrente que percorre este enrolamento. Foi decidido durante a redação do código que a posição e as dimensões do retângulo seriam dadas através das coordenadas dos seus vértices inferior esquerdo $(x_1, y_1)$ e superior direito $(x_2, y_2)$ tomando como origem $(0, 0)$ o vértice inferior esquerdo da janela do transformador. O usuário deve entrar com valores em metros para as posições $x_1$, $x_2$, $y_1$ e $y_2$. A densidade de corrente $J$ deve possuir unidade ampere por metro quadrado.

Ao utilizar o programa em seus estágios iniciais, observou-se que muitos transformadores apresentam geometrias em que um enrolamento é composto por diversos blocos retangulares de mesma corrente espaçados igualmente. Com o intuito de facilitar a entrada de enrolamentos deste tipo, foram programados os recursos de repetição da entrada em $x$ ou $y$ com espaçamento $\Delta x$ e $\Delta y$. A implementação das repetições foi realizada através de dois *loops*, um em $x$ e outro em $y$, que geram as coordenadas e densidades de corrente de cada bloco retangular.

Ainda durante a inclusão dos enrolamentos, a cada bloco retangular, o método incrementa um contador de enrolamentos de nome *self.number_of_windings* e adiciona a corrente total do enrolamento ao saldo de correntes *self.I_balance*.

### 3.2.4 Método do esboço da janela *draw_window*

Até o momento, não existe maneira fácil do usuário verificar a geometria inserida. Mesmo o usuário tendo acesso aos vetores das características dos enrolamentos, o máximo que pode ser feito para verificá-los é a sua impressão na tela. Com o intuito de facilitar a verificação, foi programado um método para esboçar a vista em corte do transformador. Os enrolamentos são desenhados na janela e coloridos com vermelho, caso $J$ positivo, ou azul, caso $J$ negativo. Quanto maior a densidade de corrente, mais escura a cor do preenchimento.

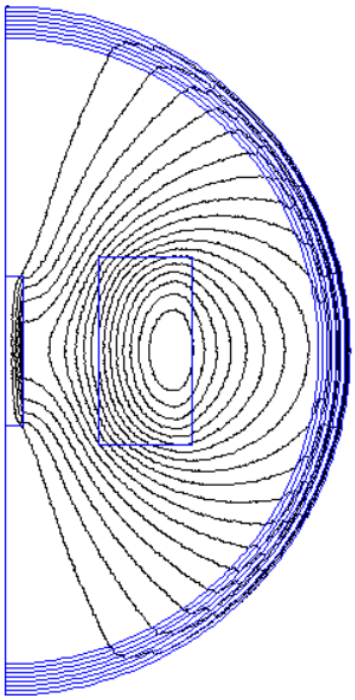
Na implementação, inicialmente, define-se um objeto do tipo *matplotlig.pyplot.subplots*. Na sequência, é desenhado um retângulo cinzento para representar o núcleo e um retângulo branco que representa o ar na janela. Na sequência, normaliza-se as densidades de correntes de todos os enrolamentos pela densidade de corrente mais alta da janela em módulo e define-se qual é a cor que cada enrolamento será pintado através do sinal e intensidade normalizada das correntes. Definidas as cores, basta desenhar os retângulos que representam os enrolamentos.

### 3.2.5 Método de solução do problema por elementos finitos *run_FEMM*

Como base para o desenvolvimento da interface da ferramenta com o *Femm* foi utilizado o exemplo *DC Magnetics: Demo2 Example* disponível na documentação da biblioteca (MEEKER, 2018). O exemplo desenha uma geometria axissimétrica com um cilindro de aço não linear e um enrolamento de cobre com seção quadrada, calcula a

indutância própria do enrolamento, adquire a indução magnética na reta $r = 0$ e traça o gráfico da indução em função de $z$. A geometria do exemplo está representada na figura 23.

Figura 23 – Geometria do exemplo *DC Magnetics: Demo2 Example* da biblioteca *pyfemm*.



Fonte: (MEEKER, 2018).

No método *run_FEMM* está implementado algo similar ao exemplo, porém deve-se desenhar uma geometria genérica de transformador e os dados de interesse a serem coletados são a energia armazenada no campo magnético e os campos $J$, $A$ e $B$ em todo o domínio da janela.

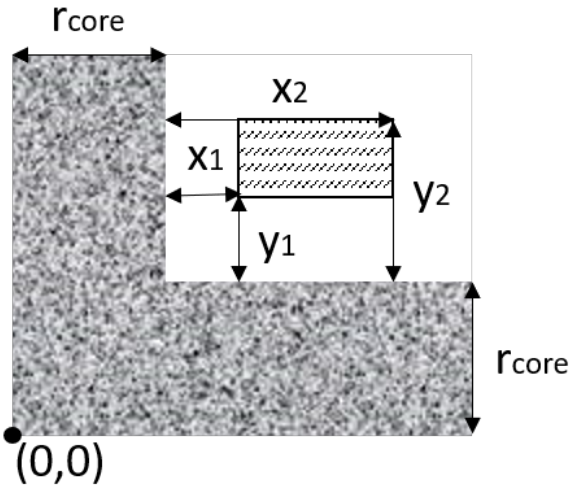
Inicialmente o programa converte todas as distâncias para milímetros, pois esta é a unidade para comprimentos que será utilizada no *Femm*. Outra mudança de variável necessária foi somar o valor do raio do núcleo às posições $x_1$, $x_2$, $y_1$ e $y_2$ pois a origem considerada no desenho do *Femm* será $r = 0$ no centro do núcleo e $z = 0$ na base do jugo inferior. Conforme a figura 24.

Na sequência, é necessário multiplicar as densidades de corrente de cada enrolamento pela sua seção transversal para calcular a corrente total que passa pela seção retangular de cada um dos enrolamentos.

Caso a fonte de corrente seja inserida no *pyfemm* em uma espira com a seção transversal do enrolamento, o *Femm* interpreta que a densidade de corrente não será uniforme em um problema axissimétrico. A interpretação é que a circunferência interna do enrolamento é menor que a externa e por este motivo teria menor resistência, gerando uma densidade de corrente inversamente proporcional ao raio. Como os enrolamentos

do transformador são compostos por diversas espiras em série, este comportamento não condiz com a realidade do fenômeno estudado.

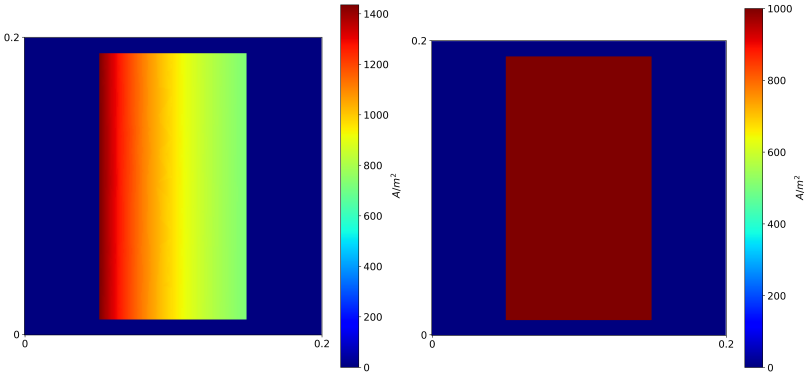
Figura 24 – Mudança de variáveis de $x_1$, $x_2$, $y_1$ e $y_2$ para entrada no *Femm*.




Fonte: O autor (2021).

Para simular corrente uniforme nos enrolamentos, a biblioteca *pyfemm* permite que cada enrolamento seja composto por espiras em série. Como o objetivo é somente criar as fontes de corrente na janela, o número de espiras não é relevante, desde que gere uma densidade de corrente uniforme igual à definida pelo usuário. Obteve-se um bom resultado ao definir-se mil espiras em série por enrolamento, cada espira percorrida por um milésimo da corrente total. A figura 25 demonstra como o *Femm* trataria o problema caso utilizada somente uma espira por enrolamento, comparando-a com o programa implementado com mil espiras.

Figura 25 – Gráfico do tipo mapa da interpretação do *Femm* de um enrolamento com uma espira (à esquerda) em comparação com um enrolamento com mil espiras em série (à direita).




Fonte: O autor (2021).

Na sequência utiliza-se o *pyfemm* para abrir o *Femm*, caso o usuário tenha solicitado que o *Femm* não seja encerrado no fim da sessão, o programa é aberto em primeiro plano, caso queira que a aplicação se encerre, ela será aberta em plano de fundo. Um novo documento do *Femm* é, então, criado e nele definido o tipo de problema. Dimensões em milímetros, simetria axial ou plana conforme entrada do usuário, precisão de $10^{-8}$ e, caso geometria planar, a profundidade da geometria. Os últimos comandos de definição do problema são as definições de propriedades dos materiais ar, bobinas e aço linear.

O próximo passo é desenhar as geometrias, definir propriedades dos materiais e definir fontes de corrente. Inicialmente desenha-se o retângulo externo do material ferromagnético, insere-se um ponto de identificação de bloco na coordenada $(r_{core}/2, r_{core}/2)$ e define-se que o material deste bloco é aço linear. Na sequência, desenha-se linhas verticais que separam a perna à direita da janela dos jugos superior e inferior, permitindo assim que defina-se o material desta perna de maneira independente do restante do núcleo. É criada uma nova definição de material como ar ou aço dependendo do que está armazenado na entrada *right_leg*. Após o desenho do ferro, o programa define as condições de contorno como abertas através da função *femm.mi_makeABC* da biblioteca *pyfemm*

A janela de ar é desenhada e dois pontos de identificação de bloco são posicionados, um na coordenada $(1, -1)$ logo abaixo do jugo inferior e outro próximo do canto inferior esquerdo da janela. Os dois pontos são definidos como material ar. Por fim, cria-se um *loop* que passa por todos os enrolamentos. A cada iteração do *loop*, o enrolamento é desenhado, é criada uma propriedade de circuito com corrente igual a um milésimo da corrente total que percorre o bloco. No centro do retângulo é adicionado um identificador de bloco que é subsequentemente preenchido com o material "bobina" e mil enrolamentos do último circuito instanciado.

O arquivo da geometria é salvo com nome *transformer.fem* e o método de elementos finitos é chamado pelo programa. A solução é carregada pelo comando *femm.mi_loadsolution*, é selecionado o domínio de cálculo inteiro e solicita-se ao *Femm* que retorne a energia armazenada no campo magnético do domínio selecionado. A energia é armazenada internamente na variável $W_{FEMM}$ e é usada como entrada do cálculo da indutância de dispersão através da equação (3.3). Onde $I$ é a corrente de entrada *input_I* fornecida pelo usuário. A indutância resultante é armazenada no objeto *self.L_FEMM*.

$$L_{FEMM} = \frac{2\,W_{FEMM}}{I^2} \tag{3.3}$$

Em seguida, é verificado se o usuário deseja que sejam coletadas matrizes contendo os valores de $J(r,z)$, $A(r,z)$ e $B(r,z)$ para plotar posteriormente. Caso a variável

*plot_generation* tenha valor booleano verdadeiro, divide-se o domínio em pixels. A janela tem sua altura $y_0$ dividida em 250 passos e sua largura $x_0$ dividida em uma quantidade de passos que garanta a proporção entre $x_0$ e $y_0$. Utiliza-se dois *loops*, um em $x$ e outro em $y$, para coletar os campos $J$, $A$, $B_x$ e $B_y$ através de funções nativas da biblioteca *pyfemm*. As primeiras e últimas linhas e colunas das matrizes podem estar incorretas pois no domínio elas representam interfaces entre aço e ar, o interesse está somente na parte da janela. Escreve-se então que para os campos $A$, $B_x$ e $B_y$, as primeiras linhas e colunas recebem valores iguais às segundas e as últimas recebem os valores das penúltimas. O módulo da indução magnética é calculado a partir das suas componentes $x$ e $y$.

Foi observado que, em coordenadas cilíndricas, a biblioteca *pyfemm* coleta $2\,\pi\,r\,A$, em vez de $A$ e, então, para armazenar o valor de $A$, deve-se dividir os valores coletados por $2\,\pi\,r$. A biblioteca também fornece valores de $B_x$ e $B_y$ com sinais invertidos quando resolve um problema planar e, então, para armazenar os valores das componentes da indução caso geometria planar, deve-se multiplicar os valores coletados por menos um.

As matrizes contendo os valores de $J(x,y)$, $A(x,y)$, $B_x(x,y)$, $B_y(x,y)$ e $|B(x,y)|$ são armazenadas nos objetos *self.Jxy_FEMM*, *self.Axy_FEMM*, *self. Bx_xy_FEMM*, *self. By_xy_FEMM* e *self.Bxy_FEMM*. Por fim, caso o booleano *close_FEMM* seja verdadeiro, o *software Femm* é encerrado.

### 3.2.6 Método de solução do problema pelo método de Roth *run_Roth*

O primeiro método analítico que foi implementado foi o método de Roth. A densidade de corrente é tratada por Roth como uma série dupla de Fourier e, por este motivo, as linhas iniciais do código calculam os coeficientes $J_{mn}$ da série de Fourier da corrente na janela. Inicialmente, armazenam-se os números de termos em $x$ e $y$ da série de Fourier nos objetos *self.m_max* e *self.n_max*.

O método exige que a janela do transformador seja redimensionada para largura e altura igual a $\pi$ radianos e, por este motivo, todos os comprimentos de interesse em $x$ devem ser multiplicados pelo fator $\pi/x_0$, os comprimentos em $y$ devem ser multiplicados pelo fator $\pi/y_0$ e as densidades de corrente devem ser convertidas de $A/m$ para $A/rad$ através da multiplicação pela área da janela dividida por $\pi$ ao quadrado. Com os comprimentos e as densidades de corrente redimensionados são, então, armazenados nas variáveis locais $\theta_{x1}$, $\theta_{x2}$, $\theta_{y1}$, $\theta_{y2}$ e $J_{rad}$. As cinco variáveis são vetores com $k$ entradas, onde $k$ é o número de enrolamentos do transformador.

São criados dois vetores com os números naturais de 1 a $m_{max}$ e 1 a $n_{max}$. Os vetores são repetidos por $k$ colunas para formar as matrizes $m_{m_{max}\times k}$ e $n_{n_{max}\times k}$. É instanciada uma matriz $Jmn_{(m_{max}+1)\times(n_{max}+1)}$ com todas entradas iniciadas em zero. É

implementada então a equação (3.4) para a geometria do transformador.

$$J_{mn} = \begin{array}{ll} \frac{1}{\pi^2}\sum_{i=1}^{k} J^{(i)}\left(\theta_{x_2}^{(i)}-\theta_{x_1}^{(i)}\right)\left(\theta_{y_2}^{(i)}-\theta_{y_1}^{(i)}\right) & \begin{array}{l} m{=}0 \\ n{=}0 \end{array} \\ \frac{2}{n\pi^2}\sum_{i=1}^{k} J^{(i)}\left(\theta_{x_2}^{(i)}-\theta_{x_1}^{(i)}\right)\left(\sin n\theta_{y_2}^{(i)}-\sin n\theta_{y_1}^{(i)}\right) & \begin{array}{l} m{=}0 \\ n{=}1,2,\ldots \end{array} \\ \frac{2}{m\pi^2}\sum_{i=1}^{k} J^{(i)}\left(\sin m\theta_{x_2}^{(i)}-\sin m\theta_{x_2}^{(i)}\right)\left(\theta_{y_2}^{(i)}-\theta_{y_1}^{(i)}\right) & \begin{array}{l} m{=}1,2,\ldots \\ n{=}0 \end{array} \\ \frac{4}{mn\pi^2}\sum_{i=1}^{k} J^{(i)}\left(\sin m\theta_{x_2}^{(i)}-\sin m\theta_{x_1}^{(i)}\right)\left(\sin n\theta_{y_2}^{(i)}-\sin n\theta_{y_1}^{(i)}\right) & \begin{array}{l} m{=}1,2,\ldots \\ n{=}1,2,\ldots \end{array} \end{array} \tag{3.4}$$

O valor de $J_{00}$ é obtido através da soma das densidades de corrente dos enrolamentos ponderadas pela razão da área ocupada por ele na janela. É implementado a partir de uma multiplicação termo a termo dos vetores $J_{rad}$ por $(\theta_{x2} - \theta_{x1})$ e $(\theta_{y2} - \theta_{y1})$ dividido por $\pi^2$. Soma-se então todas as entradas do vetor resultante da multiplicação.

Para calcular os demais valores da linha $J_{0n}$, multiplica-se termo a termo os vetores $\theta_{y2}$ e $\theta_{y1}$ pela matriz $n_{n_{max}\times k}$. A biblioteca realiza a distribuição da multiplicação dos vetores por todas as $n_{max}$ linhas. Calcula-se o seno das entradas das matrizes resultantes e os senos são subtraídos. A matriz contendo a subtração dos senos é dividida elemento a elemento pela matriz $n$ e dividida novamente por $\pi^2$. Tem-se então uma matriz de formato $n_{max} \times k$ que deve ser multiplicada elemento a elemento pelos vetores de $k$ entradas $J_{rad}$ e $\theta_{x2} - \theta_{x1}$. Por fim, multiplica-se a matriz resultante por 2 e somam-se as suas linhas para gerar um vetor de $k$ entradas que será armazenado em $J_{0,n}$ com $n$ maior que zero.

Os valores da coluna $J_{m,0}$ com $m$ maior que zero são calculados da mesma forma que na linha $J_{0,n}$, exceto pelas trocas das variáveis $n$ por $m$, $x$ por $y$ e vice-versa.

Para completar o cálculo da amplitude das harmônicas, foi criado um *loop* para computar as entradas da matriz $J_{m,n}$ com $m$ e $n$ maiores que zero. Cada iteração do *loop* calcula os coeficientes da série de Fourier para um dos $k$ enrolamentos e incrementa os seus valores na matriz $J_{mn}$.

Anteriormente ao *loop*, cria-se dois tensores de três dimensões. Armazena-se nas variáveis *__aux1* e *__aux2* tensores de dimensões $m_{max} \times 1 \times k$ e $1 \times n_{max} \times k$ conforme a equação (3.5).

$$\begin{aligned} aux_1[m,1,i] &= \frac{sin\left(m\,\theta_{x2}[i]\right) - sin\left(m\,\theta_{x1}[i]\right)}{m} \\ aux_2[1,n,i] &= \frac{sin\left(n\,\theta_{y2}[i]\right) - sin\left(n\,\theta_{y1}[i]\right)}{n} \end{aligned} \tag{3.5}$$

A cada uma das $k$ iterações do *loop*, calcula-se a contribuição do i-ésimo enrolamento para as entradas da matriz $J_{m,n}$ com $m$ e $n$ maiores que zero, através da multiplicação da constante $4\,J_{rad}[i]\,/\,\pi^2$ pelo produto matricial da i-ésima coluna da variável *__aux1* pela i-ésima linha da variável *__aux2*.

Na sequência é criada uma matriz $h_{mn}$ cujo elemento $h_{00}$ é 1, os demais elementos da linha zero e da coluna zero são 2 e os elementos para $m$ e $n$ maior que zero assumem valor 4. É criada uma variável auxiliar *__my_nx_sq* para armazenar uma matriz $m \times n$ contendo a soma dos quadrados de $m\,x_0$ e $n\,y_0$.

Através da equação (3.6), computa-se as entradas da matriz $A_{mn}$. Soma-se as entradas da multiplicação elemento a elemento das matrizes $A_{mn}$, $J_{mn}$ e $1/h$. Este somatório é então multiplicado pela constante $2\,\pi^2\,p\,I^2_{input}$, resultando na indutância de dispersão do método de Roth conforme a equação (3.7).

$$A_{mn} = \mu_0 \frac{J_{mn} x_0 y_0}{m^2 y_0^2 + n^2 x_0^2} \tag{3.6}$$

$$L = \frac{\pi^2 p}{I^2} \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \frac{J_{mn}\,A_{mn}}{h} \tag{3.7}$$

Por fim as matrizes $J_{mn}$ e $A_{mn}$, bem como a indutância de dispersão são armazenadas nos objetos *self.Jmn*, *self.Amn* e *self.L_Roth*.

#### 3.2.6.1 Métodos de computação de $J(x,y)$, $A(x,y)$ e $B(x,y)$ pelo método de Roth.

Os métodos *calc_J_xy* e *calc_A_xy* são implementados de forma semelhante. Eles têm como objetivo computar os valores das grandezas $J$ e $A$ em todo o domínio de cálculo a partir dos coeficientes das séries de Fourier duplas armazenadas em *self.Jmn* e *self.Amn*.

Inicialmente, define-se o número de vezes que dividirá o domínio de cálculo. Por padrão, a resolução é a divisão de $y_0$ em 250 passos e a de $x_0$ em um número de passos proporcional à razão entre a largura e a altura da janela. Os números de passos em cada direção são armazenados nas variáveis internas $steps_x$ e $steps_y$.

É definido um vetor linha e um vetor coluna, cujas entradas são os valores de $x$ e $y$ discretizados em $steps_x$ e $steps_y$ passos. Ao passo em $x$ dá-se a notação $\Delta x$ e ao passo em $y$ a notação $\Delta y$. Repete-se o vetor linha $steps_y$ vezes e o vetor coluna $steps_x$ vezes para formar duas matrizes de dimensões $steps_y \times steps_x$ cujas entradas são descritas a partir da equação (3.8).

$$\begin{aligned} x\,[i,j] &= j\,\Delta x \\ y\,[i,j] &= i\,\Delta y \end{aligned} \tag{3.8}$$

Multiplica-se as matrizes $x$ e $y$ por vetores de dimensões $1 \times 1 \times m_{max} \times 1$ e $1 \times 1 \times 1 \times n_{max}$, respectivamente, cujas entradas são os números naturais de 0 a $m_{max}$ e 0 a $n_{max}$. Multiplica-se ainda os tensores por $\pi/x_0$ e $\pi/y_0$. Os resultados são os tensores de quatro dimensões $m\theta_x$ e $n\theta_y$. A equação (3.9) descreve as entradas dos tensores.

$$\begin{aligned} m\theta_x\,[i,j,m,1] &= \frac{m\,(j\,\Delta x)\,\pi}{x_0} \\ n\theta_y\,[i,j,1,n] &= \frac{n\,(i\,\Delta y)\,\pi}{y_0} \end{aligned} \tag{3.9}$$

A biblioteca *numpy* permite que sejam calculados os cossenos de cada elemento dos tensores e multiplicá-los elemento a elemento para que se chegue a um tensor $cos(m\theta_x)\,cos(n\theta_y)$ de dimensões $steps_y \times steps_x \times m_{max} \times n_{max}$ cujas entradas estão descritas na equação (3.10).

$$cos(mx)\,cos(ny)\,[i,j,m,1] = cos\left(\frac{m\,(j\,\Delta x)\,\pi}{x_0}\right)\,cos\left(\frac{n\,(i\,\Delta y)\,\pi}{y_0}\right) \tag{3.10}$$

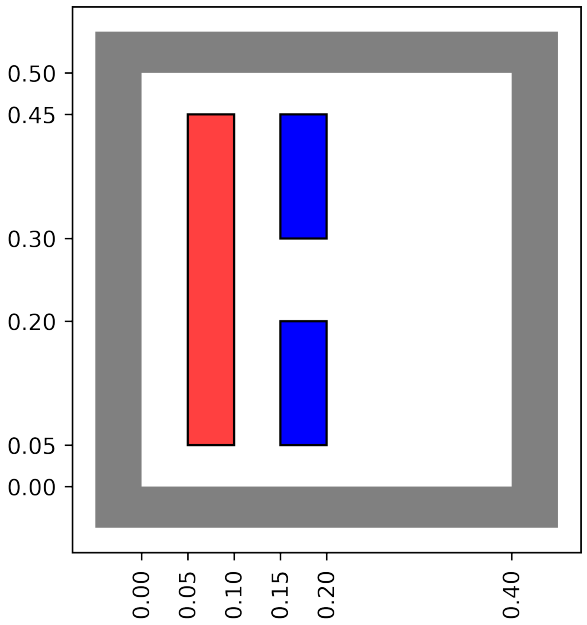
Multiplica-se elemento a elemento cada uma das matrizes $m \times n$ que compõem o tensor pelos coeficientes armazenados no objeto *self.J_mn* (ou *self.A_mn* dependendo da grandeza física que se deseja plotar). Esta multiplicação forma o tensor $J_{xymn}$ ($A_{xymn}$) e, então, são feitos sucessivos somatórios dos termos em $n$ e $m$ para resultar em uma matriz $J_{xy}$ ($A_{xy}$) que contém o valor do campo em todos os pontos discretizados do domínio. A equação (3.11) descreve os elementos do versor $J_{xymn}$ e a equação (3.12) descreve os da matriz $J_{xy}$.

$$J_{xymn}\,[i,j,m,1] = J_{mn}\,[m,n]\,cos\left(\frac{m\,(j\,\Delta x)\,\pi}{x_0}\right)\,cos\left(\frac{n\,(i\,\Delta y)\,\pi}{y_0}\right) \tag{3.11}$$

$$J_{xy}\,[i,j] = \sum_{m=0}^{m_{max}}\sum_{n=0}^{n_{max}} J_{mn}\,[m,n]\,cos\left(\frac{m\,(j\,\Delta x)\,\pi}{x_0}\right)\,cos\left(\frac{n\,(i\,\Delta y)\,\pi}{y_0}\right) \tag{3.12}$$

A matriz $J_{xy}$ ($A_{xy}$) é então armazenada no objeto *self.J_xy_Roth* (*self.A_xy_Roth*). Avaliou-se no fim do projeto que caso se deseje plotar os campos com muitas harmônicas, os tensores quadridimensionais acabam ficando com um número de entradas que pode extrapolar a memória do computador que está rodando o programa. Caso fosse implementada a rotina novamente, optar-se-ia por um caminho que não dependesse do uso simultâneo desta quantidade de memória, podendo-se implementar os somatórios através de *loops* e não operações com tensores.

Figura 26 – Geometria exemplo para a implementação do método de Rabins.




Fonte: O autor (2021).

No caso do método *calc_B_xy*, deve-se calcular $B_{xy}$ em função dos coeficientes em $A_{mn}$ já calculados. A equação (3.13) apresenta os campos $\mathbf{B_x}$ e $\mathbf{B_y}$.

$$\begin{aligned} \mathbf{B_x}(x,y) &= -\sum_{m=0}^{\infty}\sum_{n=0}^{\infty} A_{mn}\,\frac{n\,\pi}{y_0}\,cos\left(\frac{m\,(j\,\Delta x)\,\pi}{x_0}\right)\,sin\left(\frac{n\,(i\,\Delta y)\,\pi}{y_0}\right)\,\hat{\imath} \\ \mathbf{B_y}(x,y) &= \sum_{m=0}^{\infty}\sum_{n=0}^{\infty} A_{mn}\,\frac{m\,\pi}{x_0}\,sin\left(\frac{m\,(j\,\Delta x)\,\pi}{x_0}\right)\,cos\left(\frac{n\,(i\,\Delta y)\,\pi}{y_0}\right)\,\hat{\jmath} \end{aligned} \tag{3.13}$$

De maneira semelhante ao cálculo da matriz $J_{xy}$, são calculadas as matrizes $Bx_{xy}$ e $By_{xy}$. Através da raiz quadrada da soma dos quadrados de $B_x$ e $B_y$, tem-se então o módulo do vetor $B$. As matrizes com os valores das componentes $x$ e $y$ e o módulo da indução em todos os pontos do domínio são, então, armazenadas nos objetos *self.Bx_xy_Roth*, *self.By_xy_Roth* e *self.B_xy_Roth*, respectivamente.

### 3.2.7 Método de solução do problema pelo método de Rabins *run_Rabins*.

Nesta seção, optou-se por utilizar um exemplo para demonstrar o funcionamento do programa para que se facilite a visualização das variáveis. O exemplo utilizado será uma geometria composta por três blocos de seção retangular, o enrolamento de baixa tensão está localizado mais próximo ao núcleo à esquerda da janela e os demais blocos compõem o enrolamento de alta tensão e estão posicionados à direita conforme a Figura 26.

Diferentemente do método de Roth em que se trabalha com o domínio como um

todo, o método de Rabins exige que o domínio da janela do transformador se divida em regiões. Para realizar a discretização da janela, inicialmente escreveu-se o método *generate_regions* que fornecerá ao método *run_Rabins* as séries de Fourier das fontes de corrente de cada região, bem como quais regiões devem ser consideradas como regiões de corrente e quais regiões são preenchidas completamente por ar. A generalidade da ferramenta depende que esta função seja implementada para uma geometria qualquer, conforme será apresentado.

Deve-se definir então o número necessário de regiões para o método de Rabins. O que define a divisão é que $J$ não varia radialmente dentro de uma mesma região, possibilitando que $J(r,z)$ possa ser descrita através de uma série de Fourier somente em $z$ em cada uma delas.

Observa-se que o exemplo deve ser dividido em cinco regiões. A região mais interna composta por ar, seguida pela região que contém o enrolamento de baixa tensão, uma coluna de ar, o enrolamento de alta tensão e a região externa sem a presença de correntes.

Como ponto de partida para definir as regiões, é necessário encontrar as posições em que ocorrem transições de regiões. As interfaces estarão sempre localizadas no início e fim da janela e nas posições em que foram inseridos enrolamentos. Desta forma, basta instanciar um vetor que contenha as posições 0, $x_0$ e as contidas nos vetores $x_1$ e $x_2$, descartando posições que possam aparecer duplicadas. É isto que está implementado, o programa define um vetor $all_x$ que concatena um vetor que contém uma entrada nula aos vetores armazenados nos objetos *self.x1* e *self.x2* e a um vetor que contém a largura da janela armazenada no objeto *self.x0*. Na sequência, utiliza-se a função *numpy.unique* para eliminar as posições repetidas e ordená-las em ordem crescente. O resultado é armazenado no objeto *self.region_x*.

Para o exemplo apresentado, as equações (3.14), (3.15), (3.16) e (3.17) descrevem como os dados são tratados. Nota-se que o número de entradas do vetor *region_x* é o número de regiões incrementado em 1. Armazena-se então o comprimento deste vetor subtraído por 1 (no caso, 5) no objeto *self.number_of_regions*.

$$x_1 = \begin{bmatrix} 0,05 & 0,15 & 0,15 \end{bmatrix} \tag{3.14}$$

$$x_2 = \begin{bmatrix} 0,10 & 0,20 & 0,20 \end{bmatrix} \tag{3.15}$$

$$all\_x = \begin{bmatrix} 0 & 0,05 & 0,15 & 0,15 & 0,10 & 0,20 & 0,20 & 0,40 \end{bmatrix} \tag{3.16}$$

$$region\_x = \begin{bmatrix} 0 & 0,05 & 0,10 & 0,15 & 0,20 & 0,40 \end{bmatrix} \tag{3.17}$$

Na sequência tem-se interesse de localizar a quais regiões, cada enrolamento pertence. No exemplo, numerando-se as regiões de zero a quatro e os enrolamentos de zero a dois, tem-se que o enrolamento zero pertence à região um e os demais pertencem a região três. Para estruturar esta relação de pertencimento, cria-se uma matriz de booleanos com 3 (número de enrolamentos) linhas e 5 (número de regiões) colunas. O elemento $ij$ desta matriz será 1 caso o enrolamento $i$ esteja contido total ou parcialmente na região $j$ e será 0 caso contrário.

Para que se chegue a esta estrutura, foi necessário comparar cada entrada do vetor $region\_x$ aos vetores $x_1$ e $x_2$. Caso o valor de $region\_x[j]$ seja simultaneamente maior ou igual a $x_1[i]$ e menor que $x_2[i]$ conclui-se que a região $j$ contém o elemento $i$. Na implementação, repetiu-se o vetor $region\_x$ por três linhas, formando a variável auxiliar $region\_x\_matrix$ que foi comparada coluna a coluna com os elementos dos vetores $x_1$ e $x_2$. Gera-se duas matrizes de dimensão $3 \times 6$ cujos elementos assumem os valores booleanos $region\_x[j] \geq x_1[i]$ e $region\_x[j] < x_2[i]$. No exemplo estas matrizes auxiliares podem ser escritas conforme as equações (3.18) e (3.19).

$$aux_1 = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{3.18}$$

$$aux_2 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \tag{3.19}$$

Multiplicando as duas matrizes elemento a elemento e rejeitando a última coluna, obtém-se a matriz $3 \times 5$ desejada. Esta matriz é então armazenada no objeto *self.winding_belongs_to_region*. Para o exemplo, esta matriz recebe os valores descritos na equação (3.20).

$$winding\_belongs\_to\_region = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.20}$$

Antes de aplicar o método de Rabins, deve-se calcular a série de Fourier em $z$ da corrente em cada região. Para encontrá-la, calcula-se os coeficientes da série de Fourier de cada enrolamento como se cada enrolamento estivesse sozinho na região. As séries de Fourier das fontes de correntes das regiões será um somatório das séries dos enrolamentos que estão contidos nela. Neste exemplo, por simplicidade de notação,

serão utilizados somente seis coeficientes de $J_0$ a $J_5$. Este parâmetro pode ser escolhido através da entrada do parâmetro $n = 5$ no método *run_Rabins*.

O método *J_winding_fourier* é chamado para calcular a série de Fourier dos 3 enrolamentos, armazenando os seus coeficientes no objeto *self.Jn_winding*, uma matriz de dimensão $3 \times 6$ cujo elemento $ij$ representa o coeficiente $j$ da série de Fourier do enrolamento $i$. Para o exemplo, a matriz $Jn_{winding}$ recebe os valores descritos na equação (3.21).

$$Jn_{winding} = \begin{bmatrix} 2,40 \cdot 10^{3} & 2,12 \cdot 10^{-13} & -1,12 \cdot 10^{3} & 7,07 \cdot 10^{-14} & -9,08 \cdot 10^{2} & 0,00 \cdot 10^{00} \\ -1,20 \cdot 10^{3} & -1,63 \cdot 10^{3} & -1,41 \cdot 10^{-13} & 1,19 \cdot 10^{3} & 1,21 \cdot 10^{3} & 5,09 \cdot 10^{2} \\ -1,20 \cdot 10^{3} & 1,63 \cdot 10^{3} & -9,90 \cdot 10^{-13} & -1,19 \cdot 10^{3} & 1,21 \cdot 10^{3} & -5,09 \cdot 10^{2} \end{bmatrix} \tag{3.21}$$

Após isto, utiliza-se a matriz *winding_belongs_to_region* para realizar o somatório dos coeficientes dos enrolamentos que pertencem a cada região. Os coeficientes das regiões são então armazenados no objeto *self.Jn_regions*, para o exemplo a matriz $5 \times 6$ está apresentada na equação (3.22).

$$Jn_{regions} = \begin{bmatrix} 0,00 & 0,00 & 0,00 & 0,00 & 0,00 & 0,00 \\ 2,40 \cdot 10^{3} & 2,12 \cdot 10^{-13} & -1,12 \cdot 10^{3} & 7,07 \cdot 10^{-14} & -9,08 \cdot 10^{2} & 0,00 \\ 0,00 & 0,00 & 0,00 & 0,00 & 0,00 & 0,00 \\ -2,40 \cdot 10^{3} & 4,55 \cdot 10^{-13} & -1,13 \cdot 10^{-12} & 9,09 \cdot 10^{-13} & 2,42 \cdot 10^{3} & -5,68 \cdot 10^{-13} \\ 0,00 & 0,00 & 0,00 & 0,00 & 0,00 & 0,00 \end{bmatrix} \tag{3.22}$$

Para aplicar o método de Rabins será necessário armazenar quais regiões contêm correntes. É então armazenado um vetor no objeto *self.current_regions_indexes* que contém os índices das regiões que contêm fontes de corrente. No caso do exemplo, o vetor seria de dimensão 2 tendo como entradas os valores 1 e 3.

Após chamar a função *generate_regions*, o próximo passo da preparação do método é criar as variáveis $r$ e $x$ da dedução de Rabins. Na dedução, $x$ não mais é a posição cartesiana, mas sim uma mudança de variável de $r$ para transformar a lei de Ampère em coordenadas cilíndricas na equação de Bessel. Para que se chegue ao raio $r$ basta mover do vértice inferior esquerdo da janela para o centro do núcleo ferromagnético a origem do plano de coordenadas. Esta mudança do $x$ cartesiano para $r$ pode ser feita através da equação (3.23). A equação (3.24) demonstra a mudança de

variável de $r$ para $x_{Bessel}$ realizada por Rabins.

$$r = x + r_{core} \tag{3.23}$$

$$x_{Bessel} = m\,r, \qquad m = \frac{n\,\pi}{y_0}, \qquad n = 1,\, 2,\, 3,\, ..,\, \infty \tag{3.24}$$

Por questões de legibilidade do código e deste trabalho, para que se evite confusões com o $x$ do plano cartesiano, se referirá à variável $x$ da Função de Bessel como a sua definição $m\,r$.

Reescreve-se então as posições de interface entre regiões em termos de raios em um vetor *regions_r*. Para o exemplo, o vetor *regions_r* pode ser escrito conforme a equação (3.25). Criam-se vetores auxiliares $n$ contendo os números naturais de 1 ao $n$ máximo de entrada e $m$ que é o próprio vetor $n$ multiplicado pela constante $\pi/y_0$. As equações (3.26) e (3.27) definem os vetores $n$ e $m$ para o exemplo com $n_{max} = 5$.

$$r = \begin{bmatrix} 0,05 & 0,1 & 0,15 & 0,2 & 0,25 & 0,45 \end{bmatrix} \tag{3.25}$$

$$n = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix} \tag{3.26}$$

$$m = \begin{bmatrix} \frac{\pi}{0,5} & \frac{2\pi}{0,5} & \frac{3\pi}{0,5} & \frac{4\pi}{0,5} & \frac{5\pi}{0,5} \end{bmatrix} = \begin{bmatrix} 6,28 & 12,57 & 18,85 & 25,13 & 31,42 \end{bmatrix} \tag{3.27}$$

A partir da multiplicação matricial de *regions_r* transposto por $m$, chega-se a uma matriz $6 \times 5$ cuja linha $i$ contém os valores de $m\,r$ para o raio da i-ésima interface entre regiões. Esta matriz é armazenada no objeto *self.regions_mr* conforme a equação (3.28). Em especial, o vetor que forma a primeira linha da matriz é armazenado no objeto *self.mr_core*.

$$regions\_mr = regions\_r^{\,t} \cdot m = \begin{bmatrix} 0,05 \\ 0,1 \\ 0,15 \\ 0,2 \\ 0,25 \\ 0,45 \end{bmatrix} \cdot \begin{bmatrix} 6,28 & 12,57 & 18,85 & 25,13 & 31,42 \end{bmatrix} = \begin{bmatrix} 0,31 & 0,63 & 0,94 & 1,26 & 1,57 \\ 0,63 & 1,26 & 1,88 & 2,51 & 3,14 \\ 0,94 & 1,88 & 2,83 & 3,77 & 4,71 \\ 1,26 & 2,51 & 3,77 & 5,03 & 6,28 \\ 1,57 & 3,14 & 4,71 & 6,28 & 7,85 \\ 2,83 & 5,65 & 8,48 & 11,31 & 14,14 \end{bmatrix} \tag{3.28}$$

O programa inicializa em zero uma variável $2\,W$ que eventualmente armazenará o valor do dobro da energia do campo magnético disperso no transformador. Inicia-se um *loop* por todas as regiões que contêm fontes de corrente para o cálculo da contribuição de cada região para $2\,W$.

A cada iteração, utiliza-se uma função auxiliar *self.rabins_region* para calcular os coeficientes $P$, $Q$, $R$ e os vetores de coeficientes $S_n$, $T_n$ e $U_n$ vistos pelo enrolamento $i$ conforme as equações (3.29) e (3.30).

$$\begin{aligned} P^{(i)} &= -\mu_0 \frac{J_0^{(i)}}{3} \\ Q^{(i)} &= \mu_0 \left( \frac{J_0^{(i)}\, r_e^{(i)}}{2} + \sum_{j=i+1}^{k} \frac{J_0^{(j)} \left( r_e^{(j)} - r_i^{(j)} \right)}{2} \right) \\ R^{(i)} &= \mu_0 \left( \sum_{j=1}^{i-1} \frac{J_0^{(j)} \left( r_e^{3(j)} - r_i^{3(j)} \right)}{6} - \frac{J_0^{(i)} r_i^{3(i)}}{6} \right) \end{aligned} \tag{3.29}$$

$$\begin{aligned} S_n^{(i)} &= \mu_0 \frac{1}{m^2} \left( J_n^{(i)}\, E_n^{(i)} + \sum_{j=i+1}^{k} J_n^{(j)}\, C_n^{(j)} \right) \\ T_n^{(i)} &= \mu_0 \frac{1}{m^2} \left( \sum_{j=1}^{i-1} J_n^{(j)}\, G_n^{(j)} + J_n^{(i)}\, F_n^{(i)} + \sum_{j=i+1}^{k} J_n^{(j)}\, D_n^{(j)} \right) \\ U_n^{(i)} &= -\mu_0 \frac{1}{m^2} \frac{\pi}{2} J_n^{(i)} \end{aligned} \tag{3.30}$$

O método *rabins_region* começa inicializando em zero os coeficientes $P$, $Q$ e $R$, bem como os vetores de $n$ entradas $S_n$, $T_n$ e $U_n$ para o enrolamento $i$. Como se pode observar, os enrolamentos $j$ mais externos que o enrolamento estudado ($j > i$) contribuem com incrementos no número real $Q$ e nos vetores $S_n$ e $T_n$, estes incrementos podem ser somados a $Q$, $S_n$ e $T_n$ através de um *loop* que passe pelos enrolamentos $i+1$ a $k$. Na sequência, as variáveis $P$, $Q$ e $R$, $S_n$, $T_n$ e $U_n$ recebem seus incrementos por conta da contribuição de potencial magnético do próprio enrolamento $i$. Por fim, realiza-se mais um *loop* para calcular os incrementos nas variáveis $R$ e $T_n$ causados por enrolamentos mais próximo ao núcleo que $i$. As implementações estão de acordo com as equações (3.29) e (3.30).

Os cálculos das constantes $C_n$, $D_n$, $E_n$, $F_n$ e $G_n$ foram implementados em métodos separados em função das funções de Bessel, Struve e suas integrais implementadas no início do programa. Verifica-se a importância do recurso da biblioteca *scipy* e da implementação que as funções automaticamente se distribuam por vetores. Caso isto

não fosse possível, seriam necessários *loops* de 1 a $n$ para o cálculo de cada um dos vetores $C_n$, $D_n$, $E_n$, $F_n$ e $G_n$.

Retorna-se ao cálculo da energia magnética com $P$, $Q$ e $R$, $S_n$, $T_n$ e $U_n$ já computados tendo em vista o enrolamento $i$, basta calcular então a contribuição da fonte de corrente $i$ para a energia. Na implementação divide-se a energia entre o número real $2\,W_0$ e o vetor de $n$ entradas $2\,W_n$ conforme a equação (3.31).

$$\begin{aligned} 2\,W_0^{(i)} &= J_0^{(i)}\,2\,\pi\,L\int_{r_i^{(i)}}^{r_e^{(i)}}\left(P^{(i)}\,r^3+Q^{(i)}\,r^2+R^{(i)}\right)\,dr \\ 2\,W_n^{(i)} &= \pi\,L\,\frac{J_n^{(i)}}{m^2}\int_{mr_i^{(i)}}^{mr_e^{(i)}}\left(S_n^{(i)}\,mr\,I_1(mr)+T_n^{(i)}\,mr\,K_1(mr)+U_n^{(i)}\,mr\,L_1(mr)\right)\,d(mr) \end{aligned} \tag{3.31}$$

Com as integrais contendo as funções de Bessel e Struve já implementadas no início do programa e com a trivialidade das integrais polinomiais, facilmente implementa-se no *loop* em $i$ a equação (3.31). Incrementando a cada iteração a energia total $2\,W$ pelo somatório das entradas de $2\,W_n(i)$ e pelo número real $2\,W_0(i)$. Ao final do *loop*, estará implementada a equação (3.32).

$$2\,W=\sum_{i=1}^{k}2\,W^{(i)}=\sum_{i=1}^{k}\left(2\,W_0^{(i)}+\sum_{n=1}^{n}2\,W_n^{(i)}\right) \tag{3.32}$$

Por fim, divide-se $2\,W$ pelo quadrado da corrente de entrada e chega-se ao valor da indutância de dispersão pelo método de Rabins que é armazenada no objeto *self.L_Rabins*.

#### 3.2.7.1 Métodos de computação de $J(x,y)$, $A(x,y)$ e $B(x,y)$ pelo método de Rabins.

Cada um dos métodos inicializa matrizes que receberão os valores das grandezas em cada um dos pontos do domínio. O domínio é discretizado em 250 passos em $y_0$ e, em $x$, em um número de passos proporcional a $x_0$.

O cálculo de $J(x,y)$ é feito através do método *calc_J_xy_Rabins*. O valor de $J$ é calculado em função de $y$ para cada uma das regiões através das séries de Fourier armazenadas em *Jn_regions*. Cada posição de $x$ recebe então os valores de $J$ referentes à região a que essa posição pertence. A matriz é, então, armazenada no objeto *J_xy_Rabins*.

Os cálculos de $A(x,y)$ e $B(x,y)$ são feitos através dos métodos *calc_A_xy_Rabins* e *calc_B_xy_Rabins*. Nestes métodos são feitos *loops* que passão por todas as regiões do domínio. A cada iteração, são calculadas as constantes $P$, $Q$, $R$, $S_n$, $T_n$ e $U_n$ para a região $i$ e calculados os valores de $A$, $B_x$ e $B_y$ nos pontos do domínio que pertencem à i-ésima região através das equações (3.33), (3.34) e (3.35). A matriz contendo os valores de $A$

em todo o domínio é armazenada no objeto *A_xy_Rabins*. Os valores de $B_x$ e $B_y$ são armazenados nos objetos *Bx_xy_Rabins* e *By_xy_Rabins* e o módulo de $B$ armazenado em *B_xy_Rabins*.

$$\begin{aligned} A^{(i)}(x,y) =& P^{(i)}\, r^2 + Q^{(i)}\, r + R^{(i)}\, \frac{1}{r} + \\ & \sum_{n=1}^{\infty} \left( S_n^{(i)}\, I_1(m\, r) + T_n^{(i)}\, K_1(m\, r) + U_n^{(i)}\, L_1(m\, r) \right) cos\,(m\, y) \end{aligned} \tag{3.33}$$

$$B_x^{(i)}(x,y) = \sum_{n=1}^{\infty} m\, \left( S_n^{(i)}\, I_1(m\, r) + T_n^{(i)}\, K_1(m\, r) + U_n^{(i)}\, L_1(m\, r) \right) sin\,(m\, z) \tag{3.34}$$

$$\begin{aligned} B_y^{(i)}(x,y) =& 3\, P^{(i)}\, r \\ & + 2\, Q^{(i)} \\ & + \sum_{n=1}^{\infty} \left( S_n^{(i)}\, I_1(m\, r) + S_n^{(i)} \frac{m\, r}{2}\, (I_0(m\, r) + I_2(m\, r)) \right. \\ & + T_n^{(i)}\, K_1(m\, r) - T_n^{(i)} \frac{m\, r}{2}\, (K_0(m\, r) + K_2(m\, r)) \\ & \left. + U_n^{(i)}\, L_1(m\, r) + U_n^{(i)} \frac{m\, r}{2} \left( L_0(m\, r) + L_2(m\, r) + \frac{2\, m\, r}{3\pi} \right) \right) cos\,(m\, z)\, \frac{1}{r} \end{aligned} \tag{3.35}$$

### 3.2.8 Métodos para plotar os mapas dos campos.

Um dos objetivos deste trabalho é permitir que o projetista de transformador visualize os campos conforme altera o seu projeto. Por este motivo, a cada passo do desenvolvimento do código, foram implementados métodos de coleta dos valores que os campos $J$, $A$, $B_x$, $B_y$ e $|\mathbf{B}|$ assumem no domínio da janela. Neste trecho final de código, são escritos métodos que permitem ao usuário a plotagem destes campos.

Todos os métodos de plotagem apresentados utilizam como base o método *plot_heatmap_generic* que recebe como entradas a matriz com os valores que o campo a ser plotado assume em cada ponto do domínio *field_xy*, uma rotulação opcional *label* para identificar as unidades da legenda do gráfico e um nome opcional para que se salve o gráfico como imagem.

O método é iniciado inserindo as dimensões da plotagem respeitando as dimensões da janela. É então utilizada a função *imshow* da *matplotlib* com a entrada do mapeamento de cores setada em *jet*, mapeando valores baixos em azul e altos em vermelho. É inserido o rótulo da legenda e são setadas as cotas de início e fim da janela. Por

fim, é implementada a possibilidade de salvar a imagem em formato *portable network graphics* com alta qualidade.

Cada um dos demais métodos do tipo *plot_heatmap* é, por fim, implementado recorrendo ao *plot_heatmap_generic*, inserindo como matriz de entrada o campo a que se refere e como rótulo a unidade deste campo.

Como pode-se observar, a biblioteca não permite que se plote campos vetoriais de maneira direta. Somente plotando os campos escalares dos módulos dos vetores $J$, $A$ e $B$ ou das componentes $B_x$ e $B_y$ da indução magnética. Como expansão deste trabalho, em uma próxima revisão da biblioteca seria possível prever plotagens vetoriais, principalmente caso se implemente o cálculo das forças.

# 4 RESULTADOS OBTIDOS

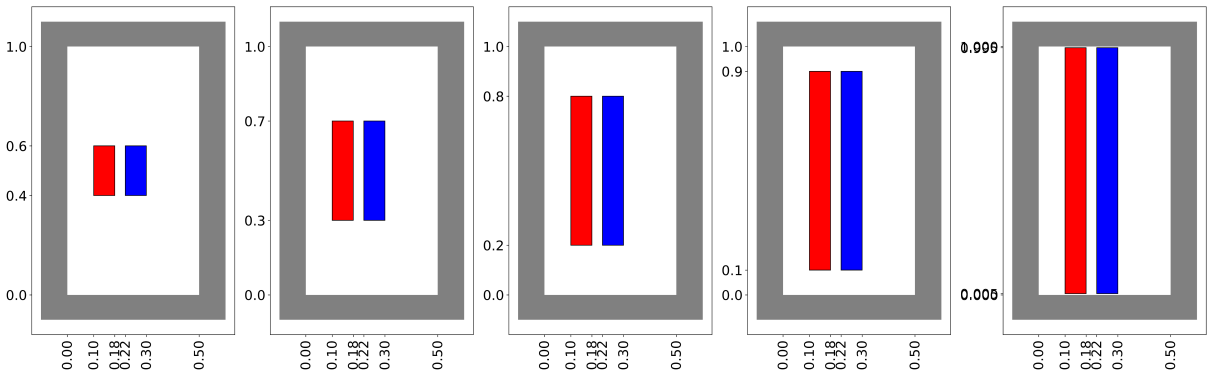
Neste capítulo serão propostas algumas geometrias de transformadores com o intuito de avaliar o desempenho da biblioteca e dos métodos analíticos na representação dos campos de indução magnética. O objetivo desta seção não somente é demonstrar o funcionamento dos métodos, mas também indicar onde eles falham em representar o problema.

Cada geometria gerada possuirá dois enrolamentos com mil espiras cada e os cálculos analíticos serão comparados com os resultados do método de elementos finitos axial com ar à direita da janela (suposição de Rabins). Os desvios percentuais nos valores de indutância serão calculados em função deste método. Serão simuladas também as geometrias com elementos finitos planares e material ferromagnético preenchendo a perna à direita da janela (suposição de Roth).

## 4.1 TRANSFORMADOR DE DOIS ENROLAMENTOS TIPO SANDUÍCHE DE MESMA ALTURA

Nesta seção, será instanciado o transformador com geometria mais simples com dois enrolamentos tipo sanduíche de seção retangular próximos. O esperado com esta geometria é que as aproximações de Roth e Rabins se mostrem ambas válidas. A altura de ambos enrolamentos será variada de 20% a pouco menos de 100% da altura da janela conforme a figura 27. Os resultados das indutâncias calculadas estão apresentados na tabela 1.

Figura 27 – Geometrias propostas de transformadores de dois enrolamentos tipo sanduíche de mesma altura.



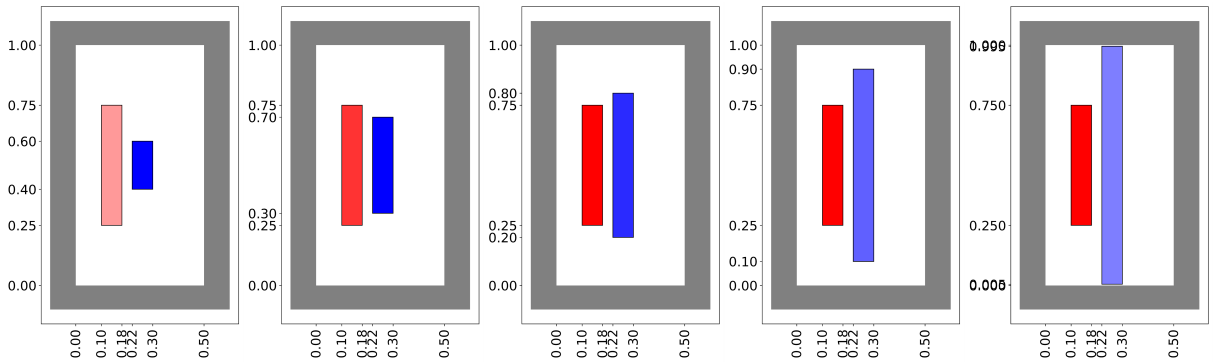
Fonte: O autor (2021).

Como se pode verificar na tabela 1, os dois métodos analíticos chegam a resultados satisfatórios de indutância. O campo magnético tende a se confinar entre os

Tabela 1 – Indutâncias das geometrias propostas de transformadores de dois enrolamentos tipo sanduíche de mesma altura. A altura do enrolamento está descrita como uma fração da altura da janela.

| Altura | Femm axial | Femm planar | Roth | Rabins | Desvio Roth | Desvio Rabins |
|---|---|---|---|---|---|---|
| 0,20 | 619,54 mH | 630,8 mH | 639,27 mH | 625,79 mH | 3,18% | 1,01% |
| 0,40 | 400,15 mH | 405,96 mH | 413,12 mH | 406,63 mH | 3,24% | 1,62% |
| 0,60 | 296,58 mH | 296,89 mH | 306,43 mH | 303,55 mH | 3,32% | 2,35% |
| 0,80 | 242,43 mH | 242,33 mH | 247,22 mH | 246,37 mH | 1,98% | 1,63% |
| 0,99 | 217,8 mH | 216,23 mH | 221,2 mH | 221,17 mH | 1,56% | 1,54% |

Figura 28 – Geometrias propostas de transformadores de dois enrolamentos tipo sanduíche de alturas diferentes.



Fonte: O autor (2021).

enrolamentos em uma largura muito pequena da janela, sendo válida a aproximação de Roth de curvatura desprezível. A maior limitação do método de Rabins é a representação da corrente em séries de Fourier com poucas componentes, neste caso, a representação com 15 harmônicas é suficiente por não acontecerem muitas transições bruscas da corrente em $y$.

Mesmo que simples, este tipo de geometria aproxima muito bem a maior parte dos transformadores. As seções que seguem têm como objetivo levar os modelos aos seus limites, falhas dos modelos nos testes seguintes não devem ser interpretadas como impeditivas para seu uso.

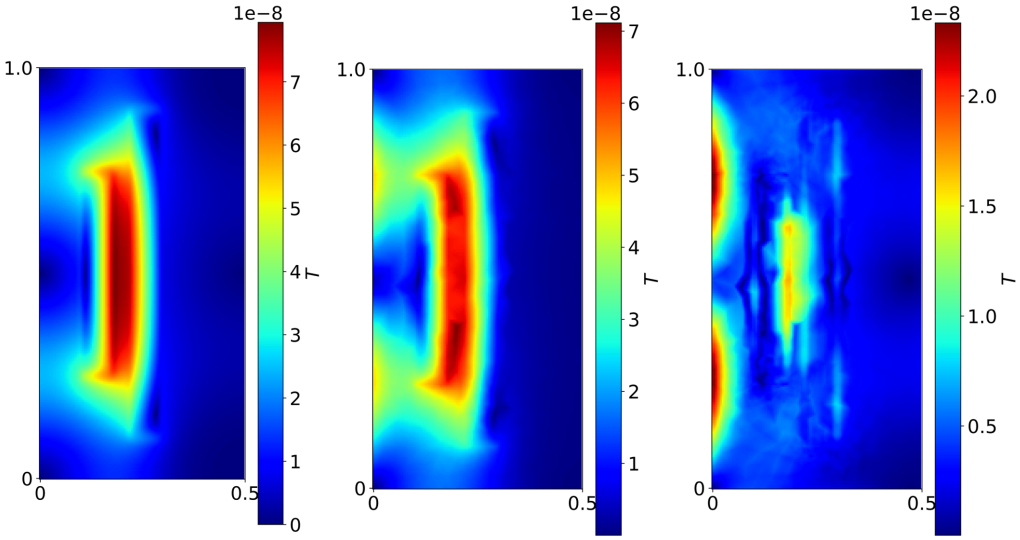
## 4.2 TRANSFORMADOR DE DOIS ENROLAMENTOS TIPO SANDUÍCHE DE ALTURAS DIFERENTES

Nesta seção, será instanciado o transformador com enrolamentos tipo sanduíche de seção retangular, próximos variando a altura de um deles. A altura do outro enrolamento é mantida constante em metade da altura da janela. Deve-se esperar que ambos os métodos desempenhem bem, lembrando que este problema foi uma das motivações de Roth ao aprimorar o método de Rogowski. As geometrias utilizadas estão representadas pela figura 28 e os resultados na tabela 2.

Tabela 2 – Indutâncias das geometrias propostas de transformadores de dois enrolamentos tipo sanduíche de alturas diferentes. A altura do enrolamento está descrita como uma fração da altura da janela.

| Altura | Femm axial | Femm planar | Roth | Rabins | Desvio Roth | Desvio Rabins |
|---|---|---|---|---|---|---|
| 0,2 | 566,94 mH | 530,75 mH | 540,44 mH | 572,46 mH | 4,67% | 0,97% |
| 0,4 | 383,84 mH | 378,54 mH | 385,23 mH | 389,82 mH | 0,36% | 1,56% |
| 0,6 | 317,82 mH | 330,39 mH | 337,09 mH | 324,82 mH | 6,06% | 2,2% |
| 0,8 | 314,13 mH | 338,29 mH | 345,51 mH | 319,37 mH | 9,99% | 1,67% |
| 0,99 | 340,45 mH | 374,81 mH | 382,28 mH | 345,84 mH | 12,29% | 1,58% |

Figura 29 – Comparação entre os campos gerados pelo método de Roth (à esquerda), de elementos finitos (ao centro) e o desvio absoluto entre eles (à direita).




Fonte: O autor (2021).

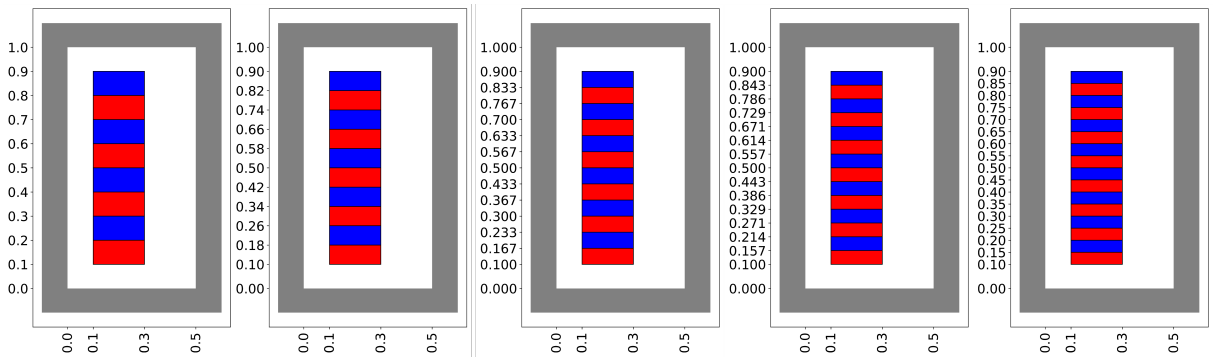
Constata-se que Roth erra ao estimar a indutância de dispersão quando o enrolamento mais afastado do núcleo é maior que o mais próximo. Observa-se na figura 29 a indução magnética no caso $h = 0,8$ conforme Roth, o *Femm* e o módulo do desvio entre os dois. Nota-se que a estimativa de Roth próxima ao núcleo erra por desconsiderar a curvatura da geometria. Os desvios do método de Rabins seguem satisfatórios.

## 4.3 TRANSFORMADOR DE DOIS ENROLAMENTOS TIPO PANQUECA COMPOSTOS POR $k$ CAMADAS CADA

Nesta seção, será instanciado o transformador com enrolamentos tipo panqueca de seção retangular variando o número de camadas da panqueca conforme a figura 30. O objetivo deste teste é levar a representação da série de Fourier da corrente de Rabins ao seu limite. Espera-se que a partir de certo ponto, não será possível representar a corrente na janela com somente 15 harmônicas em $y$. O método de Roth, no entanto,

não deve errar desde que o campo se confine no volume próximo aos enrolamentos. Os resultados são apresentados na tabela 3.

Figura 30 – Geometrias propostas de transformadores de dois enrolamentos tipo panqueca compostos por $k$ camadas cada.



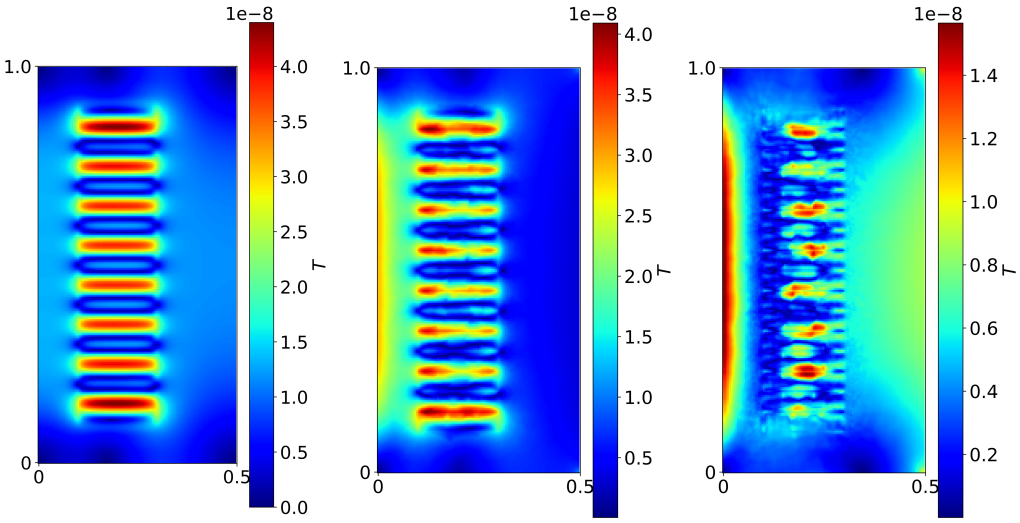
Fonte: O autor (2021).

Tabela 3 – Indutâncias das geometrias propostas de transformadores de dois enrolamentos tipo panqueca compostos por $k$ camadas cada.

| k | Femm axial | Femm planar | Roth | Rabins | Desvio Roth | Desvio Rabins |
|---|---|---|---|---|---|---|
| 4 | 92,36 mH | 105,56 mH | 107,72 mH | 95,49 mH | 16,63% | 3,39% |
| 5 | 59,98 mH | 68,54 mH | 70,24 mH | 62,47 mH | 17,1% | 4,14% |
| 6 | 42,53 mH | 47,75 mH | 49,38 mH | 43,36 mH | 16,13% | 1,96% |
| 7 | 31,13 mH | 35,08 mH | 36,43 mH | 19,11 mH | 17,01% | 38,61% |
| 8 | 23,61 mH | 27,04 mH | 28,06 mH | 14,29 mH | 18,83% | 39,47% |

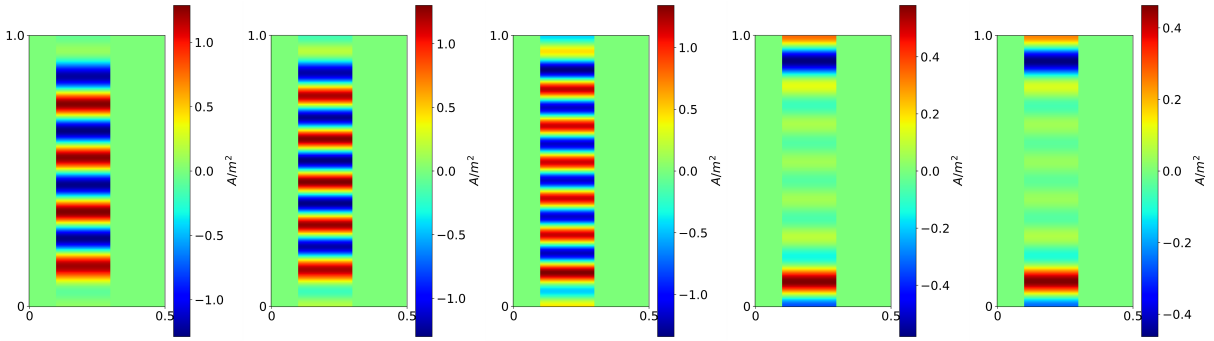
Observou-se que o campo não se confina à região próxima aos enrolamentos, mas sim escapa para o núcleo, como se pode verificar na figura 31 com $k = 8$. O método de Roth, portanto apresenta desempenho abaixo do satisfatório. Verifica-se que o método de Rabins passa a errar muito a partir de $k = 7$, a ponto de apresentar resultados piores que o método de Roth. Em $k = 7$ a série de Fourier com 15 componentes passa a não representar fielmente as correntes na janela. A figura 32 reconstrói, a partir da série de Fourier, a corrente considerada pelo método de Rabins para cada valor de $k$.

Figura 31 – Comparação entre os campos gerados pelo método de Roth (à esquerda), de elementos finitos (ao centro) e o desvio absoluto entre eles (à direita).




Fonte: O autor (2021).

Figura 32 – Representação da série de Fourier da corrente do método de Rabins em enrolamento panqueca com o número de camadas por enrolamento variando de quatro a oito.




Fonte: O autor (2021).

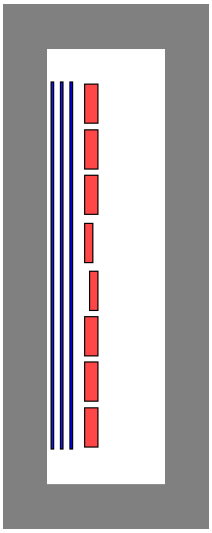
## 4.4 VALIDAÇÃO DOS MÉTODOS ANALÍTICOS COM RESULTADOS DE MEDIÇÃO.

Nesta seção, os resultados dos métodos analíticos apresentados neste TCC serão comparados com um resultado medido. A figura 33 mostra a geometria da janela de um transformador obtido com um fabricante de transformador localizado no Brasil. Nota-se que a baixa tensão é composta por 3 enrolamentos ligados em série e que a alta tensão é composta por 8 panquecas ligadas em série. Por questões de sigilo, as dimensões, os números de espiras e os dados de placa do transformador não serão apresentados.

A tabela 4 apresenta as indutâncias calculadas dos enrolamentos de BT e AT, as reatâncias percentuais dos equivalentes monofásicos do transformador calculadas a partir dos dados da AT e da BT conforme a equação (4.1), a reatância percentual do

enrolamento de BT medida pelo fabricante e o erro percentual nas reatâncias calculadas em relação à reatância medida.

Figura 33 – Esboço da geometria da janela do transformador ensaiado.



Fonte: O autor (2021).

$$X\% = 2\,\pi\,f\,L_Y\,\frac{S_{3\phi}}{V_l^2}\cdot 100\,\% = 2\,\pi\,f\,\frac{L_\Delta}{3}\,\frac{S_{3\phi}}{V_l^2}\cdot 100\,\% \tag{4.1}$$

Onde $f$ é a frequência nominal do sistema, $L_Y$ é a indutância de dispersão de um enrolamento do transformador caso os enrolamentos estejam ligados em Y, $L_\Delta$ é a indutância de dispersão de um enrolamento do transformador caso os enrolamentos estejam ligados em $\Delta$, $S_{3\phi}$ é a potência nominal do transformador, $V_l$ é a tensão de linha do sistema e $X\%$ é a reatância percentual do equivalente monofásico do transformador.

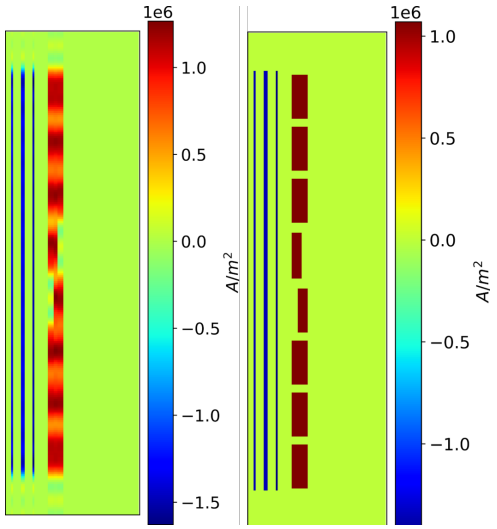
Tabela 4 – Resultados obtidos na aplicação dos métodos estudados comparados ao valor medido.

| | Medição | Femm axial | Femm planar | Roth | Rabins |
|---|---|---|---|---|---|
| Indutância da BT | - | 23,01 $\mu H$ | 23,33 $\mu H$ | 23,51 $\mu H$ | 23,14 $\mu H$ |
| Indutância da AT | - | 82,15 $mH$ | 83,30 $mH$ | 83,94 $mH$ | 82,62 $mH$ |
| Reatância da BT | 6,16 % | 6,01 % | 6,09 % | 6,14 % | 6,04% |
| Reatância da AT | - | 6,01 % | 6,09 % | 6,14 % | 6,04% |
| Erro na reatância da BT | - | -2,49 % | -1,12 % | -0,36 % | -1,93% |

Pode-se verificar na tabela 4 que os resultados obtidos em todos os métodos foram satisfatórios quando comparados aos dados de medição, visto que cada método apresenta erro ao estimar a reatância de dispersão do transformador inferior a um terço dos 7,5% previstos em norma.

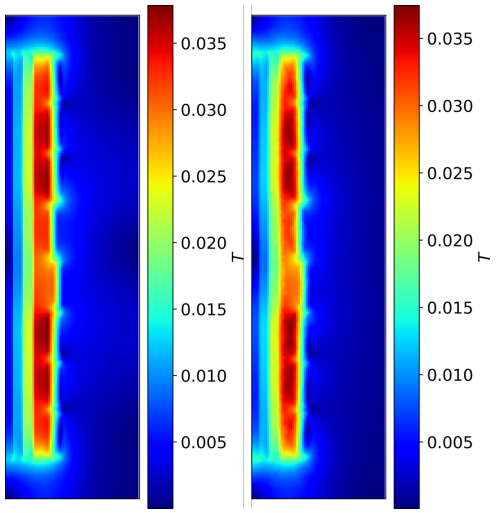
No método de Rabins a razão entre a altura da janela e o raio externo do enrolamento de AT permite que se utilize 35 componentes na série de Fourier, gerando excelente aproximação para as fontes de corrente na janela. Na figura 34 observa-se a interpretação que o método de Rabins dá às correntes na janela em comparação às correntes de entrada do método de elementos finitos.

Figura 34 – Mapa da densidade de corrente na janela do transformador ensaiado conforme entrada do método de Rabins (à esquerda) e conforme entrada do método de elementos finitos (à direita).




Fonte: O autor (2021).

Figura 35 – Mapa do módulo da indução magnética na janela do transformador ensaiado conforme cálculos via método de Roth (à esquerda) e método de elementos finitos axissimétricos (à direita).




Fonte: O autor (2021).

O método de Roth tem desempenho ainda melhor que o de Rabins por conta do campo magnético se mostrar confinado a uma faixa pequena de valores de raio, garan-

tindo a aproximação da geometria do transformador por coordenadas retangulares. A figura 35 mostra a indução magnética na janela conforme o método de Roth comparada à indução magnética no método de elementos finitos axissimétrico.

É válido lembrar que os dois métodos analíticos consideram que o núcleo é fabricado com um material de permeabilidade magnética infinita e as simulações foram realizadas considerando o núcleo com alta permeabilidade. Tanto os métodos analíticos quanto as simulações não consideram a saturação do núcleo nos cálculos.

# 5 CONCLUSÃO

Métodos numéricos como o método de elementos finitos são utilizados para cálculo de reatâncias, porém exigem tempo de processamento muito superior à computação de uma formulação analítica. Por este motivo, modelos analíticos são amplamente utilizados em projetos de transformadores. Com o desenvolvimento deste trabalho, foi possível escrever uma ferramenta computacional que aplica dois desses métodos analíticos a uma geometria arbitrária, bem como estudar as aplicações e as limitações de cada método.

O método de Roth resolve com facilidade problemas com qualquer configuração de enrolamentos por conta da sua representação da corrente em série dupla de Fourier. Porém tem a sua efetividade muito diminuída quando o campo magnético disperso se espalha pela janela. A sua resolução da lei de Ampère em coordenadas cartesianas gera erros em um problema de simetrias cilíndricas.

O método de Rabins é extremamente versátil e não foi encontrada nenhuma geometria em que o método teórico se afaste do valor de indutância encontrado numericamente. A grande limitação desta ferramenta para a solução da lei de Ampère é o uso das funções de Bessel e Struve. Essas funções apresentam problemas de convergência para argumentos grandes, o que inviabiliza o detalhamento das fontes de corrente do transformador em grande número de termos da série de Fourier. Neste trabalho, a implementação utilizada das funções de Bessel e Struve permite somente que se utilize com confiabilidade cerca de 20 termos para detalhar a corrente, o que não é suficiente caso haja necessidade de representar diversas transições de corrente em $y$.

Primeiramente os cálculos foram comparados a resultados obtidos utilizando a ferramenta de elementos finitos *Femm*. Nas análises foram utilizados, não somente os valores de indutância de dispersão, mas também gráficos em duas dimensões dos campos vetoriais de interesse para um entendimento mais preciso da física das geometrias estudadas. Em seguida, os métodos apresentados no TCC foram validados a partir da comparação com um resultado de medição. Apresentando valor satisfatório de exatidão conforme norma.

Citam-se as seguintes sugestões para trabalhos futuros:

- Expandir a biblioteca com os demais itens de projeto de transformadores, forças nos enrolamentos de baixa e alta tensão, cálculo de capacitâncias e campo elétrico, perdas por correntes induzidas, etc.

- Realizar em coordenadas cilíndricas a integração da densidade de energia re-

sultante do trabalho de Roth. A tentativa libertaria o usuário da necessidade de estimar um comprimento médio para a circunferência do transformador no método de Roth. Foi realizada a integração numérica em coordenadas cilíndricas da energia densidade do campo magnético nos casos teste em que o método de Roth não performa bem. A abordagem gerou frutos no caso de enrolamentos tipo sanduíche de alturas diferentes, porém não apresentou melhoria no caso dos enrolamentos de tipo panqueca.

- Confrontar os modelos analíticos com dados de simulações de elementos finitos em três dimensões e com mais ensaios em transformadores reais para validá-los.

## REFERÊNCIAS


BOYAJIAN, A. Leakage reactance of irregular distributions of transformer windings by the method of double fourier series [includes discussion]. *Transactions of the American Institute of Electrical Engineers. Part III: Power Apparatus and Systems*, v. 73, n. 2, p. 1078–1086, 1954. Citado na página 32.

DANIELS, A. *Introduction to Electrical Machines*. [S.l.]: Macmillan Education UK, 1976. (Electrical and electronic engineering). ISBN 9781349156894. Citado na página 28.

HARRIS, C. R. et al. Array programming with NumPy. *Nature*, v. 585, p. 357–362, 2020. Citado na página 43.

HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007. Citado na página 43.

IEEE Standard for General Requirements for Liquid-Immersed Distribution, Power, and Regulating Transformers. *IEEE Std C57.12.00-2010 (Revision of IEEE Std C57.12.00-2006)*, p. 1–70, 2010. Citado na página 22.

KAPP, G. *Transformatoren für Wechselstrom und Drehstrom: eine Darstellung ihrer Theorie, Konstruktion und Anwendung*. [S.l.]: Julius Springer, 1900. (Nineteenth Century Collections Online (NCCO): Science, Technology, and Medicine: 1780-1925). ISBN 9783486730944. Citado 2 vezes nas páginas 28 e 29.

MAXWELL, J. *A Treatise on Electricity and Magnetism*. [S.l.]: Clarendon Press, 1873. (A Treatise on Electricity and Magnetism, v. 1). Citado na página 25.

MEEKER, D. *Finite Element Method Magnetics: pyFEMM*. [S.l.], 2018. Citado 3 vezes nas páginas 43, 54 e 55.

RABINS, L. Transformer reactance calculations with digital computers. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, v. 75, n. 3, p. 261–267, 1956. Citado na página 35.

REYNDERS, J. The prediction of fault currents in a large multiwinding reactor transformer. In: *2003 IEEE Bologna Power Tech Conference Proceedings,*. [S.l.: s.n.], 2003. v. 2, p. 5. Citado na página 22.

RIBEIRO, G. R. Cálculo de reatância de dispersão em transformadores usando o método de rabin e o método de elementos finitos. *Computing in Science & Engineering*, UFSC, 2014. Citado na página 23.

ROGOWSKI, W. *Ueber das Streufeld und den Streuinduktionskoeffizienten eines Transformators mit Scheibenwicklung und geteilten Endspulen*. Tese (Doutorado) — Technischen Hochschule zu Danzig, 1908. Citado 2 vezes nas páginas 29 e 30.

ROTH, E. *Introduction à l'étude analytique de l'échauffement des machines électriques*. [S.l.]: Étienne Chiron, Éditeur, 1927. Citado na página 32.

VECCHIO, R. et al. *Transformer Design Principles*. [S.l.]: CRC Press, 2002. Citado na página 21.

VIRTANEN, P. et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, v. 17, p. 261–272, 2020. Citado na página 43.

ZAKAREVICIUS, R. A. Calculation of the switching-on overvoltages in a series-connected thyristor string. *IEEE Transactions on Industry Applications*, IA-13, n. 5, p. 407–417, 1977. Citado na página 22.

# APÊNDICE A - CÓDIGO FONTE DA FERRAMENTA COMPUTACIONAL IMPLEMENTADA.

```
#3.2.1 Inicializa  o do C digo

import numpy as np
from numpy import pi as pi
from numpy import cos as cos
from numpy import sin as sin
from numpy import exp as exp

import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle

import femm

from scipy.special import k0 as K0
from scipy.special import k1 as K1
from scipy.special import kn
from scipy.special import i0 as I0
from scipy.special import i1 as I1
from scipy.special import iv
from scipy.special import iti0k0
from scipy.special import modstruve
from scipy.special import itmodstruve0

u0 = (4e-7)*pi

def K2(x):
    return kn(2,x)
def I2(x):
    return iv(2,x)
def L0(x):
    return modstruve(0,x)
def L1(x):
    return modstruve(1,x)
def L2(x):
    return modstruve(2,x)

def integral_t_K1_0(x):
    if type(x) == type(np.array([0])):
        __it = np.zeros(x.shape)
        __it[x!=0] = iti0k0(x)[1]-x*K0(x)
    else:
        if(x==0):
            __it = 0
        else:
            __it = iti0k0(x)[1]-x*K0(x)
    return __it

def integral_t_K1(x1,x2):
    return integral_t_K1_0(x2) - integral_t_K1_0(x1)

def integral_t_I1_0(x):
    return x*I0(x)-iti0k0(x)[0]
```

```
def integral_t_I1(x1,x2):
    return integral_t_I1_0(x2)-integral_t_I1_0(x1)

def integral_t_L1_0(x):
    return x*L0(x)-(x**2)/pi-itmodstruve0(x)

def integral_t_L1(x1,x2):
    return integral_t_L1_0(x2)-integral_t_L1_0(x1)

#3.2.2 Inicializa  o da classe transformer
class transformer:
    def __init__(self, x0, y0, r_core = ''):

        self.x0 = x0
        self.y0 = y0
        if r_core=='':
            self.r_core = y0/5
        else:
            self.r_core = r_core
        self.x1 = np.array([])
        self.x2 = np.array([])
        self.y1 = np.array([])
        self.y2 = np.array([])
        self.J  = np.array([])
        self.I_balance = 0.0
        self.number_of_windings = 0

    #3.2.3 M todo de inclus o de enrolamento add_winding
    def add_winding(self, x1, y1, x2, y2, J,
                    repet_x = 1, Delta_x = 0, repet_y = 1, Delta_y = 0):

        for __j in range (repet_x):
            for __i in range (repet_y):
                self.x1 = np.append(self.x1, np.array([x1 + __j*(Delta_x+x2-x1)]))
                self.x2 = np.append(self.x2, np.array([x2 + __j*(Delta_x+x2-x1)]))
                self.y1 = np.append(self.y1, np.array([y1 + __i*(Delta_y+y2-y1)]))
                self.y2 = np.append(self.y2, np.array([y2 + __i*(Delta_y+y2-y1)]))
                self.J  = np.append(self.J , np.array([J]))
                self.I_balance += J*(x2-x1)*(y2-y1)
                self.number_of_windings += 1
        if (self.I_balance < 1e-10)*(self.I_balance > -1e-10):
            self.I_balance = 0

    #3.2.4 M todo do esbo o da janela draw_window
    def draw_window (self):

        #define Matplotlib figure and axis
        fig, ax = plt.subplots(figsize=((self.x0+self.r_core)*10/(self.y0+self.r_core),10))
        #add iron
        ax.add_patch(Rectangle((-self.r_core,-self.r_core),self.x0+2*self.r_core,
                               self.y0+2*self.r_core,facecolor = 'grey', fill=True))
        #add air
        ax.add_patch(Rectangle((0,0),self.x0,self.y0, facecolor = 'white', fill=True))

        #add windings
        __J_max = max(np.max(self.J),-np.min(self.J))
        __J_red = (self.J>0).astype(int)
        __J_blue = (self.J<0).astype(int)
        __J_alpha = (self.J>0)*self.J/__J_max+(self.J<0)*self.J/-__J_max
```

```
        for i in range (self.number_of_windings):
            ax.add_patch(Rectangle((self.x1[i], self.y1[i]),
                                   self.x2[i]-self.x1[i], self.y2[i]-self.y1[i],
                                   edgecolor = 'black',
                                  facecolor = (__J_red[i],0,__J_blue[i],__J_alpha[i]), fill=True))
        #display plot
        ax.set_xticks([0] + list(self.x1) + list(self.x2) + [self.x0])
        fig.autofmt_xdate(rotation=90, ha='center')
        ax.set_yticks([0] + list(self.y1) + list(self.y2) + [self.y0])
        ax.plot()

    #3.2.5 M todo de solu  o do problema por elementos finitos run_FEMM
    def run_FEMM(self, input_I, #[A]
                 simmetry = 'axi', #or 'planar'
                 p = 0,  #greater than 0 in case of planar
                 right_leg = 'steel', #or 'air'
                 close_FEMM = True, plot_generation = False):

        # Conversion lengths to mm with center leg offset
        __r_core = self.r_core*1000
        __x0 = self.x0*1000
        __y0 = self.y0*1000
        __x1 = self.x1*1000 + __r_core
        __x2 = self.x2*1000 + __r_core
        __y1 = self.y1*1000 + __r_core
        __y2 = self.y2*1000 + __r_core
        __depth = p*1000

        # Convert the current density to Amps turns
        __n = 1000
        __I = self.J*(self.x2-self.x1)*(self.y2-self.y1)/__n

        # The package must be initialized with the openfemm command.
        if close_FEMM:
            femm.openfemm(1)
        else:
            femm.openfemm()

        # We need to create a new Magnetostatics document to work on.
        femm.newdocument(0)

        # Define the problem type.  Magnetostatic; Units of mm; Axisymmetric;
        # Precision of 10^(-8) for the linear solver; a placeholder of 0 for
        # the depth dimension, and an angle constraint of 30 degrees
        femm.mi_probdef(0, 'millimeters', simmetry, 1.e-8, __depth, 30)

        # Add some block labels materials properties
        femm.mi_addmaterial('Air', 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0)
        femm.mi_addmaterial('Coil', 1, 1, 0, 0, 58*0.65, 0, 0, 1, 0, 0, 0)
        femm.mi_addmaterial('LinearIron', 2100, 2100, 0, 0, 0, 0, 0, 1, 0, 0, 0)

        # Steel
        femm.mi_drawrectangle(0, 0, __x0+ 2*__r_core, __y0+ 2*__r_core)
        femm.mi_addblocklabel(__r_core/2,__r_core/2)
        femm.mi_selectlabel(__r_core/2,__r_core/2)
        femm.mi_setblockprop('LinearIron', 0, 0, '<None>', 0, 0, 0)
        femm.mi_clearselected()

        #Right leg Air or Steel?
        femm.mi_drawline(__r_core+__x0,__r_core+__y0,__r_core+__x0,2*__r_core+__y0)
```

```
femm.mi_drawline(__r_core+__x0,            0,__r_core+__x0,  __r_core)
femm.mi_addblocklabel(__r_core+__x0+0.01,__r_core+0.01)
femm.mi_selectlabel  (__r_core+__x0+0.01,__r_core+0.01)
if right_leg == 'steel':
    femm.mi_setblockprop('LinearIron', 0, 0, '<None>', 0, 0, 0)
if right_leg == 'air':
    femm.mi_setblockprop('Air', 0, 0, '<None>', 0, 0, 0)
femm.mi_clearselected()

# Define an "open" boundary condition using the built-in function:
femm.mi_makeABC()

# Air
femm.mi_drawrectangle(__r_core, __r_core, __r_core+__x0, __r_core+__y0)
femm.mi_addblocklabel(__r_core+0.01,__r_core+0.01)
femm.mi_selectlabel(__r_core+0.01,__r_core+0.01)
femm.mi_setblockprop('Air', 0, 0, '<None>', 0, 0, 0)
femm.mi_clearselected()
femm.mi_addblocklabel(1,-1)
femm.mi_selectlabel(1,-1)
femm.mi_setblockprop('Air', 0, 0, '<None>', 0, 0, 0)
femm.mi_clearselected()

# Windings
for i in range(self.number_of_windings):
    femm.mi_drawrectangle(__x1[i], __y1[i], __x2[i], __y2[i])
    femm.mi_addcircprop('coil_'+str(i), __I[i], 1)
    femm.mi_addblocklabel((__x1[i]+__x2[i])/2,(__y1[i]+__y2[i])/2)
    femm.mi_selectlabel((__x1[i]+__x2[i])/2,(__y1[i]+__y2[i])/2)
    femm.mi_setblockprop('Coil', 0, 0, 'coil_'+str(i), 0, 0, __n);
    femm.mi_clearselected()

# Now, the finished input geometry can be displayed.
femm.mi_zoomnatural()

# We have to give the geometry a name before we can analyze it.
femm.mi_saveas('transformer.fem')

# Analysis.
femm.mi_analyze()
femm.mi_loadsolution()
femm.mo_groupselectblock()

# Inductance calculation.
__W_FEMM = femm.mo_blockintegral(2)
self.L_FEMM = 2*__W_FEMM/input_I**2

femm.mo_clearblock()

if plot_generation:
    #Calculate Jxy, Axy and Bxy
    __steps_y = 250
    __steps_x = int(__steps_y*self.x0/self.y0)
    __x = (np.linspace(0,self.x0,__steps_x)+self.r_core)*1000
    __y = (np.linspace(0,self.y0,__steps_y)+self.r_core)*1000
    __Jxy = np.zeros([__steps_y,__steps_x])
    __Axy = np.zeros([__steps_y,__steps_x])
    __Bx_xy = np.zeros([__steps_y,__steps_x])
    __By_xy = np.zeros([__steps_y,__steps_x])
```

```
            for __i in range(__steps_x):
                for __j in range(__steps_y):
                    __Jxy[__j,__i] = femm.mo_getj(__x[__i],__y[__j])
                    __Axy[__j,__i] = femm.mo_geta(__x[__i],__y[__j])
                    __Bx_xy[__j,__i] = femm.mo_getb(__x[__i],__y[__j])[0]
                    __By_xy[__j,__i] = femm.mo_getb(__x[__i],__y[__j])[1]


            __Bx_xy[0] = __Bx_xy[1]
            __Bx_xy[:,0] = __Bx_xy[:,1]
            __Bx_xy[-1] = __Bx_xy[-2]
            __Bx_xy[:,-1] = __Bx_xy[:,-2]

            __By_xy[0] = __By_xy[1]
            __By_xy[:,0] = __By_xy[:,1]
            __By_xy[-1] = __By_xy[-2]
            __By_xy[:,-1] = __By_xy[:,-2]

            __Axy[0] = __Axy[1]
            __Axy[:,0] = __Axy[:,1]
            __Axy[-1] = __Axy[-2]
            __Axy[:,-1] = __Axy[:,-2]

            __Bxy = (__Bx_xy**2+__By_xy**2)**(1/2)

            if simmetry == 'axi':
                __Axy = __Axy*1000/(2*pi*__x)
            else:
                __Bx_xy = (-1)*__Bx_xy
                __By_xy = (-1)*__By_xy

            self.J_xy_FEMM = __Jxy*1.0e6
            self.A_xy_FEMM = __Axy
            self.Bx_xy_FEMM = __Bx_xy
            self.By_xy_FEMM = __By_xy
            self.B_xy_FEMM = __Bxy

        if close_FEMM:
            femm.closefemm()

    #3.2.6 Método de solução do problema pelo método de Roth run_Roth
    def run_Roth(self, input_I, p, m_max = 500, n_max = 500):
        self.m_max = m_max
        self.n_max = n_max

        __J_rad = self.J*self.x0*self.y0/(pi**2)

        __theta_x1 = self.x1*pi/self.x0
        __theta_x2 = self.x2*pi/self.x0
        __theta_y1 = self.y1*pi/self.y0
        __theta_y2 = self.y2*pi/self.y0

        __theta2_x1 = __theta_x1.reshape((1,self.number_of_windings))
        __theta2_x2 = __theta_x2.reshape((1,self.number_of_windings))
        __theta2_y1 = __theta_y1.reshape((1,self.number_of_windings))
        __theta2_y2 = __theta_y2.reshape((1,self.number_of_windings))

        __m = np.linspace(1,m_max,m_max).reshape((m_max,1))*np.ones([m_max, self.number_of_windings])
        __n = np.linspace(1,n_max,n_max).reshape((n_max,1))*np.ones([n_max, self.number_of_windings])
```

```
        __area_ratio_winding_window = (__theta_x2-__theta_x1)*(__theta_y2-__theta_y1)/(pi**2)

        __Jmn = np.zeros([m_max+1,n_max+1])

        __Jmn[0,0] = np.sum(__J_rad*__area_ratio_winding_window)

        __Jmn[0,1:] = np.sum(2*__J_rad*(__theta2_x2-__theta2_x1)*\
                             (sin(__n*__theta2_y2)-sin(__n*__theta2_y1))/(__n*pi**2), 1)

        __Jmn[1:,0] = np.sum(2*__J_rad*(__theta2_y2-__theta2_y1)*\
                             (sin(__m*__theta2_x2)-sin(__m*__theta2_x1))/(__m*pi**2),1)

        __aux1 = (((sin(__m*__theta2_x2)-sin(__m*__theta2_x1))/__m).reshape(m_max,1,self.number_of_windings))
        __aux2 = (((sin(__n*__theta2_y2)-sin(__n*__theta2_y1))/__n).reshape(1,n_max,self.number_of_windings))

        for i in range (self.number_of_windings):
            __Jmn[1:,1:] += (4*__J_rad[i]/(pi**2))*(__aux1[:,:,i])@(__aux2[:,:,i])

        __m3 = np.linspace(0,m_max,m_max+1)
        __n3 = np.linspace(0,n_max,n_max+1)
        __h = np.ones([m_max+1,n_max+1])
        __h[:,1:] = 2*__h[:,1:]
        __h[1:,:] = 2*__h[1:,:]
        __my_nx_sq = ((__m3.reshape(m_max+1,1)@np.ones([1,n_max+1])*self.y0)**2+\
                      (np.ones([m_max+1,1])@__n3.reshape(1,n_max+1)*self.x0)**2)
        __my_nx_sq[0,0] = 1
        __Amn = __Jmn*(u0)*self.x0*self.y0/__my_nx_sq
        __Amn[0,0] = 0
        __Amn_Jmn = __Jmn*__Amn
        __sum_Amn_Jmn = np.sum(__Amn_Jmn/__h)

        self.Jmn = __Jmn*(pi**2)/(self.x0*self.y0)
        self.Amn = __Amn
        self.L_Roth = pi**2*p*__sum_Amn_Jmn/(input_I**2)

    #3.2.6.1 Método de computação de J(x, y) pelo método de Roth.
    def calc_J_xy_Roth (self, steps_y = 250):
        __steps_y = steps_y
        __steps_x = int(__steps_y*self.x0/self.y0)
        __m_max = self.m_max
        __n_max = self.n_max
        __m = np.linspace(0,__m_max,__m_max+1)
        __n = np.linspace(0,__n_max,__n_max+1)
        __Jmn = self.Jmn

        __x = np.ones([__steps_y,__steps_x])*(np.linspace(0,self.x0,__steps_x).reshape(1,__steps_x))
        __y = np.ones([__steps_y,__steps_x])*(np.linspace(0,self.y0,__steps_y).reshape(__steps_y,1))

        __x2 = __x.reshape(__steps_y,__steps_x,1,1)*np.ones([__steps_y,__steps_x,__m_max+1,1])\
                        *np.ones([__steps_y,__steps_x,1,__n_max+1])
        __y2 = __y.reshape(__steps_y,__steps_x,1,1)*np.ones([__steps_y,__steps_x,__m_max+1,1])\
                        *np.ones([__steps_y,__steps_x,1,__n_max+1])

        __mthetax = __x2*__m.reshape(1,1,__m_max+1,1)*pi/self.x0
        __nthetay = __y2*__n.reshape(1,1,1,__n_max+1)*pi/self.y0

        __cos_mx_cos_ny = cos(__mthetax)*cos(__nthetay)
        __J_xymn = (__Jmn.reshape(1,1,__m_max+1,__n_max+1)*__cos_mx_cos_ny)
        __J_xym = np.sum(__J_xymn,3)
        __J_xy = np.sum(__J_xym,2)
```

```
        self.J_xy_Roth = __J_xy

    #3.2.6.1 Método de computação de A(x, y) pelo método de Roth.
    def calc_A_xy_Roth (self, steps_y = 250):
        __steps_y = steps_y
        __steps_x = int(__steps_y*self.x0/self.y0)
        __m_max = self.m_max
        __n_max = self.n_max
        __m = np.linspace(0,__m_max,__m_max+1)
        __n = np.linspace(0,__n_max,__n_max+1)
        __Amn = self.Amn

        __x = np.ones([__steps_y,__steps_x])*(np.linspace(0,self.x0,__steps_x).reshape(1,__steps_x))
        __y = np.ones([__steps_y,__steps_x])*(np.linspace(0,self.y0,__steps_y).reshape(__steps_y,1))

        __x2 = __x.reshape(__steps_y,__steps_x,1,1)*np.ones([__steps_y,__steps_x,__m_max+1,1])\
                        *np.ones([__steps_y,__steps_x,1,__n_max+1])
        __y2 = __y.reshape(__steps_y,__steps_x,1,1)*np.ones([__steps_y,__steps_x,__m_max+1,1])\
                        *np.ones([__steps_y,__steps_x,1,__n_max+1])

        __mthetax = __x2*__m.reshape(1,1,__m_max+1,1)*pi/self.x0
        __mthetay = __y2*__n.reshape(1,1,1,__n_max+1)*pi/self.y0

        __cos_mx_cos_ny = cos(__mthetax)*cos(__mthetay)
        __A_xymn = (__Amn.reshape(1,1,__m_max+1,__n_max+1)*__cos_mx_cos_ny)
        __A_xym = np.sum(__A_xymn,3)
        __A_xy = np.sum(__A_xym,2)
        self.A_xy_Roth = __A_xy

    #3.2.6.1 Método de computação de JB(x, y) pelo método de Roth.
    def calc_B_xy_Roth (self, steps_y = 250):
        __steps_y = steps_y
        __steps_x = int(__steps_y*self.x0/self.y0)
        __m_max = self.m_max
        __n_max = self.n_max
        __m = np.linspace(0,__m_max,__m_max+1)
        __n = np.linspace(0,__n_max,__n_max+1)
        __Amn = self.Amn

        __x = np.ones([__steps_y,__steps_x])*(np.linspace(0,pi,__steps_x).reshape(1,__steps_x))
        __y = np.ones([__steps_y,__steps_x])*(np.linspace(0,pi,__steps_y).reshape(__steps_y,1))

        __x2 = __x.reshape(__steps_y,__steps_x,1,1)*np.ones([__steps_y,__steps_x,__m_max+1,1])\
                        *np.ones([__steps_y,__steps_x,1,__n_max+1])
        __y2 = __y.reshape(__steps_y,__steps_x,1,1)*np.ones([__steps_y,__steps_x,__m_max+1,1])\
                        *np.ones([__steps_y,__steps_x,1,__n_max+1])

        __mx = __x2*__m.reshape(1,1,__m_max+1,1)
        __ny = __y2*__n.reshape(1,1,1,__n_max+1)

        __n_cos_mx_sin_ny = cos(__mx)*sin(__ny)*(__n.reshape(1,1,1,__n_max+1))
        __Bx_xymn = (__Amn.reshape(1,1,__m_max+1,__n_max+1)*__n_cos_mx_sin_ny)
        __Bx_xym = np.sum(__Bx_xymn,3)
        __Bx_xy = np.sum(__Bx_xym,2)*pi/self.y0

        __m_sin_mx_cos_ny = sin(__mx)*cos(__ny)*(__m.reshape(1,1,__m_max+1,1))
        __By_xymn = (-1)*(__Amn.reshape(1,1,__m_max+1,__n_max+1)*__m_sin_mx_cos_ny)
        __By_xym = np.sum(__By_xymn,3)
        __By_xy = np.sum(__By_xym,2)*pi/self.x0
```

```
        self.Bx_xy_Roth = __Bx_xy
        self.By_xy_Roth = __By_xy

        self.B_xy_Roth = (__Bx_xy**2+__By_xy**2)**(1/2)

    def J_winding_fourier(self, n):
        self.Jn_winding = np.zeros([self.number_of_windings, n+1])
        __n = np.linspace(1, n, n)
        self.Jn_winding[:,0] = self.J*(self.y2-self.y1)/self.y0
        for i in range(self.number_of_windings):
            self.Jn_winding[i,1:] = (self.J[i]*2/(__n*pi))*\
            (sin(__n*pi*self.y2[i]/self.y0)-sin(__n*pi*self.y1[i]/self.y0))

    def generate_regions(self, n):
        __all_x = np.concatenate((np.array([0]), self.x1, self.x2, np.array([self.x0])),
                                 axis = None)
        self.region_x = np.unique(__all_x)
        self.number_of_regions = self.region_x.shape[0]-1
        __x1 = self.x1.reshape([self.number_of_windings,1])
        __x2 = self.x2.reshape([self.number_of_windings,1])

        __region_x_matrix = np.ones([self.number_of_windings, self.number_of_regions+1])*\
                                    (self.region_x.reshape([1,self.number_of_regions+1]))
        #winding i belongs to region j? True or False
        self.winding_belongs_to_region = ((__x1<=__region_x_matrix)*(__x2>__region_x_matrix))[:,:-1]

        self.n = n
        self.J_winding_fourier(self.n)
        #Value of Jk coefficient from winding i in region j
        __winding_belongs_to_region = self.winding_belongs_to_region.reshape([self.number_of_windings,
                                                                               self.number_of_regions, 1])*\
                                      np.ones([self.number_of_windings, self.number_of_regions, self.n+1])
        __Jn_winding = self.Jn_winding.reshape([self.number_of_windings, 1, self.n+1])*\
                                      np.ones([self.number_of_windings, self.number_of_regions, self.n+1])
        self.Jn_regions = np.sum(__winding_belongs_to_region*__Jn_winding, axis = 0)
        __current_regions_bool = (np.sum(self.Jn_regions**2, 1))>0
        self.current_regions_indexes = np.linspace(0,self.number_of_regions-1,self.number_of_regions)\
                                       [__current_regions_bool].astype(int)

    def Cn(self, region_index):
        return integral_t_K1(self.regions_mr[region_index],self.regions_mr[region_index+1])

    def Dn(self, region_index):
        return (I0(self.mr_core)/K0(self.mr_core))*self.Cn(region_index)

    def En(self, region_index):
        return integral_t_K1(0,self.regions_mr[region_index+1])

    def Fn(self, region_index):
        return self.Dn(region_index) - integral_t_I1(0,self.regions_mr[region_index])

    def Gn(self, region_index):
        return self.Dn(region_index) + integral_t_I1(self.regions_mr[region_index],
                                                     self.regions_mr[region_index+1])

    def rabins_region(self, region_index):
        __regions_r = self.region_x + self.r_core
        __n = np.linspace(1, self.n, self.n)
        __m = __n*pi/self.y0
        __regions_mr = self.regions_mr
```

```
        __P = 0
        __Q = 0
        __R = 0
        __Sn = np.zeros(self.n)
        __Tn = np.zeros(self.n)
        __Un = np.zeros(self.n)

        #Influence of the currents that are more distant from the core than the one in question
        for __j in self.current_regions_indexes[region_index<self.current_regions_indexes]:
            __Q += u0*self.Jn_regions[__j,0]*(__regions_r[__j+1]-__regions_r[__j])/2
            __Sn += (u0*self.Jn_regions[__j,1:]/__m**2)*self.Cn(__j)
            __Tn += (u0*self.Jn_regions[__j,1:]/__m**2)*self.Dn(__j)

        #Influence of the current in the studied region
        __P += u0*self.Jn_regions[region_index,0]*(-1/3)
        __Q += u0*self.Jn_regions[region_index,0]*(__regions_r[region_index+1])/2
        __R += u0*self.Jn_regions[region_index,0]*(-__regions_r[region_index]**3)/6
        __Sn += (u0*self.Jn_regions[region_index,1:]/__m**2)*self.En(region_index)
        __Tn += (u0*self.Jn_regions[region_index,1:]/__m**2)*self.Fn(region_index)
        __Un = -(u0*self.Jn_regions[region_index,1:]*pi/(2*__m**2))

        #Influence of the currents that are closer to the core than the one in question
        for __j in self.current_regions_indexes[region_index>self.current_regions_indexes]:
            __R += u0*self.Jn_regions[__j,0]*(__regions_r[__j+1]**3-__regions_r[__j]**3)/6
            __Tn += (u0*self.Jn_regions[__j,1:]/__m**2)*self.Gn(__j)

        return __P, __Q, __R, __Sn, __Tn, __Un

    #3.2.7 Método de solução do problema pelo método de Rabins run_Rabins.
    def run_Rabins (self, input_I, n):
        self.generate_regions(n)
        __regions_r = self.region_x + self.r_core
        __n = np.linspace(1, self.n, self.n)
        __m = __n*pi/self.y0
        __regions_mr = (__regions_r.reshape([self.number_of_regions+1,1]))@\
                       (__m.reshape([1,self.n]))
        self.regions_mr = __regions_mr
        self.mr_core = __regions_mr[0]

        __2W = 0
        for __i in self.current_regions_indexes:
            __Pi, __Qi, __Ri, __Sni, __Tni, __Uni = self.rabins_region(__i)
            __2W0 = 2*pi*self.Jn_regions[__i,0]*self.y0*\
                    ((__Pi/4)*(__regions_r[__i+1]**4-__regions_r[__i]**4)+
                     (__Qi/3)*(__regions_r[__i+1]**3-__regions_r[__i]**3)+
                     (__Ri)*(__regions_r[__i+1]-__regions_r[__i]))
            __2Wn = (2*pi*self.Jn_regions[__i,1:]/__m**2)*(self.y0/2)*\
                    (__Sni*integral_t_I1(__regions_mr[__i],__regions_mr[__i+1])+
                     __Tni*integral_t_K1(__regions_mr[__i],__regions_mr[__i+1])+
                     __Uni*integral_t_L1(__regions_mr[__i],__regions_mr[__i+1]))
            __2W += __2W0 + np.sum(__2Wn)
        self.L_Rabins = __2W/input_I**2

    #3.2.7.1 Método de computação de J(x, y) pelo método de Rabins.
    def calc_J_xy_Rabins (self, steps = 250):
        __steps_y = steps
        __steps_x = int(__steps_y*self.x0/self.y0)

        __Jxy_regions = np.zeros([__steps_y,self.number_of_regions])
```

```
        __x = np.linspace(0,self.x0,__steps_x)
        __y = np.linspace(0, self.y0, __steps_y).reshape([__steps_y,1])*np.ones([__steps_y, self.n])
        __n = np.linspace(1, self.n, self.n)

        for j in range (self.number_of_regions):
            __Jxy_regions[:,j] = self.Jn_regions[j,0] + np.sum(self.Jn_regions[j,1:]*\
                                                    cos(__n*pi*__y/self.y0), axis=1)

        __Jxy2_regions = np.zeros([__steps_y,__steps_x])

        for i in range (self.number_of_regions):
            __boolean_i = ((__x>=self.region_x[i])*(__x<=self.region_x[i+1])).astype(bool)
            __Jxy2_regions[:,__boolean_i] = __Jxy_regions[:,i].reshape([__steps_y,1])

        self.J_xy_Rabins = __Jxy2_regions

    #3.2.7.1 Método de computação de (x, y) pelo método de Rabins.
    def calc_A_xy_Rabins (self, steps = 250):
        __steps_z = steps
        __steps_r = int(__steps_z*self.x0/self.y0)
        __n = np.linspace(1, self.n, self.n)
        __m = __n*pi/self.y0

        __regions_r = self.region_x + self.r_core
        __regions_mr = (__m.reshape([1,self.n])*np.ones([self.number_of_regions+1,self.n]))*\
                       (__regions_r.reshape([self.number_of_regions+1,1]))
        self.regions_mr = __regions_mr

        __r = self.r_core + np.ones([__steps_z,__steps_r])*\
                            (np.linspace(0,self.x0,__steps_r).reshape(1,__steps_r))
        __mr = __r.reshape(__steps_z,__steps_r,1)*np.ones([__steps_z,__steps_r,self.n])*\
              __m.reshape(1,1,self.n)
        __z = np.ones([__steps_z,__steps_r])*(np.linspace(0,self.y0,__steps_z).reshape(__steps_z,1))
        __mz = __z.reshape(__steps_z,__steps_r,1)*np.ones([__steps_z,__steps_r,self.n])*\
               (__m.reshape(1,1,self.n))

        __r_sq = __r**2
        __r_inv = 1/__r
        __I1_mr = I1(__mr)
        __K1_mr = K1(__mr)
        __L1_mr = L1(__mr)
        __cos_mz = cos(__mz)

        __Arz = np.zeros([__steps_z,__steps_r])

        for __i in range (self.number_of_regions):
            __P, __Q, __R, __Sn, __Tn, __Un = self.rabins_region(__i)
            __boolean = (__r>=(self.region_x[__i]+self.r_core))*(__r<=(self.region_x[__i+1]+self.r_core))
            __Arz[__boolean] = (__P*__r_sq + \
                                __Q*__r + \
                                __R*__r_inv + \
                                np.sum((__Sn.reshape(1,1,self.n)*__I1_mr+\
                                        __Tn.reshape(1,1,self.n)*__K1_mr+\
                                        __Un.reshape(1,1,self.n)*__L1_mr)*\
                                        __cos_mz, axis = 2))[__boolean]
        self.A_xy_Rabins = __Arz

    #3.2.7.1 Método de computação de B(x, y) pelo método de Rabins.
    def calc_B_xy_Rabins (self, steps = 250):
        __steps_z = steps
```

```
        __steps_r = int(__steps_z*self.x0/self.y0)
        __n = np.linspace(1, self.n, self.n)
        __m = __n*pi/self.y0

        __regions_r = self.region_x + self.r_core
        __regions_mr = (__m.reshape([1,self.n])*np.ones([self.number_of_regions+1,self.n]))*\
                       (__regions_r.reshape([self.number_of_regions+1,1]))
        self.regions_mr = __regions_mr

        __r = self.r_core + np.ones([__steps_z,__steps_r])*\
                            (np.linspace(0,self.x0,__steps_r).reshape(1,__steps_r))
        __mr = __r.reshape(__steps_z,__steps_r,1)*np.ones([__steps_z,__steps_r,self.n])*\
              __m.reshape(1,1,self.n)
        __z = np.ones([__steps_z,__steps_r])*\
                    (np.linspace(0,self.y0,__steps_z).reshape(__steps_z,1))
        __mz = __z.reshape(__steps_z,__steps_r,1)*np.ones([__steps_z,__steps_r,self.n])*\
               (__m.reshape(1,1,self.n))

        __ones = np.ones([__steps_z,__steps_r])
        __r_inv = 1/__r
        __I0_mr = I0(__mr)
        __I1_mr = I1(__mr)
        __I2_mr = I2(__mr)
        __K0_mr = K0(__mr)
        __K1_mr = K1(__mr)
        __K2_mr = K2(__mr)
        __L0_mr = L0(__mr)
        __L1_mr = L1(__mr)
        __L2_mr = L2(__mr)
        __cos_mz = cos(__mz)
        __sin_mz = sin(__mz)

        __Br = np.zeros([__steps_z,__steps_r])
        __Bz = np.zeros([__steps_z,__steps_r])

        for __i in range (self.number_of_regions):
            __P, __Q, __R, __Sn, __Tn, __Un = self.rabins_region(__i)
            __boolean = (__r>=(self.region_x[__i]+self.r_core))*(__r<=(self.region_x[__i+1]+self.r_core))
            __Br[__boolean] = (np.sum(__m.reshape(1,1,self.n)*(\
                                      __Sn.reshape(1,1,self.n)*__I1_mr+\
                                      __Tn.reshape(1,1,self.n)*__K1_mr+\
                                      __Un.reshape(1,1,self.n)*__L1_mr)*\
                                      __sin_mz, axis = 2))[__boolean]
            __Bz[__boolean] = (3*__P*__r + \
                               2*__Q*__ones + \
                               np.sum((__Sn.reshape(1,1,self.n)*(__I1_mr+(__mr/2)*(__I0_mr+__I2_mr))+\
                                       __Tn.reshape(1,1,self.n)*(__K1_mr-(__mr/2)*(__K0_mr+__K2_mr))+\
                                       __Un.reshape(1,1,self.n)*\
                                           (__L1_mr+(__mr/2)*(__L0_mr+__L2_mr+(2*__mr/(3*pi)))))*\
                               __cos_mz, axis = 2)*__r_inv)[__boolean]

        self.Bx_xy_Rabins = __Br
        self.By_xy_Rabins = __Bz
        self.B_xy_Rabins = (__Br**2+__Bz**2)**(1/2)

    #3.2.8 M todos para plotar os mapas dos campos.
    def plot_heatmap_generic(self, field_xy, label = '', save_name = False):
        __f = plt.figure()
        __f.set_figwidth(self.x0*10/self.y0)
        __f.set_figheight(10)
```

```
        plt.imshow(field_xy, cmap = 'jet', interpolation = 'nearest', origin='lower')
        plt.colorbar(label = label)

        plt.xticks([0,field_xy.shape[1]],[0, self.x0])
        plt.yticks([0,field_xy.shape[0]],[0, self.y0])

        if not(save_name == False):
            plt.savefig(fname = save_name+'.png', dpi=600, bbox_inches='tight')

        plt.show()

    def plot_heatmap_FEMM_J(self, save_name = False):
        __field_xy = self.J_xy_FEMM
        __label = '$A/m^2$'
        self.plot_heatmap_generic(field_xy = __field_xy,
                                  label = __label,
                                  save_name = save_name)

    def plot_heatmap_FEMM_A(self, save_name = False):
        __field_xy = self.A_xy_FEMM
        __label = '$V\,s/m$'
        self.plot_heatmap_generic(field_xy = __field_xy,
                                  label = __label,
                                  save_name = save_name)

    def plot_heatmap_FEMM_B(self, save_name = False):
        __field_xy = self.B_xy_FEMM
        __label = '$T$'
        self.plot_heatmap_generic(field_xy = __field_xy,
                                  label = __label,
                                  save_name = save_name)

    def plot_heatmap_Roth_J(self, save_name = False):
        self.calc_J_xy_Roth()
        __field_xy = self.J_xy_Roth
        __label = '$A/m^2$'
        self.plot_heatmap_generic(field_xy = __field_xy,
                                  label = __label,
                                  save_name = save_name)

    def plot_heatmap_Roth_A(self, save_name = False):
        self.calc_A_xy_Roth()
        __field_xy = self.A_xy_Roth
        __label = '$V\,s/m$'
        self.plot_heatmap_generic(field_xy = __field_xy,
                                  label = __label,
                                  save_name = save_name)

    def plot_heatmap_Roth_B(self, save_name = False):
        self.calc_B_xy_Roth()
        __field_xy = self.B_xy_Roth
        __label = '$T$'
        self.plot_heatmap_generic(field_xy = __field_xy,
                                  label = __label,
                                  save_name = save_name)

    def plot_heatmap_Rabins_J(self, save_name = False):
        self.calc_J_xy_Rabins()
        __field_xy = self.J_xy_Rabins
```

```
        __label = '$A/m^2$'
        self.plot_heatmap_generic(field_xy = __field_xy,
                                  label = __label,
                                  save_name = save_name)

    def plot_heatmap_Rabins_A(self, save_name = False):
        self.calc_A_xy_Rabins()
        __field_xy = self.A_xy_Rabins
        __label = '$V\,s/m$'
        self.plot_heatmap_generic(field_xy = __field_xy,
                                  label = __label,
                                  save_name = save_name)

    def plot_heatmap_Rabins_B(self, save_name = False):
        self.calc_B_xy_Rabins()
        __field_xy = self.B_xy_Rabins
        __label = '$T$'
        self.plot_heatmap_generic(field_xy = __field_xy,
                                  label = __label,
                                  save_name = save_name)
```