Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia Aeroespacial

Desenvolvimento teórico de família de antenas dipolo e Yagi-Uda e ferramentas computacionais para projeto de arranjos

Autor: Vítor Lima Aguirra
Orientador: Prof. Dr. Sébastien Rondineau

Brasília, DF
2023

Vítor Lima Aguirra

Desenvolvimento teórico de família de antenas dipolo e Yagi-Uda e ferramentas computacionais para projeto de arranjos

Monografia submetida ao curso de graduação em (Engenharia Aeroespacial) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Aeroespacial).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Sébastien Rondineau

Brasília, DF

2023

Vítor Lima Aguirra

Desenvolvimento teórico de família de antenas dipolo e Yagi-Uda e ferramentas computacionais para projeto de arranjos/ Vítor Lima Aguirra. – Brasília, DF, 2023-

401 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Sébastien Rondineau

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2023.

1. Aguirra. 2. Engenharia Aeroespacial. I. Prof. Dr. Sébastien Rondineau. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Desenvolvimento teórico de família de antenas dipolo e Yagi-Uda e ferramentas computacionais para projeto de arranjos

CDU 02:141:005.6

Vítor Lima Aguirra

Desenvolvimento teórico de família de antenas dipolo e Yagi-Uda e ferramentas computacionais para projeto de arranjos

Monografia submetida ao curso de graduação em (Engenharia Aeroespacial) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Aeroespacial).

Trabalho aprovado. Brasília, DF, 14 de Dezembro de 2023:

Prof. Dr. Sébastien Rondineau
Orientador

Prof. Dr. Daniel Costa Araújo
Convidado 1

Prof. Dr. William Reis
Convidado 2

Brasília, DF
2023

Este trabalho é dedicado à ciência, que trouxe à humanidade evolução e desenvolvimento, e que dá sentido à curta vida que temos.

Agradecimentos

O maior e mais sincero agradecimento aos meus pais, todas minhas conquistas eu tenho a vocês a quem agradecer, vocês são inspiração de pessoas honestas e respeitosas e me ensinaram a sempre dar o meu melhor. Gostaria de deixar um grande agradecimento à meu orientador, que me ensinou muito sobre engenharia no mundo real, teve toda a paciência para explicar as coisas mais simples, se tornou um grande professor em diversas questões da minha vida. Um grande agradecimento à minha namorada, Thayná, sua força me sustentou nos momentos mais difíceis, sua companhia iluminou minha vida. E outro grande agradecimento ao Ighor Dias, o professor que me ensinou sobre força de vontade, responsabilidade, atitude e perseverança.Aos meus irmãos, por me ensinarem todas as coisas que não deviam ser ensinadas a uma criança, amo muito vocês. À Angélica e ao Rodrigo, por suas amizades e companheirismo. À F. D. C. Willard, por suas contribuições vocais ao tema.

Resumo

Antenas Yagi-Uda são antenas de diretividade razoável, amplamente utilizadas em sistemas de telecomunicações. Por sua simplicidade, são antenas de baixo custo e fáceis de serem projetadas. O número de elementos de uma antena pode ser escolhido para atingir variados valores de diretividade. Arranjos de antenas apresentam a possibilidade de configurar antenas simples de forma a produzir um campo eletromagnético resultante complexo e otimizado para diferentes aplicações. Este trabalho é resultado do desenvolvimento de uma família com 4 variações de antenas Yagi-Uda, com diretividade entre 3 e 9 dB, e códigos em *Octave* e *Python* para calcular o campo elétrico resultante de arranjos arbitrários de antenas. Em seu desenvolvimento, foram utilizadas tabelas disponíveis na literatura em conjunto com *Ansys Electronics HFSS*, para desenhar e otimizar as antenas para atender diferentes requisitos, além de relações bem conhecidas de fator de arranjo de antenas disponíveis na literatura. O resultado final são antenas que podem ser utilizadas em uma estação de solo, criando um arranjo de antenas para comunicação com satélites em órbita terrestre baixa, variando a diretividade do sistema e otimizando a distribuição de radiação no formato desejado.

Palavras-chaves: Antena. Arranjos. Ansys Electronics. Telecomunicação espacial. Radiação eletromagnética.

Abstract

Yagi-Uda antennas have been widely utilized in space communication systems due to their simplicity, low cost and reasonable and optimizable directivity. The number of passive elements can be modified in order to achieve the desired directivity, based on tables available on the literature, and little optimizations are required to design an efficient antenna. Antenna arrays, in the other hand, provide an effective method for utilizing simple antennas in the construction of a complex resultant electric field, which can be modified and optimized as desired for a variety of applications. This work presents the results of the design of a family of four antennas, with directivity ranging from 3 to 9 dB, and a collection of softwares in *Octave* and *Python* that calculates the resultant electric field for an arbitrary antenna array. The individual antennas were designed and optimized on Ansys Electronics HFSS in order to fulfill the project requirements for communication with LEOs satellites. Different combinations of said antennas can be utilized to create a ground station, varying their position and orientation in order to modify the resulting radiation pattern as desired.

Key-words: Yagi-Uda Antennas. LEO. Ground Station. Telecommunication Systems. Spacial Telecommunication.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

LEO	*Low Earth Orbit*
LOFAR	*Low Frequency Array*
MeerKAT	Arranjo de 64 antenas, originalmente chamado *Karoo Antenna Array*
SKAO	SKA Observatory, organização de radio-astronomia com telescópios na Austrália e África do Sul
Y1EL	Antena Dipolo
Y2EL	Antena Yagi-Uda de 2 elementos
Y3EL	Antena Yagi-Uda de 3 elementos
Y4EL	Antena Yagi-Uda de 4 elementos
1Y4EL	Arranjo contendo apenas uma antena Yagi-Uda de 4 elementos
2Y4EL	Arranjo com 2 antenas Yagi-Uda de 4 elementos
3Y4EL	Arranjo com 3 antenas Yagi-Uda de 4 elementos
4Y4EL	Arranjo com 4 antenas Yagi-Uda de 4 elementos
5Y4EL	Arranjo com 5 antenas Yagi-Uda de 4 elementos
BFGS	Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm
L-BFGS-B	Limited-memory BFGS subject to simple bounds on the variables
RHCP	Polarização circular da regra da mão direita *Right-Hand Circular Polarization*
LHCP	Polarização circular da regra da mão esquerda *Left-Hand Circular Polarization*

Lista de símbolos

μ_0	Permeabilidade magnética intrinsica no espaço livre
ϵ_0	Permissividade elétrica intrinsica no espaço livre
c_0	Velocidade da luz no espaço livre
f	Frequência
ω	Frequência angular ou pulsação
λ_0	Comprimento de onda no espaço livre
k_0	Número de onda
$\vec{k}$	Vetor de onda
δ	Defasagem geométrica
F_{ai}	Fator de arranjo de uma antena i
ϕ	Ângulo azimutal no sistema de coordenadas esféricas
θ	Ângulo polar theta no sistema de coordenadas esféricas
α	Ângulo de elevação da orientação da antena
β	Ângulo de azimute da orientação da antena
γ	Ângulo de rolamento da orientação da antena
$\vec{P}$	Vetor de observação do campo eletromagnético
$\vec{p}$	Vetor de translação
I	Corrente elétrica
j	Número imaginário
η_0	Impedância intrínsica do espaço livre
$\vec{\mathscr{E}}$	Vetor de campo elétrico
$\vec{E}$	Vetor de campo elétrico em regime harmônico
$\vec{\mathscr{D}}$	Vetor de densidade de fluxo elétrico

$\vec{D}$	Vetor de densidade de fluxo elétrico em regime harmônico
$\vec{\mathscr{H}}$	Vetor de campo magnético
$\vec{H}$	Vetor de campo magnético em regime harmônico
$\vec{\mathscr{B}}$	Vetor de densidade de fluxo magnético
$\vec{B}$	Vetor de densidade de fluxo magnético em regime harmônico
ρ_τ	Densidade de cargas elétricas
J_τ	Densidade de corrente elétrica
J	Corrente de excitação
∇	Operador Nabla
$\vec{S}$	Vetor de Poyting
A_θ	Abertura de meia potência em um plano em que θ varia
A_ϕ	Abertura de meia potência em um plano em que ϕ varia

Sumário

Introdução

Sistemas de Telecomunicações

Sistemas de comunicação desempenham papel fundamental no funcionamento de diversas tecnologias. O envio de informações utilizando ondas eletromagnéticas está presente no dia a dia desde formas óbvias como na utilização da *Internet* e sistemas de GPS, como em maneiras mais sutis em dispositivos sem fio como *mouses*, teclados, controles remotos e equipamentos de micro-ondas, até mesmo em transmissão de energia. (GODARA, 1997; RE et al., 2019)

A utilização de satélites em órbita baixa se faz interessante por diminuir consideravelmente o custo de lançamento do satélite. Tal diminuição de custos atualmente compensa a perda de vida útil devido ao pequeno arrasto terrestre presente nestas órbitas e os possíveis custos adicionais com sistemas de controle de órbita e atitude necessários para corrigir os efeitos deste arrasto. O arrasto também se apresenta como uma solução conveniente para de-órbita dos satélites após o término de sua missão, liberando a órbita para futuras missões e contribuindo para uma menor quantidade de lixo espacial, o qual é tema recorrente na mídia e assunto de importante consideração no design de missões atualmente. (GANZ; GONG; LI, 1994)

Para a utilização dos satélites, é necessário o uso de sistemas de telecomunicação que transmitem dados entre satélite e solo. Tais sistemas utilizam-se da radiação eletromagnética como meio de propagação da informação. Assim, naturalmente, se faz necessário o desenvolvimento de antenas que compõem parte dos sistemas de telecomunicação necessários, e cada aplicação pode exigir qualidades e parâmetros específicos das antenas a serem utilizadas. (MARAL et al., 1991)

Portanto, existem diversos modelos de antenas, cada qual com nichos de aplicações diferentes. Cada antena apresenta propriedades inerentes ao seu design, como parâmetros S, diretividade e impedância, que definem sua aplicação. Sendo assim, para cada aplicação, certas propriedades são mais necessárias e valorizadas que outras. As antenas variam de modelos que objetivam uma ampla gama de aplicações com certa eficiência até antenas com nichos mais específicos, assim obtendo maior desempenho. (CHEN, 1975)

Arranjos de Antenas

A comunicação de uma antena na superfície terrestre com um satélite em *LEO* é dificultada pelas grandes distâncias e obstáculos entre os sistemas, assim, o desenvolvi-

mento de uma antena capaz de realizar este tipo de comunicação se torna muito cara e imprática. Uma alternativa mais interessante é a utilização de diversas antenas simples e baratas, trabalhando em conjunto para atingir as propriedades de uma antena mais sofisticada (RONDINEAU et al., 2006). Este arranjo de antenas dispõe de diversas vantagens em relação ao desenvolvimento de uma só antena com geometria complicada, e a interferometria entre as antenas do arranjo é utilizada de forma inteligente para atingir desempenhos muito maiores do que o que seria possível com apenas uma antena e com custo muito menos elevado. Muitas vezes o custo do arranjo como um todo, contabilizando várias antenas, é consideravelmente menor que o custo para o desenvolvimento de uma única antena capaz de suprir as mesmas propriedades do arranjo. Possivelmente, os requisitos da missão podem ser tão específicos ao ponto que tornam-se inviáveis de serem atendidos com uma única antena. (ROSHI et al., 2021)

Um estudo de caso interessante é o do observatório Arecibo (2023). Construído em 1963, o observatório contava com uma antena de centenas de metros de diâmetro, a maior antena do mundo em sua época, com geometria aproximadamente esférica e com toda sua estrutura suspensa por cabos sobre o solo. O observatório, marco da engenharia e fundamental no desenvolvimento da astronomia durante sua operação, parou sua operação devido à falhas estruturais severas, após mais de cinco décadas de funcionamento. Começou então o planejamento da demolição controlada do telescópio, entretanto, em 2020, devido a uma falha catastrófica da estrutura o telescópio foi completamente destruído, por sorte não houve feridos no incidente. O custo de construção e operação do telescópio era alto, e a manutenção mecânica da estrutura de centenas de metros era complexa, cara e demorada, demandando o desligamento completo do sistema por dias ou semanas. Apesar das grandes contribuições do telescópio durante sua operação, não há planos para reconstrução do telescópio. As aplicações do antigo telescópio podem atualmente ser supridas por outros observatórios e telescópios em operação, muitos destes utilizando arranjos de antenas.

Atualmente, estão em operação dezenas de estruturas espalhadas pelo mundo com propósito de estudos astronômicos similares ao observatório Arecibo. Entretanto, tais sistemas não utilizam apenas uma antena gigante e complexa, e sim centenas ou milhares de antenas simples. Sistemas como o *ASTRON* (2023), *SARAO* (2022) e *SKAO* (2023) são formados por múltiplas antenas, espalhadas em múltiplas localizações, todas atuando em conjunto e trabalhando como uma só. Estes arranjos podem ter áreas efetivas quilométricas, com antenas a centenas de quilômetros de distância uma da outra, ou mesmo em lados opostos do planeta. De fato, muitas vezes um arranjo de antenas pode transformar o planeta inteiro em uma antena. As possibilidades de arranjos de antenas são ilimitadas. Um exemplo é o LOFAR (*Low Frequency Array*), um projeto inicialmente dos países baixos, que atualmente tem cooperação internacional contendo antenas em diversos países europeus.

No tópico de comunicação terrestre com satélites em *LEO*, os trabalhos de Rondineau et al. (2006) e Ingram et al. (2005) mostram que arranjos de antenas simples podem de fato executar o mesmo trabalho de antenas muito mais complexas, com custo muito menor e ainda com eficiência superior.

Os arranjos de antenas necessitam de algoritmos avançados para cálculo da interferometria dos dados das diversas antenas e reconstrução da informação em software. Um ótimo exemplo disto é o projeto do Telescópio de Horizonte de Eventos (2023), responsável pela divulgação da primeira imagem real de um buraco negro na história. De acordo com o *National Geographic* (2022), a imagem foi construída a partir de uma enorme quantidade de dados, coletados por telescópios localizados em lados opostos do planeta. O tratamento dos dados para reconstrução da imagem demandou meses de computação intensa. Este arranjo de antenas se comporta como uma antena de dimensão planetária, e a imagem do buraco negro, que é extrema importância para confirmação de teorias físicas relativísticas, não seria possível sem a utilização de um arranjo de antenas de dimensões gigantes.

A manutenção mecânica das antenas de um arranjo é muito mais simples, considerando que cada antena do arranjo não precisa ser uma antena muito sofisticada. Além do baixo custo, quando há falha mecânica em uma antena, esta pode ser desativada para manutenção sem a necessidade de paralisar a operação de todo o sistema e sem grandes efeitos no desempenho do arranjo, o que é muito interessante e prático.

Quando não é possível utilizar diversas antenas pelo mundo, é possível ainda utilizar uma antena com posição que varia no tempo para se comportar de maneira similar a um arranjo, formando assim uma antena ou radar de abertura sintética, como discutido por Moreira et al. (2013). Por exemplo radares montadas em aviões, capazes de sobrevoar regiões extensas. O radar continuamente escaneia em vôo a região ao redor, e utilizando um processamento dos dados do radar em posições e tempos diferentes, constituindo ao final uma imagem de boa resolução formada efetivamente por diversos radares, embora na prática há apenas um.

Justificativa

Como é discutido em Balanis (2005), as antenas do tipo Yagi-Uda são interessantes devido à sua simplicidade e alta diretividade. O desenvolvimento de uma antena Yagi tem como base uma antena Dipolo, que é uma antena simples de baixo custo. Elementos denominados parasitas ou diretores, são adicionados ao redor da antena dipolo de forma a modelar o campo elétrico resultante e a diretividade da antena conforme desejado.

Para certas aplicações, entretanto, é requerida uma coleção específica de propriedades que podem ser difíceis de serem obtidas com a utilização de uma única antena. É interessante então utilizar múltiplas antenas para cumprir os requisitos de uma missão.

Quando um conjunto de antenas é utilizado de forma que o conjunto atua como uma única antena, modelando suas propriedades de formas específicas, este conjunto é chamado de arranjo de antenas. (RONDINEAU et al., 2006)

Para auxiliar no projeto de arranjos de antenas, se faz interessante o desenvolvimento de ferramentas computacionais para simulação rápida dos campos elétricos resultantes de arranjos de antenas, onde seja possível implementar algoritmos de otimização numérica para otimizar um arranjo para uma aplicação específica.

Objetivos

Gerais

O objetivo final do projeto é o desenvolvimento de uma estação de base para comunicação com satélites terrestres em órbita baixa. A estação contará com um arranjo de antenas que será responsável por receber os dados enviados por quaisquer satélites em uma *LEO* fixa. Inicialmente, o arranjo terá um apontamento manual, o que significa que não contará com um sistema automático para movimentar as antenas para receber informações de uma órbita qualquer, mas as ferramentas computacionais desenvolvidas neste projeto podem ser utilizadas para calcular a melhor orientação e alimentação do arranjo para então este ser manualmente modificado. Não há impedimentos para implementar futuramente um sistema motorizado para apontamento das antenas, entretanto, não há necessidade para tal sistema no atual projeto. Tal sistema automático pode ser interessante para aplicações onde seja necessário um arranjo mais dinâmico com distribuições de radiação variantes no tempo. Para o projeto atual, é importante que a polarização das antenas seja circular para evitar perdas de informações na ionosfera.

A estação de base consistirá em um arranjo plano de antenas posicionado no solo. O arranjo será desenvolvido futuramente, utilizando os conceitos, projetos e códigos apresentados neste trabalho. As antenas utilizadas podem ser as desenvolvidas neste trabalho, ou outros modelos disponíveis comercialmente ou projetados conforme o necessário. Dado uma órbita *LEO* cujo satélite de interesse se encontra, a melhor distribuição de radiação para a comunicação do solo com esta órbita deve ser definida previamente.

Específicos

Serão desenvolvidas uma família de antenas com diretividades variadas, que serão utilizadas no arranjo a ser desenvolvido. Após projetadas a família de antenas, as simulações numéricas executadas no *ANSYS Electronics* devem fornecer a distribuição de radiação de cada antena, que é necessária para o cálculo da interferometria entre as antenas no arranjo. Para auxiliar no projeto do arranjo de antenas proposto, será desenvol-

vido um código de cálculo de campo elétrico resultante de arranjos de antenas arbitrários. O código utilizará os campos elétricos de cada antena individualmente exportadas do *ANSYS Electronics* para determinar o campo elétrico do arranjo. A posição e orientação de cada antena serão definidas por meio de um código de otimização a ser desenvolvido. Este código irá simular a distribuição de radiação do arranjo várias vezes, utilizando a distribuição de radiação de cada antena e as fórmulas de defasagem geométrica e rotação tridimensional, modificando a posição, orientação, e número de antenas no arranjo, até que a distribuição de radiação resultante do arranjo seja suficientemente parecida com a distribuição desejada para que a função de custo utilizada retorne um valor satisfatório.

Organização do trabalho

O trabalho atual é separado em duas partes: O desenvolvimento teórico de uma família de antenas Yagi-Uda para serem utilizadas no desenvolvimento de um arranjo; E o desenvolvimento de ferramentas computacionais para auxiliar no projeto do arranjo, sendo parametrizáveis a posição, orientação e corrente de alimentação de cada antena individualmente.

Na primeira parte, será tratado o desenvolvimento de uma antena dipolo como base das antenas Yagi-Uda. A geometria da antena será desenvolvida a partir da literatura, sendo a antena otimizada de forma a diminuir o parâmetro S. Depois, utilizando como elemento excitador a antena dipolo desenvolvida, as antenas Yagi-Uda serão projetadas adicionando elementos parasitas ao redor da antena dipolo, de forma a modificar o diagrama de radiação da antena aumentando a diretividade da mesma.

Na segunda parte do trabalho, algoritmos para calculo de fator de arranjo e campo elétrico resultante de arranjos de antenas serão desenvolvidos. As antenas desenvolvidas na primeira parte do trabalho serão utilizadas para validação dos algoritmos desenvolvidos. Uma interface gráfica será desenvolvida para facilitar o projeto de arranjos de antenas. Por fim, serão testados algoritmos de otimização numérica para projetar arranjos de antenas com objetivos específicos. Um diagrama de radiação alvo será modelado com funções matemáticas simples. Então, os algoritmos de otimização numérica serão utilizados para recriar o diagrama de radiação alvo utilizando as antenas as antenas projetadas na primeira parte do trabalho.

1 Fundamentação Teórica

Conceitos chave devem ser revisados para o desenvolvimento do projeto de antenas atual. Dentre estes estão o parâmetros S, diretividade e o ângulo de abertura a meia potência. Estas são propriedades de uma antena que definem sua performance e suas aplicações, e são fatores importantes neste projeto pois definem os requisitos que a antena projetada deve atender.

1.1 Vetores, Notação vetorial e Notação matricial

Para manter certo vigor matemático no decorrer deste trabalho e facilitar o entendimento dos assuntos abordados, serão apresentados alguns conceitos básicos sobre vetores, operações com vetores, e notação matricial de vetores. Estes conceitos são muito conhecidos e utilizados na literatura (HILBERT, 1950; HHEINBOCKEL, 1996), e são extremamente úteis na modelagem física e computacional das antenas de forma eficiente.

Um vetor $\vec{V}$ é definido de acordo com a Equação 1.1.

$$\vec{V} = V_x\hat{x} + V_y\hat{y} + V_z\hat{z} \tag{1.1}$$

Os símbolos $\hat{x}$, $\hat{y}$, e $\hat{z}$ são vetores unitários sobre eixos x, y, e z respectivamente, e formam a base canônica do um espaço vetorial com dimensão R^3. Um ponto (x,y,z) no espaço pode ser representado por um vetor $\vec{P}$, e o vetor $\vec{V}$ pode estar posicionado sobre este, e ser dado em função de $\vec{P}$, representando um campo vetorial.

$$\vec{P} = x\hat{x} + y\hat{y} + z\hat{z} \tag{1.2}$$
$$\vec{V}(\vec{P}) = V_x(\vec{P})\hat{x} + V_y(\vec{P})\hat{y} + V_z(\vec{P})\hat{z} \tag{1.3}$$
$$\vec{V}(\vec{P}) = \vec{V}(x,y,z) = V_x(x,y,z)\hat{x} + V_y(x,y,z)\hat{y} + V_z(x,y,z)\hat{z} \tag{1.4}$$

Um vetor pode ainda variar no tempo, sendo representado como

$$\vec{V} = \vec{V}(x,y,z,t) = V_x(x,y,z,t)\hat{x} + V_y(x,y,z,t)\hat{y} + V_z(x,y,z,t)\hat{z} \tag{1.5}$$

Um vetor $\vec{Q}$ pode ser dado pela soma de dois vetores $\vec{V}$ e $\vec{P}$, sendo

$$\vec{Q} = \vec{V} + \vec{P} = (V_x + P_x)\hat{x} + (V_y + P_y)\hat{y} + (V_z + P_z)\hat{z} \tag{1.6}$$
$$\vec{Q} = Q_x\hat{x} + Q_y\hat{y} + Q_z\hat{z} \tag{1.7}$$

sendo

$$Q_x = V_x + P_x \tag{1.8}$$
$$Q_y = V_y + P_y \tag{1.9}$$
$$Q_z = V_z + P_z \tag{1.10}$$

Um vetor tem módulo definido pela Equação 1.11, que representa o comprimento do vetor.

$$|\vec{V}| = V = \sqrt{V_x^2 + V_y^2 + V_z^2} \tag{1.11}$$

O produto interno, ou produto escalar, entre dois vetores $\vec{V}$ e $\vec{Q}$, escrito como $\vec{V} \cdot \vec{Q}$, é definido pela Equação 1.12, e tem como resultado um valor escalar.

$$\vec{V} \cdot \vec{Q} = V_x Q_x + V_y Q_y + V_z Q_z \tag{1.12}$$

A notação matricial de um vetor $\vec{V}$ é

$$\vec{V} = \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = V_x \hat{x} + V_y \hat{y} + V_z \hat{z} \tag{1.13}$$

O que significa dizer que a notação vetorial e matricial representam o mesmo vetor, são o mesmo objeto matemático representado de maneiras distintas. Hora uma notação pode ser útil, hora outra pode ser mais coveniente. Em geral, a notação vetorial é mais utilizada para os cálculos analíticos, enquanto a notação matricial é uma forma mais conveniente de visualizar o vetor e representá-lo em códigos, e operações com matrizes são mais eficientes computacionalmente, além de permitir computações em paralelo que agilizam significantemente as execuções de algoritmos de computador.

O produto vetorial entre dois vetores $\vec{V}$ e $\vec{Q}$, escrito como $\vec{V} \times \vec{Q}$, é definido como o determinante de uma matriz formada com os dois vetores, e tem como resultado um vetor perpendicular a ambos $\vec{V}$ e $\vec{Q}$, seguindo a regra da mão direita (Dextrógiro). Essa operação é ilustrada para vetores em um espaço R^3 nas Equações 1.14 e 1.15 nas formas matricial e vetorial, respectivamente.

$$\vec{V} \times \vec{Q} = \begin{bmatrix} 0 & -V_z & V_y \\ V_z & 0 & -V_x \\ V_y & -V_x & 0 \end{bmatrix} \begin{bmatrix} Q_x \\ Q_y \\ Q_z \end{bmatrix} \tag{1.14}$$

$$\vec{V} \times \vec{Q} = (V_y Q_z - V_z Q_y)\hat{x} + (V_z Q_x - V_x Q_z)\hat{y} + (V_x Q_y - V_y Q_x)\hat{z} \tag{1.15}$$

Os produtos interno e vetorial podem ainda ser expressos pelas Equações 1.16 e 1.17, onde w é o menor ângulo entre os dois vetores, e $\hat{u}$ é um vetor unitário perpendicular a ambos $\vec{V}$ e $\vec{Q}$, de acordo com a regra da mão direita.

$$\vec{V} \cdot \vec{Q} = VQ\cos(\omega) \tag{1.16}$$

$$\vec{V} \times \vec{Q} = VQ\sin(\omega)\hat{u} \tag{1.17}$$

Vetores podem ser rotacionados e transladados, o que é equivalente a uma mudança de sistema de coordenadas. Estas rotações são realizadas por matrizes de rotação, pois a notação matricial da operação de rotação é mais coveniente que a notação vetorial da rotação. Uma matriz de rotação $\overline{\overline{R}}_{\hat{u}}(\omega)$, onde $\hat{u}$ é um vetor unitário que representa o eixo de rotação da matriz e ω é o ângulo da rotação, respeita as propriedades representadas pelas Equações 1.18 e 1.19. A segunda propriedade mostra que o produto matricial de uma matrix de rotação com sua transposta é a matriz identidade (denotada neste trabalho pelo número 1), ou seja, a matriz inversa de uma matriz de rotação é igual à sua transposta. A primeira propriedade mostra que o determinante de uma matriz de rotação é igual a 1. Isto se reflete na conservação do comprimento do vetor após a rotação, ou seja, a operação de rotação não escala o vetor.

$$\det(\overline{\overline{R}}_{\hat{u}}(\omega)) = 1 \tag{1.18}$$

$$\overline{\overline{R}}_{\hat{u}}^{-1}(\omega) = \overline{\overline{R}}_{\hat{u}}^{t}(\omega) \tag{1.19}$$

onde

$$\overline{\overline{R}}_{\hat{u}}^{-1}(\omega)\overline{\overline{R}}_{\hat{u}}(\omega) = 1$$

As três matrizes de rotação mais usuais são definidas em 1.22, 1.21 e 1.20, onde a matriz de rotação $\overline{\overline{R}}_{\hat{z}}(\omega)$ rotaciona um vetor em torno do eixo $\hat{z}$, a matriz de rotação $\overline{\overline{R}}_{\hat{y}}(\omega)$ rotaciona um vetor em torno do eixo $\hat{y}$ e a matriz de rotação $\overline{\overline{R}}_{\hat{x}}(\omega)$ rotaciona um vetor em torno do eixo $\hat{x}$. A multiplicação de duas matrizes de rotação resulta em uma matriz de rotação, entretanto o resultado depende da ordem em que as matrizes aparecem, assim, a multiplicação de matrizes é uma operação não comutativa.

$$\overline{\overline{R}}_{z}(\omega) = \begin{bmatrix} \cos(\omega) & -\sin(\omega) & 0 \\ \sin(\omega) & \cos(\omega) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.20}$$

$$\overline{\overline{R}}_y(\omega) = \begin{bmatrix} \cos(\omega) & 0 & \sin(\omega) \\ 0 & 1 & 0 \\ -\sin(\omega) & 0 & \cos(\omega) \end{bmatrix} \tag{1.21}$$

$$\overline{\overline{R}}_x(\omega) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\omega) & -\sin(\omega) \\ 0 & \sin(\omega) & \cos(\omega) \end{bmatrix} \tag{1.22}$$

Matrizes de rotação são utilizadas para mudar o sistema de coordenadas de um vetor. Assim, se um sistema referencial S_i é obtido rotacionando um sistema referencial S_j pelos ângulos α e β em torno dos respectivos eixos $\hat{y}$ e $\hat{z}$, então uma matriz de rotação, $\overline{\overline{R}}_{S_j}^{S_i}(\beta, \alpha)$, é dada como uma matriz de mudança de sistema de coordenadas de um sistema S_i para um sistema S_j, por meio de uma rotação de ângulo β em torno do eixo $\hat{z}$ seguida de uma rotação de ângulo α em torno do eixo $\hat{y}$, conforme a Equação 1.23. O sinal negativo porque, a medida que o sistema referencial S_j rotaciona em relação ao sistema S_i, o sistema S_j "vê"um ponto fixo rotacionando no sentido contrário. Ou seja, se o sistema S_j rotaciona de acordo com uma rotação positiva $\overline{\overline{R}}(\omega)$, este vê um ponto fixo (no sistema S_i) qualquer rotacionar de acordo com uma rotação negativa $\overline{\overline{R}}(-\omega)$. É importante notar se as rotações são definidas sobre os eixos fixos do sistema inicial S_i ou sobre eixos que acompanham o sistema S_j, e a definição dada é para rotações sobre eixos fixos, caso contrário a ordem das matrizes seria inversa. Naturalmente, o subíndice j não deve ser confundido com o número imaginário j utilizado amplamente neste trabalho.

$$\overline{\overline{R}}_{S_j}^{S_i}(\beta, \alpha) = \overline{\overline{R}}_y(-\alpha)\overline{\overline{R}}_z(-\beta) \tag{1.23}$$

1.1.1 Coordenadas esféricas

Coordenadas esféricas são muito utilizadas no decorrer deste trabalho. Estas são dadas pelos ângulos ϕ e θ e por um raio r, com seus respectivos vetores unitários de direção $\hat{\phi}$, $\hat{\theta}$ e $\hat{r}$, que formam uma base canônica. O uso de coordenadas esféricas e suas aplicações pode ser estudado em maiores detalhes em Moon e Spencer (1961).

Desta forma, sejam $\vec{P}$ um ponto arbitrário no espaço e $\vec{V}$ um campo sobre o ponto $\vec{P}$, $\vec{V}$ em coordenadas cartesiandas é escrito como

$$\vec{V} = \vec{V}(x, y, z) = V_x\hat{x} + V_y\hat{y} + V_z\hat{z} \tag{1.24}$$

Este mesmo vetor pode ser expresso em coordenadas esféricas como

$$\vec{V} = \vec{V}(r, \theta, \phi) = V_r\hat{r} + V_\theta\hat{\theta} + V_\phi\hat{\phi} \tag{1.25}$$

onde V_r, V_θ e V_ϕ são funções de r, θ e ϕ.

Naturalmente, as coordenadas esféricas se relacionam com as coordenadas cartesianas x, y e z pelas Equações 1.28, 1.27 e 1.26, onde $\arctan_2$ é definida como a função arctan contabilizando no ângulo a posição dos argumentos nos quadrantes e sgn é a função sinal que retorna 1 quando o argumento é positivo ou nulo e -1 quando negativo. Os ângulos ϕ e θ são chamados de ângulos de apontamento, pois são utilizados para definir a direção do vetor unitário $\hat{r}$.

$$r = \sqrt{x^2 + y^2 + z^2} \tag{1.26}$$

$$\phi = \begin{cases} \arctan_2(y, x), |x| > 0 \\ 90°\text{sgn}(y), \text{caso contrário} \end{cases} \tag{1.27}$$

$$\theta = \begin{cases} \arctan\left(\frac{\sqrt{y^2+x^2}}{z}\right), |z| > 0 \\ 90°\text{sgn}(y), \text{caso contrário} \end{cases} \tag{1.28}$$

$$\hat{r} = \sin(\theta)\cos(\phi)\hat{x} + \sin(\theta)\sin(\phi)\hat{y} + \cos(\theta)\hat{z} \tag{1.29}$$

$$\hat{\theta} = \cos(\theta)\cos(\phi)\hat{x} + \cos(\theta)\sin(\phi)\hat{y} - \sin(\theta)\hat{z} \tag{1.30}$$

$$\hat{\phi} = -\sin(\phi)\hat{x} + \cos(\phi)\hat{y} \tag{1.31}$$

Assim, um ponto arbitrário $\vec{P}$ qualquer pode ser escrito como

$$\vec{P} = \vec{P}(x, y, z) = \vec{P}(r, \phi, \theta) \tag{1.32}$$

$$\vec{P} = r\sin(\theta)\cos(\phi)\hat{x} + r\sin(\theta)\sin(\phi)\hat{y} + r\cos(\theta)\hat{z} = r\hat{r} \tag{1.33}$$

É interessante calcular ϕ a partir do vetor de direção $\hat{\phi}$ quando possível, pois assim não é preciso se preocupar com as discontinuidades dos polos. Esta operação é demonstrada na Equação 1.35.

$$\hat{\phi} = x_\phi\hat{x} + y_\phi\hat{y} \tag{1.34}$$

$$\phi = \arctan_2(y_\phi, x_\phi) \tag{1.35}$$

$\vec{P}$ pode ser representado na forma matricial diretamente em coordenadas esféricas, fazendo implícitos os vetores unitários $\hat{r}$, $\hat{\theta}$ e $\hat{\phi}$, conforme ilustrado nas Equações 1.36 a 1.33.

$$\vec{P}(x, y, z) = x\hat{x} + y\hat{y} + z\hat{z} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{1.36}$$

$$\vec{P}(r,\phi,\theta) = r\hat{r} = \begin{bmatrix} r \\ 0 \\ 0 \end{bmatrix} \tag{1.37}$$

A Figura 1 mostra o uso de coordenadas esféricas para definição da posição de um ponto arbitrário em relação à uma antena localizada na origem. Nesta figura, τ representa o ângulo que orienta a direção de oscilação da projeção do campo elétrico sobre a esfera.

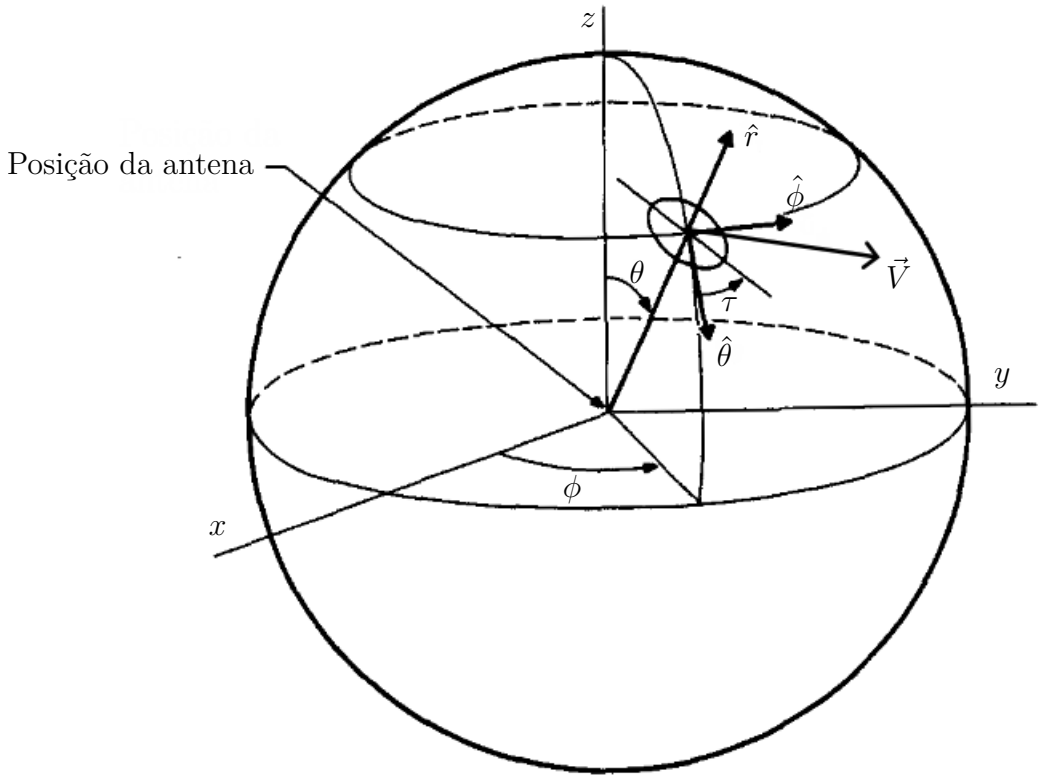


Figura 1 – Aplicação de coordenadas esféricas em uma antena

1.2 Equações de Maxwell e Radiação

Segundo James Clerk Maxwell (2023, 1861), o campo elétrico $\vec{\mathscr{E}}$ e o campo magnético $\vec{\mathscr{H}}$ são campos vetoriais no espaço em função do tempo, com componentes em cada direção, conforme as equações 1.38 e 1.39.

$$\vec{\mathscr{E}} = \vec{\mathscr{E}}(x,y,z,t) = \mathscr{E}_x(x,y,z,t)\hat{x} + \mathscr{E}_y(x,y,z,t)\hat{y} + \mathscr{E}_z(x,y,z,t)\hat{z} \tag{1.38}$$

$$\vec{\mathscr{H}} = \vec{\mathscr{H}}(x,y,z,t) = \mathscr{H}_x(x,y,z,t)\hat{x} + \mathscr{H}_y(x,y,z,t)\hat{y} + \mathscr{H}_z(x,y,z,t)\hat{z} \tag{1.39}$$

Assim como o som transporta energia por meio de vibrações em materiais sólidos e ondas de pressão em meios gasosos, os campos elétrico e magnético permitem o transporte de energia por meio das ondas eletromagnéticas, ou radiação eletromagnética, que se propaga sobre eles mesmo na ausência de matéria, como no vácuo do espaço. Os campos

elétrico e magnético são comumente chamados de campo eletromagnético, pois estão intimamente ligados e se comportam como um só em muitas aplicações do eletromagnetismo. De acordo com Stutzman (1981) e Pozar (1998), o ponto de partida para o uso do campo eletromagnético são as equações de Maxwell do eletromagnetismo: A Lei de Faradei da indução 1.40; Lei de Àmpere com a correção de Maxwell 1.41; Lei de Gauss 1.42, com a existência de carga elétrica; Lei de Gauss para o magnetismo 1.43, significando a não existência de carga magnética; A Equação da continuidade 1.44; e a densidade de corrente total 1.45; escritas na sua forma diferencial. Estas equações diferenciais relacionam os campos elétrico e magnético, a densidade de corrente total $\vec{\mathscr{J}}_\tau$ e densidade de cargas ρ_τ. Os vetores $\vec{\mathscr{D}}$ e $\vec{\mathscr{H}}$ são respectivamente proporcionais a $\vec{\mathscr{E}}$ e $\vec{\mathscr{B}}$ e introduzem os efeitos do material. No vácuo, $\vec{\mathscr{B}} = \vec{\mathscr{H}}/\mu_0$ e $\vec{\mathscr{D}} = \vec{\mathscr{E}}/\epsilon_0$.

$$\nabla \times \vec{\mathscr{E}} = -\frac{\partial}{\partial t}\vec{\mathscr{H}} \tag{1.40}$$

$$\nabla \times \vec{\mathscr{B}} = \frac{\partial}{\partial t}\vec{\mathscr{D}} + \vec{\mathscr{J}}_\tau \tag{1.41}$$

$$\nabla \cdot \vec{\mathscr{D}} = \rho_{\tau,t} \tag{1.42}$$

$$\nabla \cdot \vec{\mathscr{H}} = 0 \tag{1.43}$$

$$\nabla \cdot \vec{\mathscr{J}}_\tau = -\frac{\partial}{\partial t}\rho_{\tau,t} \tag{1.44}$$

$$\vec{\mathscr{J}}_\tau = \sigma\vec{\mathscr{E}} + \vec{\mathscr{J}} \tag{1.45}$$

A densidade de corrente total é definida pela Equação 1.45, onde $\vec{\mathscr{J}}$ é a corrente de excitação e $\sigma\vec{\mathscr{E}}$ é uma corrente induzida em materiais condutores próximos pelos campos eletromagnéticos emitidos por $\vec{\mathscr{J}}$. Ainda segundo Stutzman (1981), em problemas envolvendo eletromagnetismo a corrente de excitação é normalmente uma variável conhecida, e os campos eletromagnéticos devem ser determinados a partir desta, e, por fim, as correntes induzidas podem ser determinadas a partir dos campos eletromagnéticos. Quando a corrente de excitação $\vec{\mathscr{J}}$ oscila com frequência angular ω e amplitude constantes, os campos excitados por esta corrente também oscilam com ω e com amplitudes constantes. Por isso convém utilizar fasores para descrever os campos, de forma que o campo elétrico pode ser representado pela Equação 1.46 e o campo magnético pela Equação 1.47.

$$\vec{\mathscr{E}} = \mathrm{Re}\{\vec{E}\mathrm{e}^{j\omega t}\} \tag{1.46}$$

$$\vec{\mathscr{H}} = \mathrm{Re}\{\vec{H}\mathrm{e}^{j\omega t}\} \tag{1.47}$$

$$\vec{\mathscr{D}} = \mathrm{Re}\{\vec{D}\mathrm{e}^{j\omega t}\} \tag{1.48}$$

$$\vec{\mathscr{B}} = \mathrm{Re}\{\vec{B}\mathrm{e}^{j\omega t}\} \tag{1.49}$$

$$\vec{\mathscr{J}}_\tau = \mathrm{Re}\{\vec{J}_\tau\mathrm{e}^{j\omega t}\} \tag{1.50}$$

$$\nabla \times \vec{E} = -j\omega\vec{H} \tag{1.51}$$

$$\nabla \times \vec{B} = j\omega\vec{D} \times \vec{J_\tau} \tag{1.52}$$

$$\nabla \cdot \vec{D} = \rho_{\tau,f} \tag{1.53}$$

$$\nabla \cdot \vec{H} = 0 \tag{1.54}$$

$$\nabla \cdot \vec{J_\tau} = -j\omega\rho_{\tau,f} \tag{1.55}$$

Desta forma, os vetores são separados em uma parte que varia com o tempo ($e^{j\omega t}$) em regime harmônico, e uma parte que depende apenas da posição espacial $\vec{E} = \vec{E}(x, y, z)$ e $\vec{H} = \vec{H}(x, y, z)$. Assim, as equações de Maxwell podem ser simplificadas para a forma 1.51 a 1.55. Este processo é semelhante à aplicação da transformada de laplace às equações originais, onde as derivadas temporais são substituídas pela variável complexa s (domínio de Laplace), ou, neste caso, $j\omega$ (domínio da frequência). As Equações 1.51 a 1.54 são conhecidas como a forma fasorial das equações de Maxwell e a Equação 1.55 como a forma fasorial da equação da continuidade.

Com a representação em fasores, o campo elétrico e magnético são vetores que oscilam no tempo com uma frequência angular $\omega = 2\pi f$. Cada componente do vetor é um número complexo, onde o módulo deste número representa a amplitude de oscilação na respectiva direção, e o argumento representa a fase da onda em relação a uma referência. Esta referência é normalmente a corrente de alimentação do sistema, que causa a oscilação dos campos. A representação em fasores facilita os cálculos com os campos magnéticos por causa das propriedades de multiplicação de números complexos, sendo possível escalar e aplicar uma defasagem ao fasor facilmente.

Das equações de Maxwell, percebe-se que o campo elétrico e magnético estão intrinsicamente ligados. Conforme Pozar (1998), com as propriedades de um campo e do material, é possível determinar as propriedades do outro campo. Assim, não é possível codificar informações em um campo separadamente do outro, e não há necessidade de se preocupar sempre com ambos os campos. Para o projeto de antenas, é comum trabalhar apenas com o campo elétrico, e interesssante expressar as equações e propriedades de antenas em função apenas do campo elétrico emitido por esta.

1.2.1 Vetor de Poynting

É necessário ainda introduzir o Vetor de Poynting, denotado $\vec{S}$, dado em regime harmônico pela Equação 1.56, onde $\vec{H}^*$ é o conjugado complexo do vetor de campo magnético. O vetor de Poynting define a direção que a onda eletromagnética viaja, e sua magnitude define a densidade de potência contida na onda, que pode ser complexa. A Figura 2 apresenta uma visualização de uma onda eletromagnética, ilustrando os conceitos do campo magnético, campo elétrico e direção de propagação da onda.

$$\vec{S} = \frac{\vec{E} \times \vec{H}^*}{2} \tag{1.56}$$

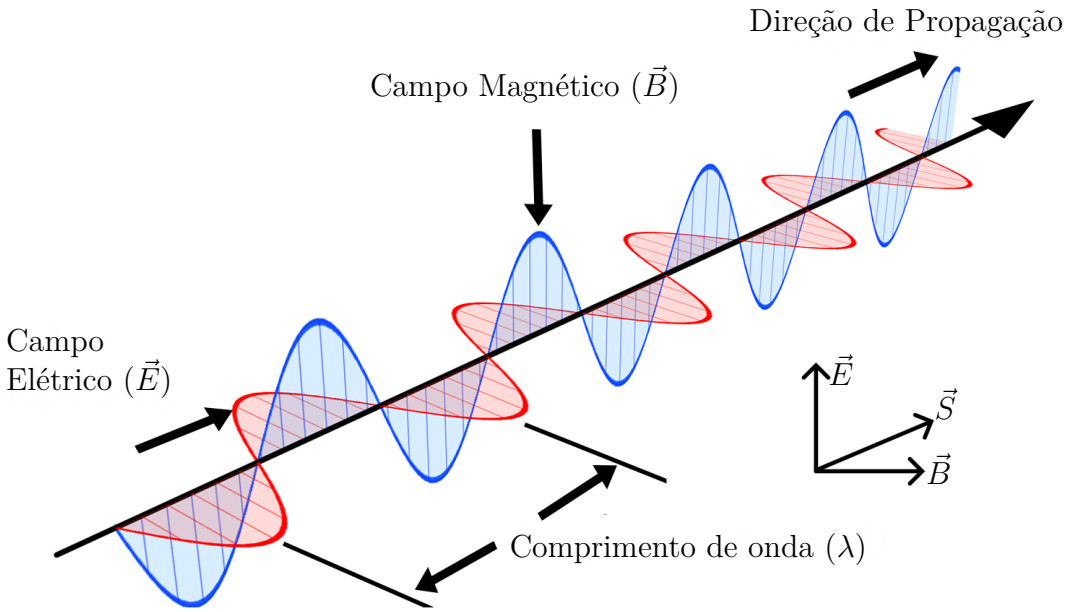


Figura 2 – Ilustração de uma onda eletromagnética

1.2.2 Vetor de onda

A constante de onda k_0 e a direção do vetor de Poynting definem um vetor chamado de vetor de onda $\vec{k}$. Este vetor representa apenas a direção de propagação da onda, mas aparece nas soluções de problemas analíticos e especialmente no cálculo do campo elétrico resultante de arranjos de antenas. O subíndice 0 denota, assim como nas demais constantes deste trabalho, que se trata da onda eletromagnética propagando no espaço livre. Esta é uma simplificação válida para a maioria dos sistemas de comunicação, pois as ondas se propagam normalmente no espaço livre ou no ar, que tem permissividade elétrica e permeabilidade magnética muito próximas do espaço livre. Ondas eletromagnéticas que se propagam fora do espaço livre conservam sua frequência, mas tem velocidade de propagação menor e consequentemente um comprimento de onda menor. Estes efeitos não são considerados neste trabalho para efeito de simplificação, e podem ser facilmente corrigidos caso a aplicação necessite.

$$k_0 = \omega / c_0 \tag{1.57}$$

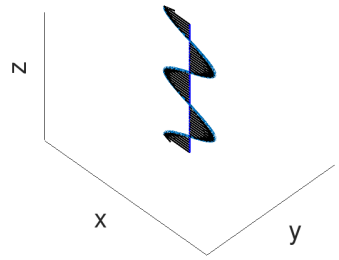
$$\vec{k}(r, \theta, \phi) = k_0 \frac{\vec{S}(r, \theta, \phi)}{|\vec{S}(r, \theta, \phi)|} \tag{1.58}$$

1.2.3 Polarização

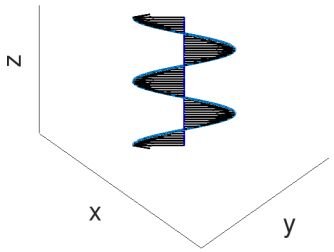
A Figura 2 mostra também outro parâmetro que caracteriza uma onda eletromagnética, a direção de oscilação do campo elétrico. É visível que, a medida que a onda se propaga, seu campo elétrico oscila em uma dada direção. Esta direção é chamada de polarização da onda. As ondas eletromagnéticas podem ter diferentes polarizações. Duas ondas eletromagnéticas com polarizações ortogonais podem transmitir informação de forma independente uma da outra. Assim, para uma mesma frequência, a polarização da onda permite enviar o dobro de informação.

Existem diferentes tipos de polarização, linear, circular, e elíptica. Polarização linear é dada por um campo elétrico oscilando sobre um eixo paralelo à direção de propagação da onda. A polarização circular é caracterizada pelo campo elétrico oscilar de forma circular sobre um plano perpendicular à direção de propagação da onda. A polarização elíptica é um caso genérico da uma oscilação circular em que um eixo de oscilação é maior que outro, fazendo com que o vetor de campo elétrico realize uma trajetória elíptica. Estas polarizações podem ser visualizadas na Figura 3.

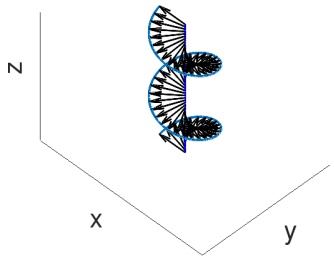
Cada tipo de polarização linear ou circular pode ser decomposta em duas componentes. As componentes da polarização linear podem ser chamadas de vertical e horizontal ou referencial e cruzada, dependendo do autor e da aplicação. A escolha da orientações das polarizações horizontal ou vertical, ou referencial e cruzada, é arbitrária, sendo definida uma componente, a outra componente é determinada perpendicular à primeira e à direção de propagação da onda. As componentes da polarização circular normalmente são chamadas de circular de mão esquerda ou direita, dependendo da orientação da rotação da onda em relação à direção de propagação da mesma. Na Figura 3, estas diferentes polarizações e suas componentes são representadas. Este trabalho utiliza a convenção da onda de polarização circular com referência no transmissor. Diferentes autores utilizam a convenção com referencial no receptor, trocando o significado das duas polarizações. Na Figura 3 a onda eletromagnética é vista saindo do plano representado, se propagando em direção ao leitor. Isto é melhor visualizado na Figura 4, que mostra a polarização circular direita e esquerda em relação à direção de propagação da onda de acordo com a convenção utilizada neste trabalho. Por praticidade, as polarizações circulares são normalmente abreviadas para circular direita (RHCP) e circular esquerda (LHCP).



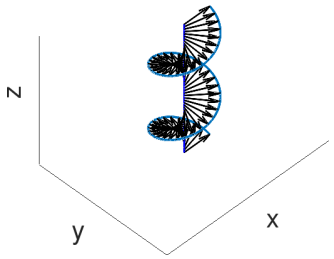
(a) Vetores de campo elétrico com polarização linear horizontal



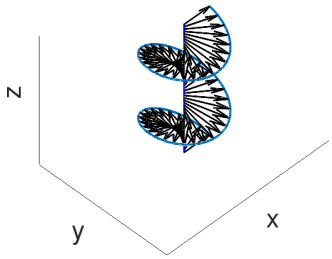
(b) Vetores de campo elétrico com polarização linear diagonal



(c) Vetores de campo elétrico com polarização circular de mão esquerda



(d) Vetor de campo elétrico com polarização circular de mão direita



(e) Vetores de campo elétrico com polarização elíptica de mão direita

Figura 3 – Polariações do vetor de campo elétrico

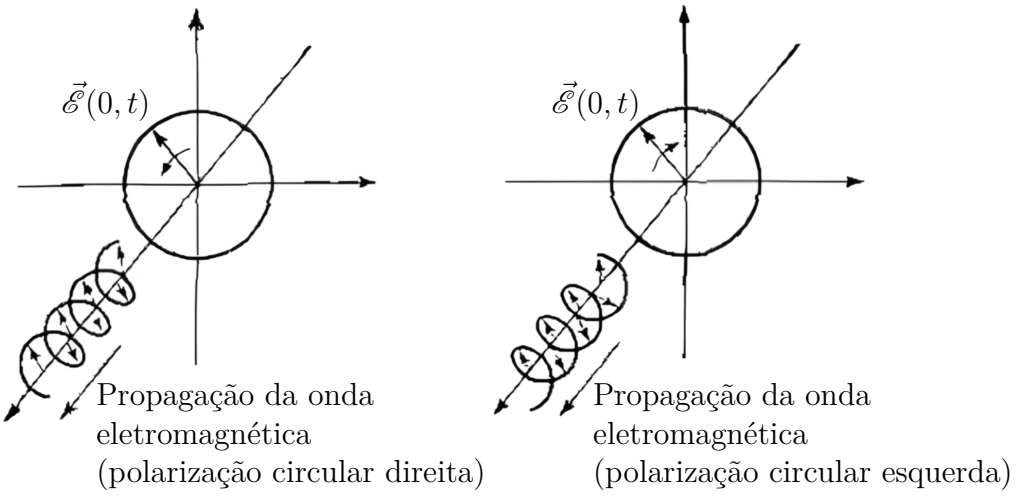


Figura 4 – Ilustração de ondas eletromagnéticas de polarizações circular direita e circular esquerda

Das Figuras 3 e 4, é visível que, a medida que a onda se propaga na direção $\hat{z}$ positiva, o campo elétrico no plano xy descreve uma trajetória circular ou elíptica. Na polarização circular direita, o campo elétrico quando observado a partir do eixo z positivo gira no sentido anti-horário, assim respeitando a "regra da mão direita"formada pela direção de propagação da onda e o sentido de rotação do campo elétrico. De forma similar, a onda de polarização circular esquerda descreve a "regra da mão esquerda".

Definidas as equações que regem a propagação de ondas eletromagnéticas em regime harmônico, é possível determinar os campos elétricos resultantes de antenas ou arranjos destas. Desta forma, eesolver um problema com radiação eletromagnética significa determinar o campo elétrico expresso normalmente em coordenadas esféricas $\vec{E} = \vec{E}(\theta, \phi, r)$ emitido por uma antena e avaliado sobre um ponto arbitrário $\vec{P} = \vec{P}(\theta, \phi, r)$, podendo assim calcular a magnitude e direção do campo elétrico em qualquer ponto do espaço. Muitas vezes as equações resultam em sistemas muito difíceis ou mesmo impossíveis de serem resolvidos analíticamente, e métodos numéricos que discretizam o espaço são empregrados e resolvem para cada elemento espacial as mesmas equações, mas para uma geometria muito mais simples, e o resultado se aproxima do que seria determinado analiticamente.

1.3 Antena Dipolo

Segundo Graf (1999) e Balanis (2005), antenas são um meio de conversão de energia elétrica em energia eletromagnética que se propaga no espaço na forma de ondas eletromagnéticas, ou seja, radiação eletromagnética. Antenas são utilizadas para transferir informação por meio de sinais eletromagnéticos, constituindo um sistema de comunicação. A propagação da informação contida na onda pode ser realizada com ondas com polari-

zação linear ou circular. Ondas com polarização linear são mais simples de serem criadas, enquanto polarizações circulares agregam complexidade à antena e ao sistema, mas contém o benefício de uma simetria radial que pode simplificar a transmissão e, então, serem mais interessantes que polarizações lineares em certas circunstâncias.

As antenas de dipolo são antenas simples e baratas, são um ótimo ponto de partida para o entendimento do funcionamento de antenas e desenvolvimento de antenas mais complexas. Assim, se faz interessante a derivação das equações do campo elétrico e energia irradiada por uma antena dipolo ideal de comprimento arbitrário L, sob circunstâncias ideais. A radiação eletromagnética é emitida radialmente a partir do dipolo, sendo constante em ϕ e variando em θ de forma que nenhuma energia é emitida na direção do eixo $\hat{z}$. (STUTZMAN, 1981)

O Dipolo infinitesimal é utilizado como ponto de partida para o desenvolvimento de antenas dipolo reais, que, por sua vez, são utilizadas para o desenvolvimento de antenas mais complexas. Os campos eletromagnéticos emitidos por uma antena dipolo podem ser determinados integrando os campos eletromagnéticos do dipolo infinitesimal. Assim, se tem uma boa aproximação dos campos eletromagnéticos emitidos por dipolos reais, quando avaliados a grandes distâncias. As antenas de dipolo reais são formadas por um dipolo e elementos para excitá-lo.

1.3.1 Dipolo Infinitesimal

O ponto de partida para a derivação do campo elétrico de um dipolo ideal é o dipolo infinitesimal, também conhecido como *Hertzian Dipole*, formado por uma corrente elétrica I de comprimento infinitesimal $l << \lambda$, posicionado na origem do sistema cartesiano, em espaço livre e orientado no eixo $\hat{z}$. Os campos eletromagnéticos produzidos pelo dipolo infinitesimal, avaliados em um ponto dado pelas coordenadas esféricas θ, ϕ e r, podem ser determinados a partir das equações de Maxwell (BALANIS, 2005), resultando nas Equações 1.60 e 1.61, onde η_0 é a impedância intrínsica do espaço livre. O dipolo infinitesimal é utilizado como elemento infinitesimal na modelagem dos campos eletromagnéticos gerados por uma antena verdadeira ou um circuito qualquer, de modo que é possível determinar os campos eletromagnéticos destes integrando o dipolo infinitesimal. Como os campos de antenas reais são formados pelos campos de dipolos infinitesimais, as propriedades dos dipolos infinitesimais se extendem às antenas reais. (FURSE, 2007)

$$\eta_0 = \sqrt{\frac{\mu_0}{\epsilon_0}} \tag{1.59}$$

$$\vec{E} = \frac{\eta_0 I l}{2\pi}\left(\frac{1}{r^2} + \frac{1}{jk_0 r^3}\right)\mathrm{e}^{-jk_0 r}\cos(\theta)\hat{r} + \frac{j\eta_0 k_0 I l}{4\pi}\left(\frac{1}{r} + \frac{1}{jk_0 r^2} - \frac{1}{k_0^2 r^3}\right)\mathrm{e}^{-jk_0 r}\sin(\theta)\hat{\theta} \tag{1.60}$$

$$\vec{H} = \frac{1}{4\pi} Il \sin(\theta) \left(\frac{jk_0}{r} + \frac{1}{r^2} \right) \mathrm{e}^{-jk_0 r} \hat{\phi} \tag{1.61}$$

É interessante notar que, enquanto o campo magnético produzido pelo dipolo infinitesimal tem componentes apenas na direção $\hat{\phi}$, o campo elétrico tem componentes nas direções $\hat{\theta}$ e $\hat{r}$. Dado que o vetor de Poynting é calculado com o produto vetorial dos campos elétrico e magnético, naturalmente o vetor de Poynting tem direção perpendicular aos campos elétrico e magnético, ou seja, o vetor de Poynting tem componentes nas direções $\hat{r}$ e $\hat{\theta}$. Como o vetor de Poynting define a direção que a onda eletromagnética se propaga, sua componente radial é a parte da radiação que viaja na direção radial, se afastando da fonte de excitação. Esta parte do vetor de Poynting define a potência que é irradiada e dissipada pela antena para o meio, similar à potência dissipada por resistores em circuitos eletrônicos, e é a energia utilizada para a transmissão de informação. Já a componente na direção $\hat{\theta}$ não consegue se afastar da fonte de excitação, e permanece presa ao redor da antena. Esta parte da energia é armazenada ao redor da antena. O vetor de Poynting contém ainda dimensões de potência real e imaginária. A potência real do vetor de Poynting é mais uma vez análoga à potência real de um resistor, enquanto a potência imaginária, chamada de potência reativa, é potência armazenada pela antena, assim como a potência armazenada em capacitores e indutores em circuitos elétricos.

A partir destas equações, é possível identificar duas regiões no espaço onde os campos se comportam de maneiras diferentes, separadas pela distância r até o dipolo infinitesimal, a regiões de campos próximos, campos intermediários e de campos distantes.

Quando $r \ll \lambda$, os termos com maiores potências de $1/r$ dominam, dando origem aos campos próximos (*near fields*). Os campos evanescentes[1] de uma antena podem interagir com antenas vizinhas se estiverem muito próximas. Os campos próximos possuem a capacidade de armazenar energia e são importantes em aplicações com alta transmissão de energia, assim, são muito utilizados em aplicações visando transmissão de energia em curtas distâncias com ondas eletromagnéticas, como fornos a indução e carregadores sem fio de aparelhos eletrônicos. Para tal, é necessário utilizar antenas transmissoras e receptoras muito próximas entre si, o que normalmente não é interessante para sistemas de comunicação, desta forma, não é o caso das aplicações deste trabalho. (BALANIS, 2005; FURSE, 2007)

Quando a distância radial é próxima o comprimento de onda, mais especificamente $r = \lambda_0/2\pi$, os termos de primeira e terceira ordem na equação do campo elétrico tendem a se cancelar, e assim os termos de segunda ordem se tornam preodminantes. A esfera de raio $r = \lambda_0/2\pi$ é denominada esfera radial, e a região do espaço próxima à esta esfera é chamada de região de campos intermediários. Nesta região, a potência reativa da

[1] Oscilações dos campos eletromagnéticos que não propagam, mas se concentram nas proximidades da antena (JACKSON, 1962)

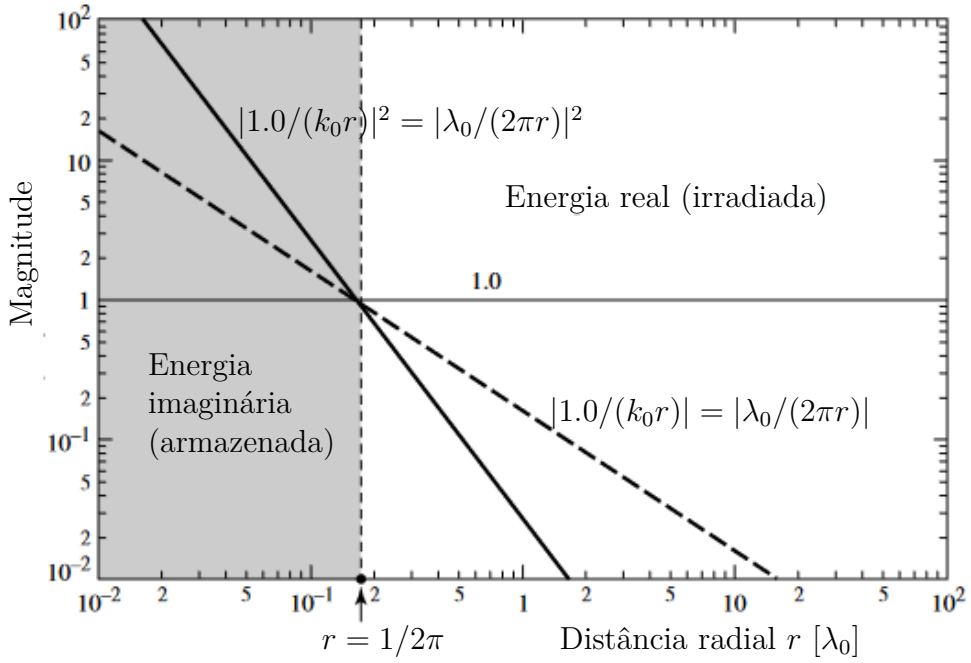


Figura 5 – Campos eletromagnéticos em relação à distância da fonte de radiação

antena é predominante, assim, esta região armazena energia e ainda é interessante para transmissão de energia, mas também não é o foco deste trabalho por envolver distâncias muito pequenas para a maioria dos sistemas de comunicação. (BALANIS, 2005)

Os termos proporcionais a $1/r^2$ e $1/r^3$ decaem rapidamente a medida que r aumenta. Para $r \gg \lambda_0$, os termos de ordens maiores podem ser desprezados e as equações simplificadas se tornam 1.62 e 1.63. Os campos eletromagnéticos destas equações são chamados de campos distantes (*far fields*), ou campos radiantes[2], pois são capazes de irradiar energia. A separação entre campos próximos, radiais e distantes pode ser vista na Figura 5, onde a esfera radial pode ser considerada a barreira que separa os campos próximos dos campos distantes. (BALANIS, 2005)(FURSE, 2007)(ORFANIDIS, 1999)

As Equações 1.62 e 1.63 descrevem os campos eletromagnéticos na região de campos distantes gerados por dipolos reais de comprimento $l << \lambda_0$.

$$\vec{E} = \frac{j\eta_0 Il}{4\pi r}\sin(\theta)\mathrm{e}^{-jk_0r}\hat{\theta} \tag{1.62}$$

$$\vec{H} = \frac{jIl}{4\pi r}\sin(\theta)\mathrm{e}^{-jk_0r}\hat{\phi} \tag{1.63}$$

Para facilitar os cálculos e definições, os campos eletromagnéticos adiante se referem apenas aos campos distantes da antena, e esta convenção se manterá por todo o trabalho. O meio em que os campos elétricos são avaliados é considerado o espaço livre quando não especificado.

[2] Campos que irradiam a energia emitida da antena

É interessante notar que o módulo do campo magnético é diretamente proporcional ao módulo do campo elétrico, de acordo com a Equação 1.64. Isto é útil pois possibilita calcular o campo magnético e as características da antena diretamente a partir do campo elétrico. Esta equação demonstra como o campo elétrico e o campo magnético estão intrinsicamente ligados, de forma que não é necessário se preocupar em determinar ambos, conforme citado anteriormente.

$$\vec{H} = \frac{1}{\eta_0}\hat{r} \times \vec{E} \tag{1.64}$$

Por fim, substituindo as equações 1.62 e 1.64 na equação 1.56, chega-se no vetor de Poynting do dipolo infinitesimal 1.70.

$$\vec{S} = \frac{\vec{E} \times \vec{H}^*}{2} \tag{1.65}$$

$$\vec{S} = \frac{1}{2}\vec{E} \times \left(\frac{1}{\eta_0}\hat{r} \times \vec{E}\right)^* \tag{1.66}$$

$$\vec{S} = \frac{1}{2}\frac{j\eta_0 k_0}{4\pi r} I l \mathrm{e}^{-jk_0 r} \sin(\theta)\hat{\theta} \times \left(\frac{-jk_0}{4\pi r} I l \sin(\theta)\mathrm{e}^{jk_0 r}(\hat{r} \times \hat{\theta})\right) \tag{1.67}$$

$$\vec{S} = \frac{1}{2}\frac{\eta_0 k_0^2}{16\pi^2 r^2} I^2 l^2 \sin^2(\theta)\left(\hat{\theta} \times (\hat{r} \times \hat{\theta})\right) \tag{1.68}$$

$$\vec{S} = \frac{1}{2}\frac{\eta_0 k_0^2}{16\pi^2 r^2} I^2 l^2 \sin^2(\theta)\left(\hat{\theta} \times \hat{\phi}\right) \tag{1.69}$$

$$\vec{S} = \frac{1}{2}\frac{\eta_0 k_0^2}{16\pi^2 r^2} I^2 l^2 \sin^2\theta\hat{r} \tag{1.70}$$

Que pode ser expresso em função do campo elétrico em campo distante pela Equação 1.71.

$$\vec{S} = \frac{|E|^2}{2\eta_0}\hat{r} \tag{1.71}$$

A Equação 1.71, embora derivada de um dipolo hertziano, continua válida para a radiação em campo distante emitida por qualquer antena. Destas equações, tem-se que o vetor de Poynting e o vetor de onda dos campos eletromagnéticos distantes produzidos por uma antena se propagam exclusivamente radialmente, afastando-se da origem dos sistema de coordenadas esféricas. Outra observação importante é que o vetor de Poynting em campos distantes e em espaço livre ($\eta_0 \in \mathbb{R}$) tem componentes apenas reais, visto que depende do módulo ao quadrado do campo elétrico. Ou seja, apenas potência real se propaga nos campos distantes. Desta forma, o vetor de onda pode ser expresso na forma da Equação 1.72, sempre perpendicular aos vetores de direção $\hat{\phi}$ e $\hat{\theta}$, mostrando que a energia é irradiada na direção $\hat{r}$, se afastando da antena. Esta simplificação é válida apenas

para a radiação em campos distantes, e facilita muito a solução analítica de problemas. Outro requisito para esta simplificação é que a origem do sistema referencial esférico esteja próximo da antena. Mais especificamente, a origem deve ser próxima de um ponto especial denominado centro de fase da antena. Assim, a Equação 1.72 pode ser simplificada se expressa em coordenadas esféricas como na Equação 1.74. Em notação matricial, o vetor de onda pode ser dado pela Equação 1.73 em coordenadas cartesianas e pela Equação 1.75 em coordenadas esféricas. (FURSE, 2007)

$$\vec{k}(\theta,\phi) = k_0 \sin(\theta)\cos(\phi)\hat{x} + k_0 \sin(\theta)\sin(\phi)\hat{y} + k_0 \cos(\phi)\hat{z} \tag{1.72}$$

$$\vec{k} = k_0 \begin{bmatrix} \sin(\theta)\cos(\phi) \\ \sin(\theta)\sin(\phi) \\ \cos(\theta) \end{bmatrix} \tag{1.73}$$

$$\vec{k} = k_0 \hat{r} \tag{1.74}$$

$$\vec{k} = \begin{bmatrix} k_0 \\ 0 \\ 0 \end{bmatrix} \tag{1.75}$$

Estas simplificações tornam as coordenadas esféricas ideais para trabalhar com campos eletromagnéticos distantes, pois as ondas eletromagnéticas sempre se propagam na direção radial, se afastando da fonte. Portanto os campos eletromagnéticos emitidos por antenas são normalmente representados em coordenadas esféricas, dados pelos ângulos ϕ e θ e por um raio r, com seus respectivos vetores unitários de direção $\hat{\phi}$, $\hat{\theta}$ e $\hat{r}$, que formam sua base canônica. (MOON; SPENCER, 1961)

1.3.2 Dipolo ideal

O dipolo ideal é modelado como um condutor infinitamente estreito de comprimento arbitrário, sendo alimentado por uma fonte senoidal de amplitude arbitrária. A distribuição de corrente sobre seu comprimento é medida experimentalmente e pode ser modelada de diferentes formas dependendo do tamanho do dipolo. O dipolo ideal pode ser modelado como um somatório infinito de dipolos infinitesimais posicionados em uma linha reta sobre o comprimento do dipolo ideal. O modelo do dipolo ideal é visualizado na Figura 6.

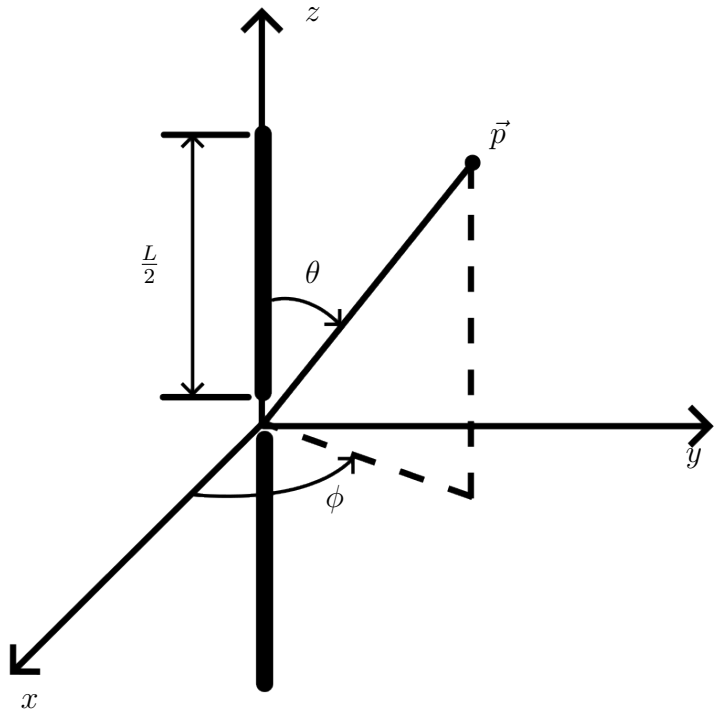


Figura 6 – Modelo do dipolo de comprimento L

Para determinar o campo elétrico de um dipolo com comprimento L próximo de λ, basta integrar o campo elétrico dos dipolos infinitesimais no comprimento do dipolo. Entretanto, para isso é necessário determinar os campos emitidos por um dipolo infinitesimal transladado, e a distribuição de corrente no comprimento do dipolo não é constante, sendo dependente do comprimento do dipolo e a posição onde é avaliada.

A translação do dipolo infinitesimal pode ser entendida como uma translação do sistema referencial por um vetor arbitrário $\vec{p}$. São então definidas as coordenadas cartesianas e esféricas no sistema referencial transladado, x', y', z', ϕ', θ' e r', que, utilizando um ponto $\vec{P}$ arbitrário como ferramenta, se relacionam com as coordenadas do sistema referencial inicial pelas Equações 1.82, 1.83, 1.81.

$$\vec{p} = p_x\hat{x} + p_y\hat{y} + p_z\hat{z} \tag{1.76}$$

$$\vec{P} = x\hat{x} + y\hat{y} + z\hat{z} \tag{1.77}$$

$$\vec{P'} = x'\hat{x} + y'\hat{y} + z'\hat{z} = \vec{P} - \vec{p} = (x - p_x)\hat{x} + (y - p_y)\hat{y} + (z - p_z)\hat{z} \tag{1.78}$$

$$r' = \sqrt{x'^2 + y'^2 + z'^2} = \sqrt{(x - p_x)^2 + (y - p_y)^2 + (z - p_z)^2} \tag{1.79}$$

$$r' = \sqrt{x^2 - 2xp_x + {p_x}^2 + y^2 - 2yp_y + {p_y}^2 + z^2 - 2zp_z + {p_z}^2} \tag{1.80}$$

$$r' = \sqrt{r^2 + |p'|^2 - 2\vec{P} \cdot \vec{p'}} \tag{1.81}$$

$$\phi' = \begin{cases} \arctan_2\left(y - p_y, x - p_x\right), x - p_x \neq 0 \\ 90°\text{sgn}(y - p_y), \text{caso contrário} \end{cases} \tag{1.82}$$

$$\theta' = \begin{cases} \arctan \dfrac{\sqrt{(x - p_x)^2 + (y - p_y)^2}}{z - p_z}, z - p_z \neq 0 \\ 90°, \text{caso contrário} \end{cases} \tag{1.83}$$

Considerando $|\vec{P}| >> |\vec{p}|$, as coordenadas esféricas no referencial transladado se tornam idênticas às coordenadas esféricas do referencial inicial. Assim, as simplificações de campo distante permitem considerar que uma translação do sistema referencial por um vetor arbitrário mantém os mesmos ângulos de apontamento ϕ e θ, quando o deslocamento é pequeno em relação à distância em que o campo está sendo observado.

$$\phi' = \begin{cases} \arctan_2\left(y, x\right), x \neq 0 \\ 90°\text{sgn}(y), \text{caso contrário} \end{cases} \tag{1.84}$$

$$\theta' = \begin{cases} \arctan\left(\dfrac{\sqrt{y^2 + x^2}}{z}\right), z \neq 0 \\ 90°, \text{caso contrário} \end{cases} \tag{1.85}$$

$$\phi' = \phi \tag{1.86}$$

$$\theta' = \theta \tag{1.87}$$

Assim, o campo elétrico produzido por um dipolo infinitesimal, transladado por um vetor $\vec{p}$ e excitado por uma corrente que pode variar de acordo com sua posição $I(\vec{p})$, pode ser aproximado pela Equação 1.88 quando observado à uma distância muito maior que a translação. Nesta equação, I tem direção $\hat{z}$ constante, mas poderia estar em qualquer direção, e o campo elétrico emitido pelo dipolo infinitesimal seria rotacionado de acordo, mas neste problema a rotação não é necessária. A distribuição de corrente $I(\vec{p})$ varia ao longo do comprimento do dipolo. O elemento $\text{e}^{-j\vec{k}\cdot\vec{p}}$ presente nesta equação é um componente muito especial, responsável por corrigir a fase do campo elétrico após a translação.

$$d\vec{E} = j\eta_0 \frac{k_0}{4\pi r} I \sin(\theta) \text{e}^{-jk_0 r} \text{e}^{-j\vec{k}\cdot\vec{p}} dz\hat{\theta} \tag{1.88}$$

Como a integração é apenas sobre o eixo z, a posição do elemento infinitesimal é $\vec{p} = z\hat{z}$ e o comprimento infinitesimal l é substituido por dz, assim:

$$-j\vec{k}\cdot\vec{p} = -j(k_0\sin(\theta)\cos(\phi)\hat{x} + k_0\sin(\theta)\sin(\phi)\hat{y} + \cos(\theta)\hat{z})\cdot z\hat{z} \tag{1.89}$$

$$-j\vec{k}\cdot\vec{p} = -jk_0\cos(\theta)z \tag{1.90}$$

$$d\vec{E} = j\eta_0\frac{k_0}{4\pi r}I(z)\sin(\theta)\mathrm{e}^{-jk_0r}\mathrm{e}^{-jk_0\cos(\theta)z}dz\hat{\theta} \tag{1.91}$$

A distribuição de corrente ao longo do dipolo pode ser determinada experimentalmente. De acordo com Balanis, a Equação 1.92 é uma boa aproximação da distribuição de corrente de um dipolo com comprimento L de magnitude próxima ao comprimento de onda, alimentado em seu centro centro por uma corrente I_0.

$$I(z) = \begin{cases} I_0\sin\left(k_0\left(\frac{L}{2}+z\right)\right), 0 \leq z \leq L/2 \\ I_0\sin\left(k_0\left(\frac{L}{2}-z\right)\right), -L/2 \leq z < 0 \end{cases} \tag{1.92}$$

Finalmente, substituindo a distribuição de corrente e integrando a Equação 1.88 em z de $-L/2$ até $L/2$:

$$d\vec{E}(r,\theta,\phi) = j\eta_0\frac{k_0}{4pir}I(z)\sin(\theta)\mathrm{e}^{-jk_0r}\mathrm{e}^{-jk_0\cos(\theta)z}dz \tag{1.93}$$

$$\vec{E}(r,\theta,\phi) = \int_{-\frac{L}{2}}^{\frac{L}{2}} d\vec{E} = \int_{-\frac{L}{2}}^{\frac{L}{2}} j\eta_0\frac{k_0}{4pir}I(z)\sin(\theta)\mathrm{e}^{-jk_0r}\mathrm{e}^{-jk_0\cos(\theta)z}dz \tag{1.94}$$

$$\vec{E}(r,\theta,\phi) = j\eta_0\frac{k_0}{4\pi r}\sin(\theta)\mathrm{e}^{-jk_0r}\int_{-\frac{L}{2}}^{\frac{L}{2}} I(z)\mathrm{e}^{-jk_0\cos(\theta)z}dz \tag{1.95}$$

Substituindo a Equação da distribuição de corrente 1.92 e definindo Φ^+ e Φ^- de acordo com as Equações 1.96 e 1.97, o campo elétrico se torna a solução da integral da Equação 1.98. Reconhecendo que $\Phi^+(z) = \Phi^-(-z)$, utilizando uma substituição de variável a integral pode ser simplificada para a Equação 1.99.

$$\Phi^+ = \sin\left(k_0\left(\frac{L}{2}+z\right)\right) \tag{1.96}$$

$$\Phi^- = \sin\left(k_0\left(\frac{L}{2}-z\right)\right) \tag{1.97}$$

$$\vec{E} = \frac{j\eta_0 I_0}{4\pi r}\mathrm{e}^{-jk_0r}\sin(\theta)\left[\int_0^{\frac{L}{2}}\Phi^-(z)\mathrm{e}^{-jk_0\cos(\theta)z}dz + \int_{\frac{L}{2}}^{0}\Phi^+(z)\mathrm{e}^{-jk_0\cos(\theta)z}dz\right] \tag{1.98}$$

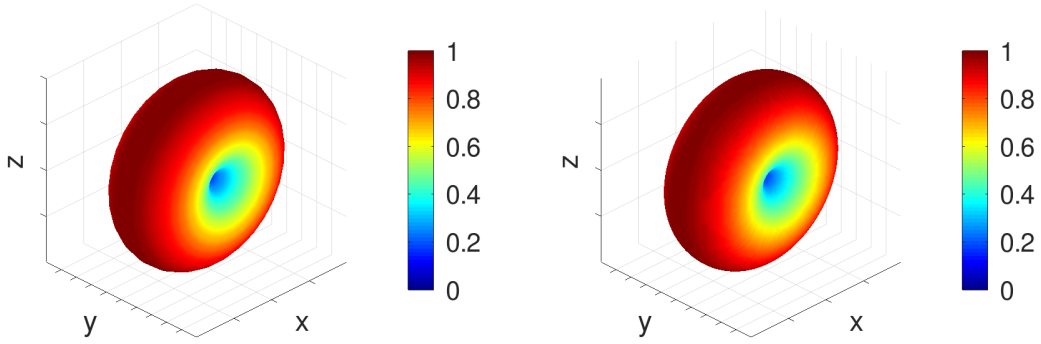
$$\vec{E} = \frac{j\eta_0 I_0}{4\pi r}\mathrm{e}^{-jk_0r}2\sin(\theta)\int_0^{\frac{L}{2}}\Phi^-(z)\mathrm{e}^{-jk_0\cos(\theta)z}dz \tag{1.99}$$

Utilizando a identidade $e^{jw} = \cos(w) + j\cos(w)$ para resolver a integral, chega-se à Equação 1.100 final do campo elétrico do dipolo ideal.

$$\vec{E} = \frac{j\eta_0 I_0}{2\pi r} e^{-jk_0 r} \left[\frac{\cos\left(\frac{k_0 l}{2}\cos(\theta)\right) - \cos\left(\frac{k_0 l}{2}\right)}{\sin(\theta)} \right] \hat{\theta} \tag{1.100}$$

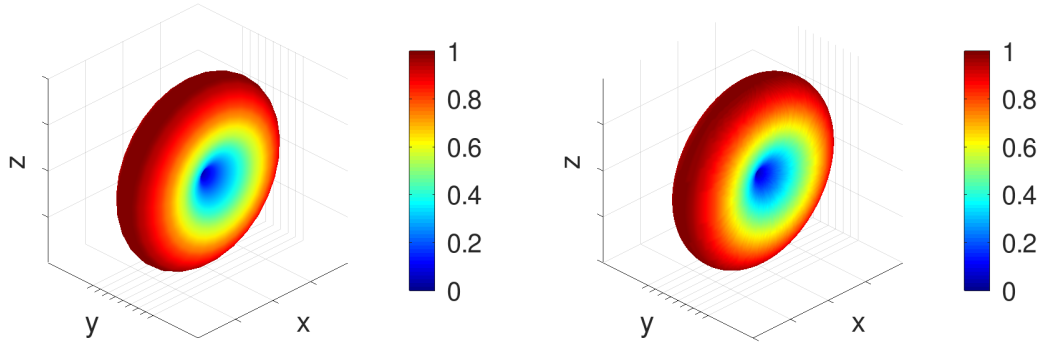
Utilizando a Equação 1.100, foram calculados no *Octave* os campos elétricos irradiados para dipolos de diferentes comprimentos. Uma rápida simulação dos um dipolos no *HFSS* permite comparar os resultados analíticos com os resultados numéricos. A simulação foi executada utilizando uma *lumped port* com frequência de 433 MHz e dipolos com as mesmas dimensões dos utilizados no *Octave*. O comprimento do dipolo é calculado considerando a velocidade da luz em espaço livre, aproximadamente 69 cm para metade do comprimento de onda.

Os resultados estão dispostos nas Figuras 7, 8, 9 e 10, onde é mostrado o diagrama de magnitude do campo elétrico normalizado das antenas dipolo simuladas numéricamente e modeladas analiticamente. A magnitude é normalizada pelo maior valor da magnitude máx$\{|\vec{E}(\theta,\phi)|\}$, sendo uma escala adimensional. É notável a semelhança no formato dos gráficos calculados analiticamente e simulados numericamente, sendo a maior diferença para o dipolo de comprimento $1.5\lambda_0$. Tal diferença pode ser explicada pelo fato de que as simulações numéricas levam em conta imperfeições nas geometrias que não são contabilizadas no modelo analítico, por exemplo a antena simulada numericamente não ter espessura infinitamente pequena, e também pelo fato do resultado analítico assumir uma distribuição de corrente no comprimento do dipolo que não necessariamente é a mais realística.



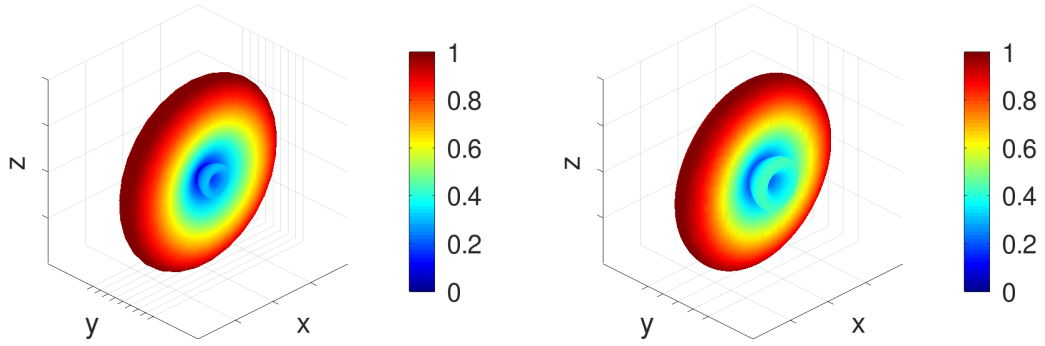
(a) Analítico *Octave* (b) Simulação *HFSS*

Figura 7 – Campo elétrico formado por um dipolo ideal de comprimento $0.5\lambda_0$



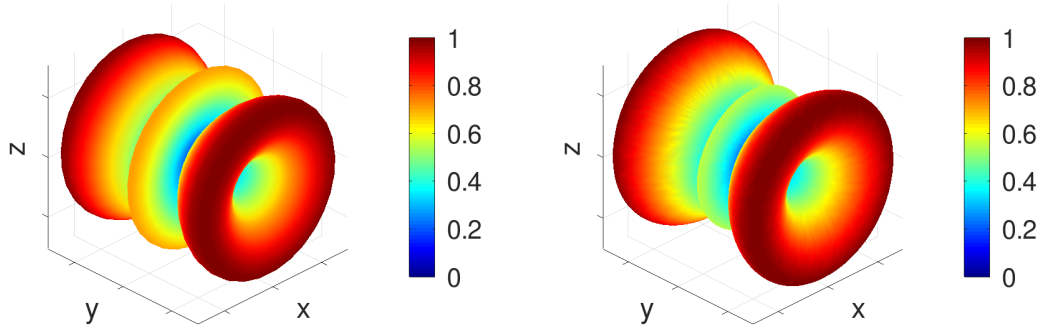
(a) Analítico *Octave* (b) Simulação *HFSS*

Figura 8 – Campo elétrico formado por um dipolo ideal de comprimento $1.0\lambda_0$



(a) Analítico *Octave* (b) Simulação *HFSS*

Figura 9 – Campo elétrico formado por um dipolo ideal de comprimento $1.25\lambda_0$



(a) Analítico *Octave* (b) Simulação *HFSS*

Figura 10 – Campo elétrico formado por um dipolo ideal de comprimento $1.5\lambda_0$

1.3.3 Diagrama de radiação

Pode ser demonstrado, de acordo com Orfanidis (1999), que o campo elétrico distante emitido por qualquer fonte obedece a relação 1.101, onde o termo $\vec{F}(\theta,\phi) = F_\theta(\theta,\phi)\hat{\theta} + F_\phi(\theta,\phi)\hat{\phi}$ é chamado de vetor de radiação, que define a magnitude da potência emitida em uma direção (θ,ϕ), e $E_\infty(r)$ é uma função complexa dependente apenas da distância do distância r do ponto de observação do campo elétrico à antena e tem a forma da Equação 1.102. Esta separação de variáveis fica evidente nas equações de campo distante do dipolo infinitesimal e no dipolo ideal. A função E_∞ pode ser entendida como uma campo elétrico esférico emanando de uma fonte pontual, enquanto o vetor $\vec{F}$ modela a magnitude e fase do campo elétrico sendo emitido em cada direção. O vetor de radiação $\vec{F}(\theta,\phi)$ contabiliza fatores geométricos e controla a distribuição de radiação da antena, enquanto a função $E_\infty(r)$ controla em geral a potência total irradiada e a reflexão da antena. Normalmente o vetor de radiação é normalizado de forma que a Equação 1.103 é satisfeita, para facilitar a comparação dos campos elétricos emitidos por diferentes antenas, e a constante de normalização é compensada na constante E_0. Da Equação 1.62, observa-se que a constante E_0 é proporcional à $\frac{j\eta_0}{4\pi}$, bem como à corrente de alimentação I_0 e ainda da geometria da antena.

$$\vec{E}(r,\theta,\phi) = E_\infty(r)\vec{F}(\theta,\phi) \tag{1.101}$$

$$E_\infty(r) = \frac{E_0}{r}\mathrm{e}^{-jk_0 r} \tag{1.102}$$

$$\text{máx}\{|\vec{F}|\} = 1 \tag{1.103}$$

O controle do projetista sobre função E_∞ é limitado. Definidas as distâncias envolvidas na comunicação entre as antenas, apenas a distribuição de radiação e corrente de alimentação podem ser modificadas para atingir os objetivos do sistema. Assim, o projeto de antenas trabalha principalmente sobre o vetor de radiação representado em coordenadas esféricas. Naturalmente, uma antena mais diretiva necessita de uma corrente de alimentação menor, e por consequência requer menos potência para funcionar. Em campos distantes, o vetor de radiação é sempre perpendicular à direção radial $\hat{r}$ e não depende de r, assim, a coordenada radial pode ser implicitamente nula, pois é redundante para representar o vetor de radiação. Isto torna a representação do vetor de radiação mais prática na forma matricial pois elimina uma dimensão. Os vetores de direção $\hat{\theta}$ e $\hat{\phi}$ formam uma base canônica de um espaço de duas dimensões complexas, e são então representados respectivamente pelas Equações 1.104 e 1.105 em coordenadas esféricas com a coordenada radial implícita. O vetor de radiação é então representado pela Equação 1.106 em coordenadas esféricas.

$$\hat{\theta} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{1.104}$$

$$\hat{\phi} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{1.105}$$

$$\vec{F} = F_\theta \hat{\theta} + F_\phi \hat{\phi} = \begin{bmatrix} F_\theta \\ F_\phi \end{bmatrix} \tag{1.106}$$

Considerando que a potência dada pelo vetor de Poynting diminui conforme a distância radial aumenta, é interessante definir uma potência por unidade de ângulo sólido em campo distante, que é independente da distância radial r. Esta é chamada de intensidade de potência $U(\theta, \phi)$, que é relacionada com a parte real do vetor de Poynting pela Equação 1.107. Dessa forma, a intensidade de potência, independente da distância radial, se torna dependente apenas de $F(\theta, \phi)$, e então dependente apenas da geometria da antena. Utilizando a intensidade de radiação, a potência total irradiada P_{rad} pela antena pode ser determinada integrando U sobre toda a superfície de uma esfera de raio infinito 1.108, onde $d\Omega = \sin(\theta) d\theta d\phi$. Substituindo a intensidade de radiação na integral, a potência total irradiada pode ser obtida, talvez de forma mais intuitiva, integrando a parte real do vetor de Poynting sobre a superfície de uma esfera de raio tendendo ao infinito, reconhecendo o jacobiano de coordenadas esféricas $r^2 \sin(\theta)$, assim confirmando as relações de 1.112 e 1.108. (ORFANIDIS, 1999; POZAR, 1998)

$$U(\theta, \phi) = \lim_{r \to \infty} r^2 |\mathrm{Re}\{\vec{S}(r, \theta, \phi)\}| \tag{1.107}$$

$$P_{rad} = \int_{-\pi}^{\pi} \int_{0}^{\pi} U d\Omega \tag{1.108}$$

$$P_{rad} = \lim_{r \to \infty} \int_{-\pi}^{\pi} \int_{0}^{\pi} r^2 |\mathrm{Re}\{\vec{S}\}| \sin(\theta) d\theta d\phi \tag{1.109}$$

Reconhecendo que, para campos distantes, apenas potência real é propagada, então $|Re\{\vec{S}\}| = |\vec{S}|$:

$$\vec{S} = \frac{1}{\eta_0} |\vec{E}|^2 \hat{r} \tag{1.110}$$

$$\vec{S} = \frac{|E_0|^2 |\vec{F}|^2}{2\eta_0 r^2} \hat{r} \tag{1.111}$$

$$U = \lim_{r \to \infty} \frac{|E_0|^2 |F|^2}{2\eta_0} \tag{1.112}$$

A Equação 1.108 se torna:

$$P_{rad} = \int_{-\pi}^{\pi}\int_{0}^{\pi} \frac{|E_0|^2|F|^2}{2\eta_0} d\theta d\phi \tag{1.113}$$

$$P_{rad} = \frac{|E_0|^2}{2\eta_0}\int_{-\pi}^{\pi}\int_{0}^{\pi} |F|^2 d\theta d\phi \tag{1.114}$$

onde

$$|F|^2 = |F_\theta|^2 + |F_\phi|^2 \tag{1.115}$$

O gráfico tridimensional polar da 1.112 forma uma superfície fechada para a qual o raio é proporcional à intensidade de radiação na direção (θ, ϕ). Este gráfico é chamado de diagrama de radiação, pois ilustra a potência da radiação emitida para cada direção (θ, ϕ). Comumente é utilizado o diagrama de radiação normalizado, obtido ao dividir a intensidade de potência para cada direção (θ, ϕ) pela intensidade máxima do gráfico $U_{máx}$. (ORFANIDIS, 1999)

Um conceito útil é o do radiador isotrópico, que irradia sua potência de forma uniforme em todas as direçõe. Assim, a intensidade isotrópica[3] de potência equivalente $U_{isotropica}$ de uma antena pode ser dada pela Equação 1.116. (ORFANIDIS, 1999)

$$U_{isotrópica} = \frac{P_{rad}}{4\pi} \tag{1.116}$$

$$U_{isotrópica} = \frac{1}{4\pi}\int_{-\pi}^{\pi}\int_{0}^{\pi} U d\theta d\phi \tag{1.117}$$

Visualizações dos conceitos envolvendo diagramas de radiação podem ser encontradas nas Figuras 11, 12 e 13. O diagrama de radiação pode ser um gráfico tridimensional, mostrando para cada direção a potência de radiação emitida, ou um gráfico bidimensional com a projeção do diagrama de radiação sobre um plano arbitrário que passa pela origem, assim mostrando a potência de radiação em função de um ângulo, normalmente o ângulo polar é utilizado.

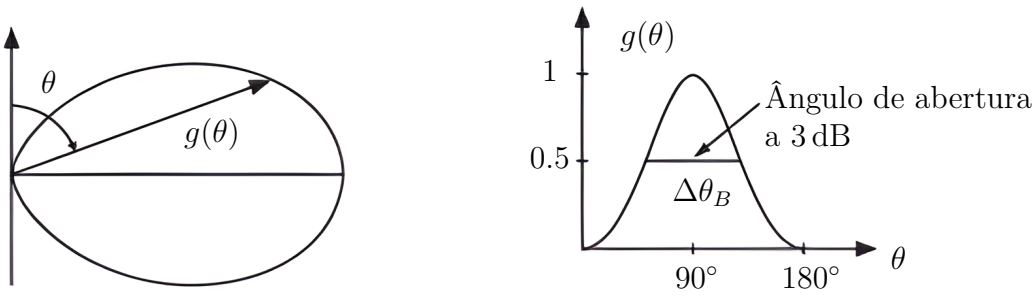


Figura 11 – Projeção do diagrama de radiação sobre um plano e gráfico da potência de radiação pelo o ângulo polar

[3] que possui propriedades físicas que são independentes da direção (diz-se de um meio); isótropo.

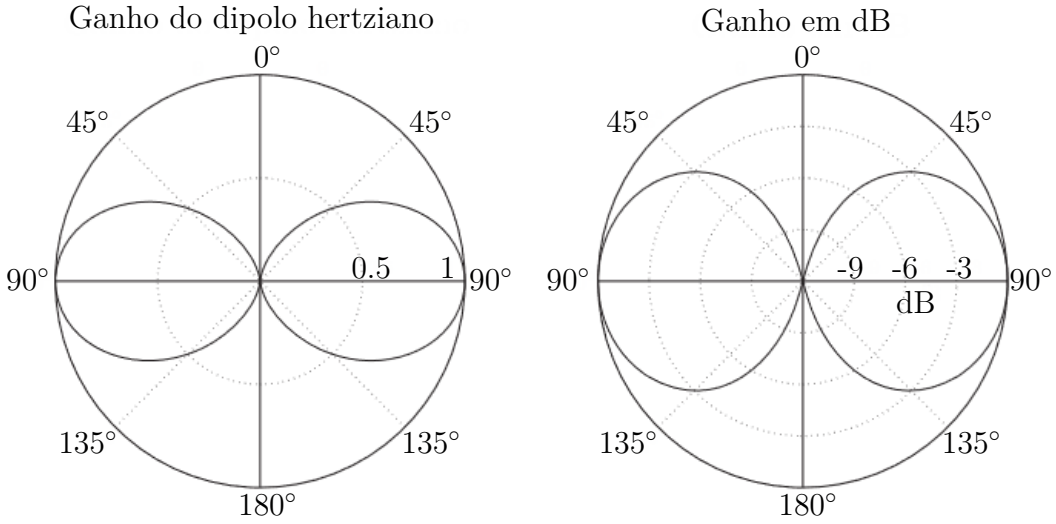


Figura 12 – Diagramas bidimensionais de radiação de um dipolo infinitesimal com potência irradiada normalizada e ganho

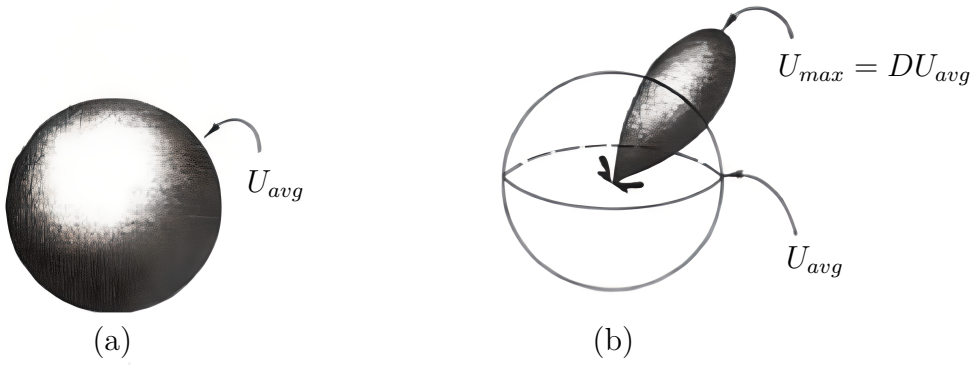


Figura 13 – Diagrama de radiação para uma antena isotrópica (a) e uma antena com alta diretividade (b)

1.3.4 Ângulo de Abertura a Meia Potência

O ângulo de abertura a meia potência é definido como a diferença $\Delta\theta_{3dB}$, onde θ_1 e θ_2 são os ângulos para os quais a intensidade de potência normalizada é -3dB. O valor de -3dB é utilizado pois define o ponto de queda de 50% da intensidade de potência irradiada. Ao determinar o ângulo de abertura a meia potência de uma antena, é possível fazer uma simplificação do diagrama de radiação dessa antena, considerando que a energia é irradiada com sua intensidade máxima constantemente dentro do ângulo de abertura, e nenhuma energia é irradiada fora deste ângulo. Esta simplificação torna mais fácil determinar o apontamento de diversas antenas dentro de um sistema de comunicação, negligenciando os lóbulos secundários quando possível e diminuindo a quantidade de computações a serem realizadas. (ORFANIDIS, 1999)

1.3.5 Diretividade

A diretividade $D(\theta, \phi)$ é definida como a intensidade de potência em uma direção (ϕ, θ), normalizada pela intensidade isotrópica de potência equivalente, de acordo com a Equação 1.118. O valor máximo da diretividade é utilizado como um parâmetro da antena, chamado de diretividade da antena D_{antena}, dada pela Equação 1.119. (ORFANIDIS, 1999)

$$D(\theta, \phi) = \frac{U(\theta, \phi)}{U_{isotrópica}} \tag{1.118}$$

$$D_{antena} = \text{máx}\{D(\theta, \phi)\} \tag{1.119}$$

1.3.6 Ganho

O ganho $G(\theta, \phi)$ de uma antena é definido pela Equação 1.120. G é definido como a intensidade de potência por ângulo sólido (Ω) da antena, normalizado pela potência total aceita pela antena nos terminais de um gerador conectado. Esta potência é, no máximo, metade da potência total do gerador, sendo a outra metade dissipada no gerador na forma de calor. O ganho é propriedade chave para a caracterização da antena, pois leva em consideração perdas ôhmicas na antena, e é uma propriedade mais representativa da eficiência do sistema. Naturalmente, o ganho de uma antena é sempre menor que a diretividade da antena devido à reflexões e perdas na antena, e pode ser determinado pela Equação 1.121, onde e é um fator de eficiência da antena, definido pela Equação 1.122 (ORFANIDIS, 1999). As Figuras 15 e 14 ilustram em um diagrama de radiação as propriedades definidas de Ganho, Diretividade, e ângulos de abertura a meia potência e seu uso para caraterização de um sistema de comunicação.

$$G(\theta, \phi) = \frac{4\pi}{P_T} \frac{dP}{d\Omega} \tag{1.120}$$

$$G(\theta, \phi) = eD(\theta, \phi) \tag{1.121}$$

$$e = \frac{P_{rad}}{P_T} \tag{1.122}$$

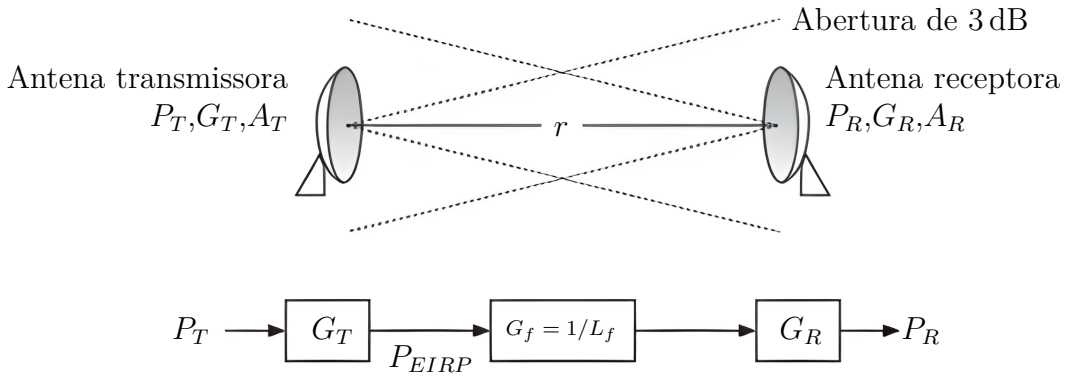


Figura 14 – Sistema de comunicação com uma antena transmissora e uma antena receptora, com seus respectivos ganhos, área equivalente e potência transmitida e recebida

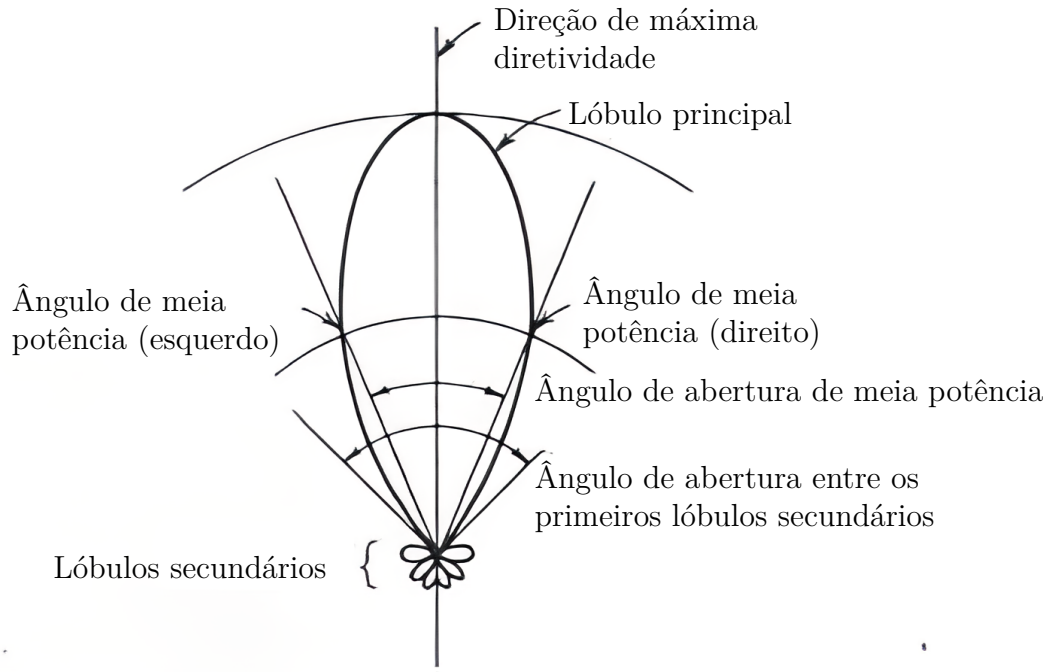


Figura 15 – Diagrama de radiação de antena diretiva, mostrando o lóbulo principal (*main lobe*), lóbulos secundários (*minor lobes*), e abertura a meia potência (*half power beamwidth*)

1.3.7 Polarização

O vetor de radiação $F(\theta, \phi)$ é formado por duas componentes, $F_\phi(\theta, \phi)\hat{\phi}$ e $F_\theta(\theta, \phi)\hat{\theta}$. Estas componentes definem a orientação da oscilação das ondas eletromagnéticas, chamada de polarização da onda, onde F_ϕ é a componente de polarização em $\hat{\phi}$ e F_θ é a componente de polarização em $\hat{\theta}$. Antenas emitem radiações em todas as polarizações, lineares e circulares, entretanto normalmente as antenas tem polarizações preferidas, onde a maior parte da energia é emitida. (ORFANIDIS, 1999)

A polarização circular de uma antena é uma polarização complexa. A definição de polarização direita varia com o autor, neste trabalho, esta é definida do ponto de vista da antena transmissora, ou seja, quando observado de fora da esfera de radiação emitida pela antena, na polarização circular direita, o campo elétrico rotaciona no sentido anti-horário de acordo com a regra da mão direita, enquanto a polarização circular esquerda rotaciona no sentido horário. Em outras palavras, a direção de propagação da onda eletromagnética e o sentido de rotação da polarização direita respeitam a regra da mão direita.

Assim como o vetor de radiação pode ser representado na base canônica $\hat{\theta}$ e $\hat{\phi}$, também pode ser representado na base de polarização circular pelos vetores 1.123 e 1.124, e as componentes de polarização circular podem ser determinadas a partir das componentes de polarização linear da antena pela Equação 1.126 para polarização circular direita e pela Equação 1.127 para a polarização esquerda. Estas operações são apenas uma mudança de base por meio de um produto interno com a nova base, e as componentes do vetor de radiação na nova base representam diretamente as componentes de polarização circular do vetor de radiação.

$$\hat{u}_{rhcp} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -j \end{bmatrix} \tag{1.123}$$

$$\hat{u}_{lhcp} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ j \end{bmatrix} \tag{1.124}$$

$$\vec{F} = F_{rhcp}\hat{u}_{rhcp} + F_{lhcp}\hat{u}_{lhcp} \tag{1.125}$$

$$F_{rhcp} = \vec{F} \cdot \hat{F}^*_{rhcp} \tag{1.126}$$

$$F_{lhcp} = \vec{F} \cdot \hat{F}^*_{lhcp} \tag{1.127}$$

Para ocorrer transmissão de informação entre duas antenas, é necessário que as polarizações de ambas antenas sejam compatíveis, ou seja, ambas tenham polarizações lineares ou circulares. Ademais, é necessário que as duas antenas concordem na orientação das orientações. Por exemplo, se duas antenas tem polarização linear, entretando uma antena tenha polarização linear orientada perpendicularmente à polarização linear aceita pela outra antena, nenhuma energia será transmitida. Normalmente é definido um sistema de coordenadas globais e as polarizações de ambas antenas são representadas neste referencial global, para que seja possível a concordância das polarizações entre as antenas. O mesmo é valido para antenas com polarização circular.

A polarização de antenas transmissoras e receptoras deve ser compatível para que haja transmissão de energia. Definindo $\hat{h}_R = \vec{F}_R/|\vec{F}_R|$ como o vetor de direção de polarização da radiação de uma antena receptora e $\hat{h}_T = \vec{F}_T/|\vec{F}_T|$ como o vetor de direção de

polarização da radiação de uma antena transmmissora, a perda de potência devido ao descasamento das polarizações pode ser determinado pelo fator de perda por desalinhamento de polarização e_{pol}, dado pela equação 1.128, que é válida tanto para polarizações lineares como circulares. Isto significa que se as polarizações das antenas receptora e transmissora não estiverem alinhadas, há perdas na transmissão. (ORFANIDIS, 1999)

$$e_{pol} = |\hat{h}_R \cdot \hat{h}_T|^2 \tag{1.128}$$

É interessante observar que é possível utilizar antenas com polarizações lineares para emitir radiação com polarização circular, orientando ambas as antenas perpendicularmente entre elas e aplicando uma defasagem de 90° no sinal de uma delas. O mesmo é válido para emitir radiação linear utilizando antenas de polarização circular. Isto é útil pois permite utilizar antenas de polarização linear para se comunicar com antenas de polarização circular, e este trabalho se conteve em desenvolver antenas de polarização linear, mas isto não se faz uma limitação para desenvolver um arranjo que emita radiação com polarização circular, que é o objetivo final.

1.3.8 Parâmetros S

Ondas eletromagnéticas transportam energia, e a função de uma antena é receber ou liberar esta energia no ambiente de forma a ser captada por outra antena, assim, são nada mais que um meio de conversão de energia elétrica em eletromagnética e vice-versa. Quando parte da energia não é transformada e irradiada, esta é refletida. No caso de uma antena transmissora, que é alimentada por um cabo coaxial, a energia transmitida sai como ondas eletromagnéticas, enquanto a energia refletida retorna ao cabo coaxial e finalmente ao circuito eletrônico conectado a este. Desnecessário dizer que isto é um fenômeno indesejável. Além de diminuir a eficiência da antena, a energia refletida pode interferir no circuito conectado e potencialmente danifica-lo.

Em simulações de sistemas eletromagnéticos, são criadas superfícies no espaço chamadas de portas. Nestas portas são definidas as excitações do campo elétrico que excitam o sistema, assim, são condições de contorno que injetam energia no sistema. A energia fornecida de uma porta pode chegar à outra porta, assim, o parâmetro S_{ij} é definido como a parte da energia fornecida pela i-ésima porta que é absorvida pela j-ésima porta. Assim, o parâmetro S_{11} é a energia emitida pela porta 1 e retorna à mesma porta, ou seja, é a energia refletida pelo sistema de volta à porta. Em uma antena sendo excitada por uma porta 1 que modela um cabo coaxial, o parâmetro S_{11} quantifica então a energia refletida pela antena de volta para o cabo coaxial, ou seja, é a razão de energia que não é irradiada. Esta propriedade da antena é muitas vezes chamado apenas de parâmetro S ou reflexão da antena. Em uma simulação de antena com apenas uma porta, toda a energia

não refletida deve ter sido irradiada para obedecer a conservação de energia, assim, $1-\mathrm{S}_{11}$ fornece a razão de energia irradiada pela antena. O princípio da reciprocidade mostra que $\mathrm{S}_{ij} = \mathrm{S}_{ji}$, e uma consequência deste princípio é que o parâmetro S de uma antena é o mesmo seja ela atuando como antena transmissora ou receptora. A Equação 1.129 mostra o cálculo do parâmetro S em dB de uma antena, onde $P_{fornecida}$ é a potência fornecida à antena pelo cabo coaxial e $P_{refletida}$ é a potência refletida pela antena para o cabo coaxial.

O parâmetro S deve ser minimizado para garantir um bom uso da energia fonecida à antena. O parâmetro S, que não deve ser confundido com o vetor de Poynting $\vec{S}$, é normalmente determinado por meio de simulações numéricas, e utilizado como um dos parâmetros principais nas simulações de otimização da geometria da antena. O parâmetros S está presente também em antenas receptoras, onde em vez da energia ser absorvida, ela pode ser refletida para o meio.

$$S_{dB} = 20\log_{10}\left(\frac{P_{refletida}}{P_{fornecida}}\right) \tag{1.129}$$

1.4 Balun

Ao conectar os terminais de um cabo coaxial diretamente ao dipolo, boa parte da energia é refletida em vez de irradiada, resultando em um alto parâmetro S. Assim, é adicionado um elemento responsável por fazer a interface entre o cabo e o dipolo. Existe uma grande variedade de geometrias capazes de fazer isto, cada qual com pontos fortes e fracos. Um ponto de partida comum é o *Balun*, e é o elemento utilizado neste trabalho por sua simplicidade.

O *Balun* é uma geometria cilíndrica com comprimento próximo de $\lambda/2$, o desenho da topologia do *Balun* utilizado neste trabalho pode ser visto na Figura 16, embora outras topologias existam (RADIO SOCIETY OF GREAT BRITAIN, 1976; GUANELLA, 1944). As hastes do dipolo são conectadas perpendicularmente na extremidade superior do *Balun*, de onde parte um rasgo de espessura pequena quee se extende até próximo da extremidade inferior. O cabo coaxial é conectado, por um conector N, na extremidade inferior, de forma que o núcleo externo do cabo é conectado ao cilindro externo e o núcleo interno se extende por dentro do *Balun* até se conectar às hastes do dipolo na parte superior. O raio e espessura do cilindro são valores arbitrários, da mesma ordem de magnitude que o raio do cabo coaxial. As dimensões iniciais do *Balun* são aproximações que garantem estar perto de um mínimo local do parâmetro S, que será encontrado numericamente por meio de algoritmos de otimização. A geometria da antena dipolo pode ser visualizada na Figura 17a.

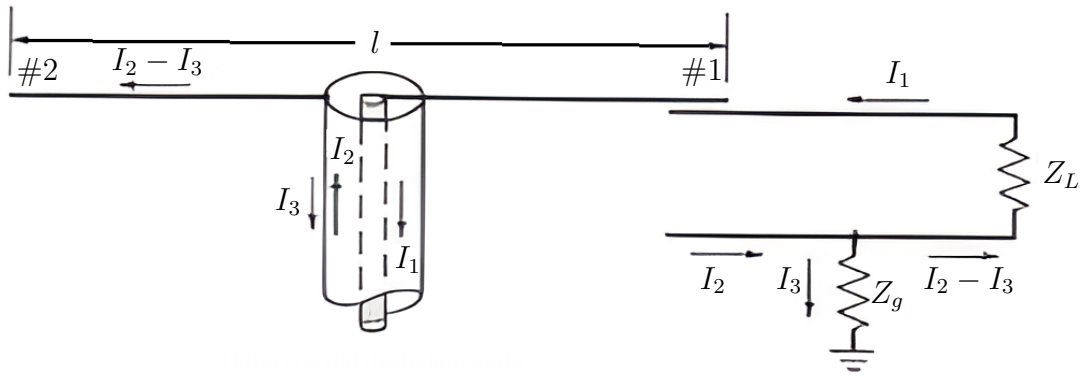


Figura 16 – Geometria do Balun

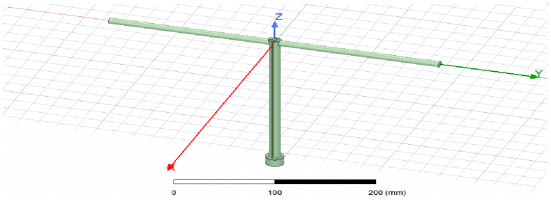
1.5 Antenas Yagi-Uda

As antenas Yagi-Uda, são formadas por uma junção de estruturas que exercem funções específicas. Nas antenas Yagi-Uda, o dipolo é chamado de elemento excitador, pois é onde ocorre a interface entre as ondas eletromagnéticas e o cabo coaxial que alimenta a antena. Caso a antena esteja atuando como transmissor, é no dipolo que a energia é introduzida no sistema, e caso esteja atuando como receptor, é onde a energia é retirada e direcionada ao cabo coaxial. A modelagem e projeto de antenas Yagi-Uda é bem conhecida na literatura (STUTZMAN, 1981; THIELE, 1969).

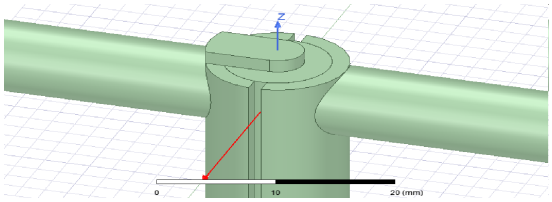
Ao posicionar um elemento passivo com comprimento maior que o comprimento do dipolo, este elemento é excitado pelas ondas eletromagnéticas do dipolo e interfere com a distribuição da energia. O resultado é um direcionamento da energia emitida para a direção contrária à posição deste elemento, aumentando a diretividade do conjunto. Elementos assim são chamados refletores. Já um elemento de comprimento menor que o comprimento do dipolo direciona a energia emitida em sua direção, também aumentando a diretividade do conjunto. Estes elementos são chamados diretores. Elementos refletores e diretores são chamados de elementos parasitas, e são elementos puramente passivos no sistema. (STUTZMAN, 1981)

Ao acrescentar elementos maiores de um lado do dipolo e elementos menores do lado contrário, consegue-se um aumento significativo da diretividade da antena. A adição de dois ou mais elementos refletores não melhora a performance da antena de forma significativa, enquanto a adição de elementos diretores deixa de ser efetiva após 9 elementos. Há ainda a necessidade de acrescentar uma estrutura de suporte, com material não condutor e de preferência que não interaja fortemente com a radiação da antena, que possa sustentar os elementos parasitas do arranjo mantendo a geometria desejada. (STUTZMAN, 1981)

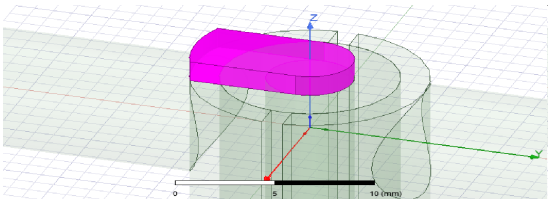
O comprimento de cada elemento das antenas Yagi-Uda pode ser parametrizado, e otimizações numéricas executadas para determinar individualmente o comprimento de



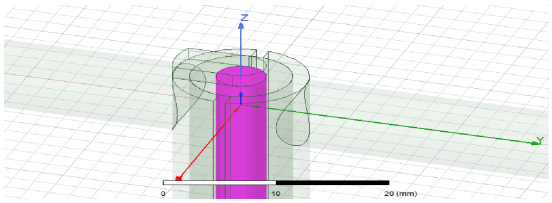
(a) Geometria da antena dipolo



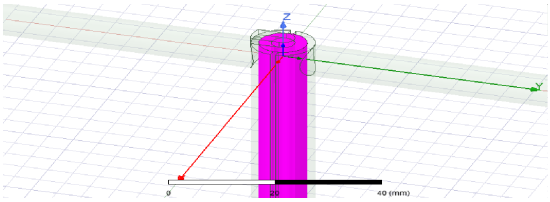
(b) Detalhe da conexão do *Balun* com as hastes do dipolo



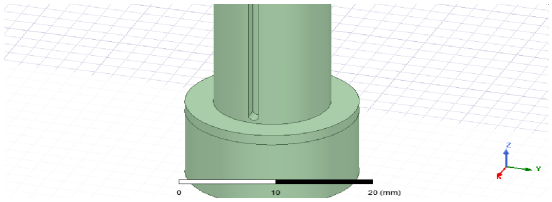
(c) Conector da antena (*PEC*)



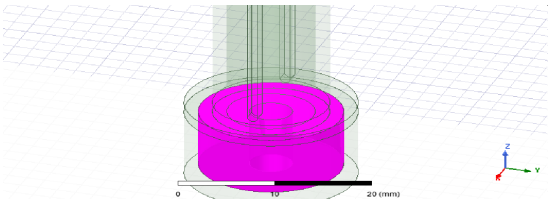
(d) Detalhe do condutor interno (*PEC*)



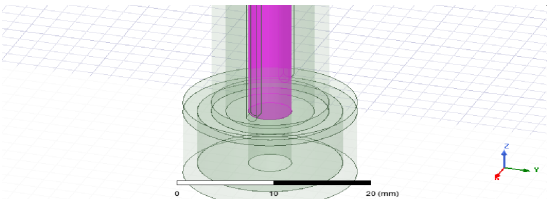
(e) Preenchimento de teflon do *Balun*



(f) Geometria da conexão do cabo coaxial



(g) Preenchimento de teflon do cabo coaxial



(h) Detalhe do condutor interno (*PEC*)

Figura 17 – Detalhes da geometria da antena dipolo criada

cada elemento utilizando a minimização do parâmetro S e o aumento da diretividade da antena como variáveis otimizadas. A literatura (CHEN, 1975; VIEZBIKE, 1968) fornece os comprimentos tabelados de cada elemento de cada antena Yagi com números variados de elementos. Estes comprimentos garantem que a otimização numérica esteja próxima de um ponto ótimo, fazendo com que a otimização numérica seja menos custosa. Em alguns casos, os comprimentos fornecidos já garantem uma performance tão boa que a otimização numérica não se faz necessária. A aplicação de antenas Yagi-Uda não se limita a antenas grandes, Dregely demonstra (DREGELY et al., 2011) a aplicação dos mesmo conceitos de antenas Yagi-Uda em frequências óticas, com métodos de fabricação de placas de circuito eletrônicas.

1.6 Arranjos de Antenas

Um arranjo de antenas, como trabalhado em Rondineau et al. (2006) e Ingram et al. (2005) é um conjunto de antenas posicionadas estrategicamente para trabalhar em

conjunto, modificando as propriedades eletromagnéticas do arranjo em relação à cada um de seus elementos. Arranjos de antenas permitem um sistema de comunicação de menor custo, uma vez que as antenas mais simples, ao atuarem em conjunto, podem formar padrões de radiação que seriam muito difíceis de serem produzidos uma única antena. As aplicações de arranjos de antenas são as mais diversas como é mostrado em Dregely et al. (2011), Santana (2021), e Sousa (2022), sendo inclusive utilizadas dentro de circuitos impressos para confeccionar antenas impressas com diretividades configuráveis. (CHEN, 2005; BHATTACHARYYA, 2006; MAILLOUX, 2017)

As ondas eletromagnéticas emitidas pelas diferentes antenas do arranjo tem interefência construtiva ou destrutiva, assim como os dipolos infinitesimais que formam antenas reais. Desta forma, é possível extender o uso do fator de arranjo nos campos eletromagnéticos das diferentes antenas do arranjo. (HARIHARAN, 2010)

Com o campo elétrico de cada antena conhecido em seu sistema local, é necessário aplicar a mudança de sistema referencial para o global utilizando matrizes de rotação e o fator de arranjo da antena. O campo elétrico resultante do arranjo $\vec{E}_a$ é a soma das contribuições de cada antena de acordo com a Equação 1.130 em notação matricial, onde $\overline{\overline{R}}_S^{S_i}$ é a matriz de rotação do sistema de referência local (S_i) de uma antena i para o sistema de referência global S do arranjo, $\vec{k}$ é o vetor de onda, $\vec{p}_i$ é o vetor posição conhecido da antena i no referencial global do arranjo, representado por 1.132 e θ_i e ϕ_i são os ângulos de apontamento no referencial local da antena. Para uma dada direção, os ângulos de apontamento no sistema local podem diferir dos ângulos de apontamento no sistema global devido à rotação entre os sistemas, mesmo na região de campo distante. Assim, os ângulos θ' e ϕ' no referencial local devem ser determinados para cada direção (ϕ, θ) no referencial global. Isto é feito rotacionando o vetor $\vec{k}$ do referencial global para o referencial local, utilizando a transposta da matriz de rotação $\left(\overline{\overline{R}}_S^{S_i}\right)^t = \overline{\overline{R}}_{S_i}^{S}$, e aplicando as relações das Equações 1.27 e 1.28.

$$\vec{E}_a(\theta, \phi) = \sum_{i=1}^{N} \mathrm{e}^{-j\vec{k}(\theta,\phi)\cdot\vec{p}_i} \overline{\overline{R}}_S^{S_i} \vec{E}_i(\theta_i, \phi_i) \tag{1.130}$$

Em um arranjo de antenas, cada antena é transladada para uma posição diferente. A origem do sistema de coordenadas do arranjo, chamado de sistema referencial global S, pode ser um ponto arbitrário qualquer, coincidindo ou não com o centro de fase de uma antena específica ou do arranjo. Vale lembrar que, trabalhando com os campos distantes de radiação, a distância da origem do sistema de coordenadas ao centro de fase de cada antena e do arranjo como um todo se torna negligenciavel. Em aplicações reais, a distância entre as antenas transmissoras e receptoras pode ser tão grande que mesmo as translações das antenas de vários comprimento de onda se tornam insignificantes, assim, validando as simplificações de campo distante utilizadas. A translação de uma antena no sistema

referencial global é representado pelo vetor de translação ou deslocamento $\Delta\vec{P}$ definido pela Equação 1.76. Um sistema referencial transladado da antena, chamado de sistema referencial local S_i, é de tal forma que um ponto $\vec{p}$ representado no sistema referencial global pode ser representado no sistema referencial local como $\vec{p}_i = \vec{p} - \Delta\vec{P}$. Os ângulos de apontamento ϕ_i e θ_i no sistema referencial local, ou seja, as coordenadas esféricas do ponto $\vec{p}_i$, podem ser determinados respectivamente com as Equações 1.82 e 1.83.

A translação da antena introduz ainda uma defasagem nas ondas eletromagnéticas emitidas por ela. Esta é chamada de defasagem geométrica δ e varia com os ângulos de apontamento. A translação da antena normalmente não é pequena em relação ao comprimento de onda da radiação, assim, é necessário avaliar o efeito da defasagem geométrica pois esta resulta em interferências construtivas e destrutivas entre as antenas. A defasagem geométria se dá pela Equação 1.133, em função do vetor de onda $\vec{k}$, dado pela Equação 1.58, e do vetor de translação.

$$\delta = k_0 x_i \sin(\theta)\cos(\phi) + k_0 y_i \sin(\theta)\sin(\phi) + k_0 z_i \cos(\theta) \tag{1.131}$$

Definindo $\vec{p}_i$ como o vetor de translação da antena i:

$$\vec{p}_i = x_i\hat{x} + y_i\hat{y} + z_i\hat{z} \tag{1.132}$$

$$\delta = \vec{k} \cdot \vec{p}_i \tag{1.133}$$

É então introduzido um fator de arranjo $F_{ai}(\theta, \phi)$ de uma antena i, em função da defasagem geométrica desta antena. F_{ai} é um fator geométrico dado pela Equação 1.134 que aplica sobre o campo elétrico o efeito da translação da antena. Este fator introduz o efeito da defasagem geométrica no campo eletromagnético que viaja na dada direção (ϕ, θ). O fator de arranjo pode ainda levar em consideração a alimentação da antena, que pode ter fase e amplitude diferentes para cada antena do arranjo, sendo multiplicado por um fator I_i que representa sua respectiva alimentação. Neste caso, I_i é um fasor onde o argumento representa a fase da alimentação em relação à uma referência e a magnitude de I_i é a magnitude da corrente de alimentação. A magnitude da parte do fator de arranjo é controlada pela corrente de alimentação, pois a translação da antena não interfere na magnitude da radiação emitida, entretanto, a soma dos campos elétricos de diversas antenas multiplicados por seus respectivos fatores de arranjos modifica a magnitude do campo elétrico resultante, devido às diferentes fases do campo elétrico de cada antena que resultam em interferências construtivas e destrutivas.

As antenas do arranjo podem ainda ser rotacionadas, o que se traduz em uma rotação do campo eletromagnético emitido pela antena. Para contabilizar este efeito, o campo elétrico da antena passa por 3 rotações, primeiramente rotacionando um ângulo β sobre seu eixo $\hat{z}$ (*azimuth*), depois rotacionando um ângulo α sobre seu eixo $\hat{y}$ (*elevation*),

e por fim um ângulo γ sobre seu eixo $\hat{x}$ (*roll*). As rotações e convenções utilizadas neste trabalho são mais discorridas na Seção 5.1. Estas 3 rotações são aplicadas na forma matricial, com uma única matriz de mudança de coordenadas $\overline{\overline{R}}_S^{Si}$ do sistema local da antena S_i para o sistema global S. Depois de aplicados os efeitos da rotação da antena, o campo elétrico é multiplicado pelo fator de arranjo, assim aplicando os efeitos da translação da antena, e o campo elétrico resultante de uma antena i em campo distante corrigido pela mudança de sistema de referência $\vec{E}_i$ se dá então pela Equação 1.136. Nesta equação, $\vec{E}_{Si}$ é o campo elétrico emitido por uma antena em seu sistema referencial local, ou seja, sem quaisquer rotações ou translações, e I_i é a corrente de excitação desta antena. Ainda na Equação 1.136, (θ_i, ϕ_i) representa a direção de apontamento do ponto de observação do campo elétrico $\vec{p}_i$ no sistema referencial local da antena, enquanto (θ, ϕ) representa a direção de apontamento do mesmo ponto no sistema referencial global. Naturalmente estas duas coordenadas esféricas são muito diferentes dependendo da orientação da antena, e é preciso cuidado para determinar o campo elétrico emitido pela antena na direção do ponto de observação. A Equação 1.136 é interpretada com todos os vetores na forma matricial, assim, a aplicação da rotação se torna apenas uma multiplicação da matriz de rotação por um vetor coluna e o termo $I_i \mathrm{e}^{-j\vec{k}(\theta,\phi)\cdot\vec{p}_i}$ é um escalar complexo que multiplica o vetor elemento por elemento.

$$F_{ai}(\theta, \phi) = I_i \mathrm{e}^{-j\vec{k}(\theta,\phi)\cdot\vec{p}_i} \tag{1.134}$$

$$\vec{E}_i(\theta, \phi) = F_{ai}(\theta, \phi)\overline{\overline{R}}_S^{S_i}\vec{E}_{Si}(\theta_i, \phi_i) \tag{1.135}$$

$$\vec{E}_i(\theta, \phi) = I_i \mathrm{e}^{-j\vec{k}(\theta,\phi)\cdot\vec{p}_i}\overline{\overline{R}}_S^{S_i}\vec{E}_{Si}(\theta_i, \phi_i) \tag{1.136}$$

2 Metodologia

2.1 Ferramentas Computacionais

Uma sequência de tarefas foi executada para projetar uma família de antenas Yagi-Uda para comunicação com satélites em órbita terrestre baixa (*Low Earth Orbit - LEO*). A primeira antena desenvolvida foi uma dipolo.

Utilizando dimensões iniciais arbitrárias, foi projetada uma antena dipolo com *Balun* simples para operação a 433 MHz no *ANSYS Electronics HFSS*. Todas as dimensões do projeto foram parametrizadas. A antena da simulação é alimentada por uma *wave port* posicionada em um pedaço de cabo coaxial, que se conecta no *Balun*. As dimensões do *Balun* foram parametrizadas e otimizadas para minimizar o parâmetro S da *wave port*.

Quando o parâmetro S da antena se mostrou satisfatório, foram introduzidos de 1 a 3 elementos passivos à simulação, formando assim 3 antenas Yagi-Uda de diretividades diferentes. Cada antena Yagi-Uda foi simulada, ainda no HFSS, para determinar suas diretividades e ângulos de abertura. Por fim, foi introduzido uma estrutura de PVC para suportar os elementos parasitas das antenas, e mais uma simulação foi executada para determinar os parâmetro S, diretividade e ângulo de abertura a 3 dB de cada antena.

2.1.1 Dipolo

2.1.1.1 Primeiros passos

Para projetar as antenas, o desenho geral do dipolo e do *Balun* foram criados. As dimensões iniciais das geometrias do dipolo e do *Balun* são definidos a partir do comprimento de onda no meio λ_0 da frequência desejada. Para a frequência de 433 MHz, o λ_0 é aproximadamente 69 cm, assim, o dipolo tem comprimento de aproximadamente 36 cm. As dimensões do *Balun* são calculadas a partir do λ_0, com comprimento também de 30 cm e largura do rasgo de 2 mm. Um pedaço de cabo-coaxial de teflon, com impedância de 50 Ω, foi introduzido na base do *Balun*, com raio interno de 2.5 mm escolhido arbitrariamente. O raio do *Balun* foi então definido como 4.5 mm. Os valores iniciais calculados garantem que a antena esteja próxima do ponto ótimo de operação, assim necessitando de menos simulações de otimização. As variáveis otimizadas foram o comprimento e largura do rasgo do *Balun*, e o otimizador utilizado no HFSS foi *Adaptive Single-Objective* (Gradiente).

2.1.1.2 Padronização

É esperado que a performance da antena seja sensível a certas dimensões da antena, enquanto outras não interfiram tanto. Para economizar em tempo de processamento, algumas dimensões da antena não são optimizadas enquanto seus valores são escolhidos levando em consideração o processo de fabricação almejado. As dimensões a serem optimizadas são parametrizadas a criação da geometria no HFSS. As dimensões que são constantes podem interferir no processo de fabricação. Por exemplo, a espessura das paredes do Balun e o diâmetro do dipolo. Mesmo não sendo dimensões que participam das optimizações, a definição bem pensada destas dimensões pode facilitar o processo de fabricação. Assim, valores padrões foram definidos para dimensões de espessura e diâmetro, sendo, respectivamente, 0.5 mm e 4 mm. Estes valores foram escolhidos com base em disponibilidade em lojas de materiais de construção, dado que são valores regularmente encontrados em espessura e diâmetro de tubos e chapas.

2.1.1.3 Simulações e optimizações no HFSS

Com a geometria criada e as dimensões a serem optimizadas definidas, foram iniciadas as simulações numéricas. Simulações com os valores iniciais das dimensões proporcionaram uma ideia da performance da antena. Então, foi configurada a otimização das variáveis para minimização do parâmetro S da antena. O objetivo foi chegar a parâmetros S abaixo de -15 dB. As tabelas 1 e 2 resumem as dimensões parametrizadas da antena dipolo após as otimizações.

Variável	Valor final	Unidade
w_Rasgo	1.00	mm
h_Balun	185	mm
h_Rasgo	183	mm

Tabela 1 – Valores finais de variáveis otimizadas

Variável	Valor inicial	Unidade
$thickness$	0.50	mm
h_Dipolo	34.6	cm
f	433	MHz
c_0	300	10^6 m/s
$lambda$	69.2	cm
$R\_InnerCore$	2.00	mm
$R\_OutterCore$	7.54	mm
R_Balun	4.60	mm
E_r	2.10	-
M_r	1.00	-
$CoaxialRatio$	3.35	-

Tabela 2 – Variáveis que não participam da otimização

2.1.2 Antenas Yagi-Uda

2.1.2.1 Elementos parasitas

A antena Dipolo é a base para o desenvolvimento de antenas Yagi-Uda. Elementos são adicionados ao redor do dipolo para modificar as propriedades do conjunto. As dimensões dos elementos parasitas (elementos diretores e refletores) são prontamente disponibilizados em tabelas em função do comprimento de onda, como no capítulo 5.4 de Stutzman (1981). As simulações preliminares resultaram nas diretividades desejadas para cada número de elementos, não sendo necessária nenhuma otimização nas posições e comprimentos dos elementos. Entretanto, houve aumento nos parâmetros S das antenas, e uma nova otimização do balun se faz necessária para refiná-los novamente.

2.1.2.2 Elementos de suporte

Posteriormente, se faz necessário projetar elementos de suporte, que unem a antena dipolo aos elementos parasitas, mantendo a geometria desejada e provendo rigidez física. O material destes elementos estruturais não deve interagir fortemente com as ondas eletromagnéticas, para não modificar a performance da antena já otimizada. Tubos de PVC foram utilizados como base para o projeto dos elementos estruturais, pois PVC é um plástico que não interage fortemente com radiação eletromagnética da antena e é um material acessível.

Extrusões foram feitas na geometria do tubo para encaixe da antena dipolo e dos elementos parasitas. O diâmetro dos tubos foi escolhido com base na disponibilidade em lojas de materiais de construção. Por fim, simulações foram executadas para extrair o parâmetro S, a diretividade e ângulo de abertura a meia potência. Elementos pequenos, como parafusos e ruelas, podem ser utilizados para prender os elementos da antena. O efeito destes elementos foi desconsiderado neste estágio do trabalho, pois é esperado que estes não afetem a performance da antena de forma significante e agregariam grande complexidade às simulações. Simulações futuras devem incluir estes elementos para confirmar que estes não afetem a performance das antenas.

2.1.3 Arranjo de antenas

Com a família de antenas definidas, cada antena separadamente funcional e características satisfatórias, o arranjo de antenas será desenvolvido. Inicalmente, um algoritmo foi criado para calcular o campo elétrico resultante de um arranjo de antenas qualquer utilizando o fator de arranjo e o campo elétrico conhecido de cada antena, conforme o desenvolvimento matemático demonstrado anteriormente, com as posições e orientações de cada antena parametrizadas. Este software será utilizado, em conjunto com uma função de custo a ser definida, em algoritmos de minimização de função de custo já fornecidos

no Octave para optimizar as posições e orientações de cada antena no arranjo, de forma que o campo elétrico resultante seja o mais próximo de um campo elétrico qualquer, que será definido de forma ideal para a missão em questão. Tendo em mente um campo elétrico ideal para a aplicação atual, o algoritmo será utilizado para construir o arranjo final. O campo elétrico produzido por cada antena desenvolvida no presente trabalho será exportado e usado no arranjo de antenas da missão.

2.1.4 Disponibilidade de materiais

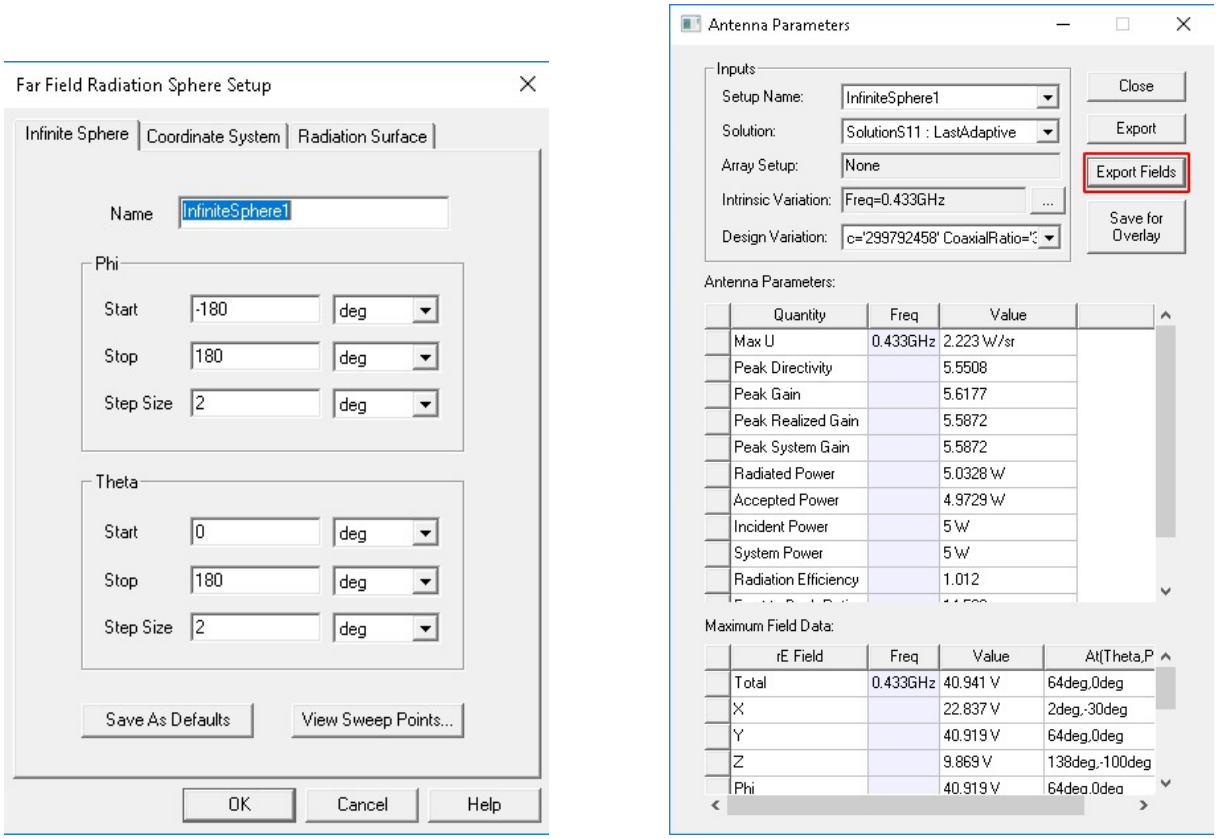
Com as dimensões devidamente optimizadas, foi efetuada uma busca em lojas de materiais para seleção de tubos e chapas com dimensões próximas dos resultados das simulações e custo razoável. As dimensões dos componentes selecionados foram introduzidas no modelo das antenas desenvolvidas, e então uma simulação foi executada para aferir que as antenas continuam com performance aceitável.

2.1.5 Exportar os campos elétricos

Para cada simulação de antena ou arranjo de antenas realizada, foi configurada uma amostragem dos campos elétricos sobre campo distantes no *HFSS*. Esta opção é configurada na aba *Radiation*, visualizada na Figura 18a, onde é criada uma condição de campos distantes chamada *Infinite Sphere*. Como o nome indica, os pontos de amostragem são modelados sobre uma esfera de raio "infinito", o que significa que estão posicionados na região de campos distantes. Após a simulação, os campos elétricos sobre os pontos de amostragem podem ser computados e exportados em arquivos .ffd ou .csv utilizando a aba *Compute Antenna Parameters*. Os campos elétricos foram exportados utilizando a opção "*Radiation -> Infinite Sphere -> Compute Antenna Parameters -> Export Fields*", conforme visualizado na Figura 18b. Os campos elétricos exportados em formato .csv foram então utilizados para modelar as antenas nos códigos desenvolvidos, para validar seu funcionamento e então projetar novos arranjos utilizando algoritmos de otimizações numéricas.

2.2 Algoritmos desenvolvidos

Foram desenvolvidos códigos em linguagens de programação *Octave* e *Python* implementando as equações e modelos matemáticos discorridos neste trabalho. Primeiramente, as equações e modelos foram desenvolvidos em *Octave*. Foram criados arranjos no *HFSS* com 1 a 5 antenas Yagi-Uda de 4 elementos desenvolvidos na primeira parte deste trabalho. O campo elétrico resultante dos arranjos foi simulado no *HFSS* e exportado. Então, os mesmos arranjos foram modelados nos *softwares* em *Octave*, e seus campos elétricos foram simulados. Isto serviu como validação dos algoritmos implementados e dos



(a) Aba de configuração da amostragem do campo elétrico da antena

(b) Aba para exportar os campos elétricos da simulação

Figura 18 – Capturas de tela da configuração da simulação

modelos matemáticos mostrados neste trabalho. Posteriormente, os mesmos algoritmos foram implementados em *Python*, desenvolvendo uma interface gráfica para facilitar o projeto dos arranjos e o uso do *software*. Por fim, foram implementados algoritmos de otimização numérica em *Python* para modelar os campos elétricos dos arranjos conforme o desejado. Com estes programas, otimizações foram executadas e seus resultados foram dispostos.

3 Antena Dipolo

A antena dipolo é parte essencial do projeto de uma antena yagi, e merece um capítulo dedicado ao seu desenvolvimento.

A geometria da antena Dipolo desenvolvida é ilustrada na Figura 19. Suas propriedades são mostradas na Figura 20. Na Figura 20a, é visualizada a diretividade padrão de uma antena dipolo, com simetria radial e pouca energia sendo irradiada na direção paralela ao dipolo. A Figura 20b ilustra o parâmetros S da antena com frequências de 400 MHz até 450 MHz, com o menor valor ao redor de 433 MHz, que é a frequência de operação para qual a antena foi dimensionada. As Figuras 20d e 20c mostram a diretividade da antena em relação aos ângulos ϕ e θ, com uma diretividade relativamente constante em θ e variando principalmente em ϕ. Os resultados da antena dipolo desenvolvida são condizentes com a literatura. A diretividade da antena não é uma propriedade a ser optimizada. É importante garantir que os parâmetros S da antena seja o menor possível, pois assim há reflexão mínima da energia, maximizando assim a eficiência da antena. É interessante notar que o efeito do Balun na diretividade é imperceptível. O diagrama de diretividade da Figura 20a tem escala em dB, variando de -4 a 10 dB para uma comparação visual com as demais antenas desenvolvidas.

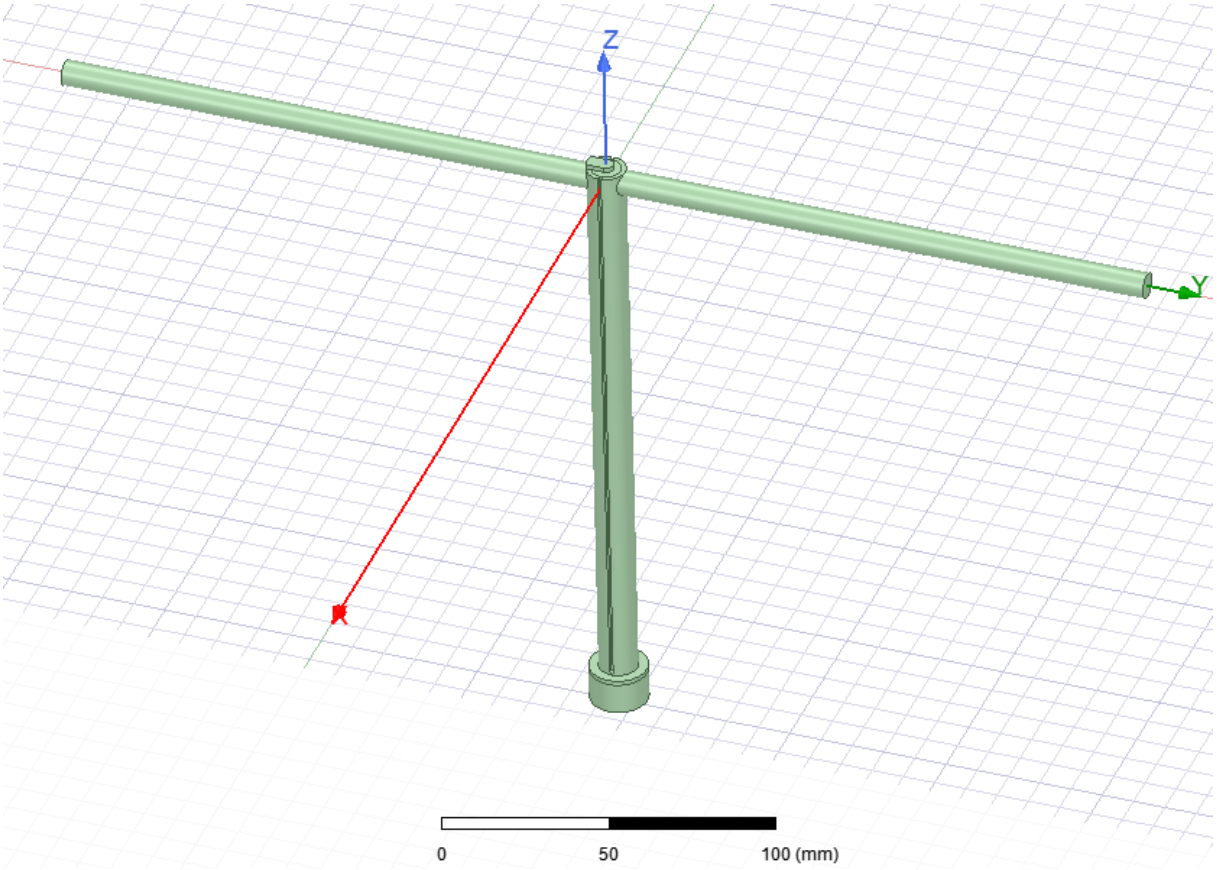
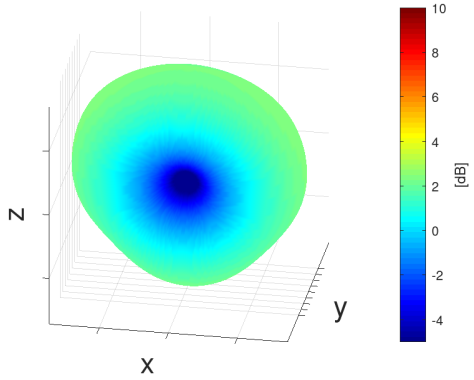
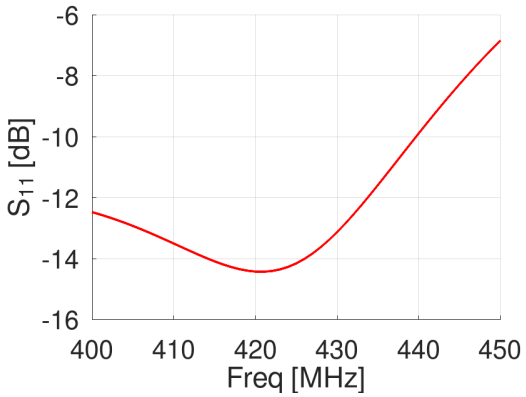


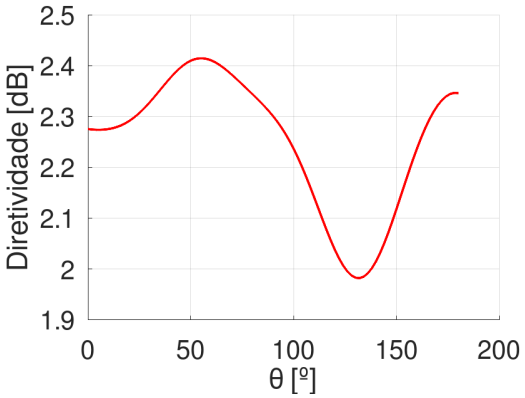
Figura 19 – Geometria da antena Dipolo desenvolvida



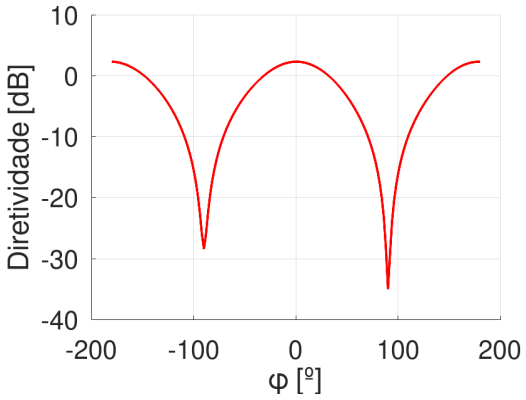
(a) Diretividade tridimensional



(b) Parâmetros S para frequências de 400 a 450 MHz



(c) Diretividade em θ para $\phi = 0°$



(d) Diretividade em ϕ para $\theta = 90°$

Figura 20 – Propriedades da antena Dipolo

Parâmetro S_{11} [dB]	-12.2
Diretividade total [dB]	2.3
A_θ [°]	42.0
A_ϕ [°]	36.0

Tabela 3 – Propriedades da antena Dipolo projetada

Um resumo das propriedades da antena Dipolo são encontrados na Tabela 3. Como esperado, a maior parte da energia emitida pela antena está no plano XZ. Embora a tabela apresente um ângulo de abertura de 3 dB em θ, a variação de diretividade em θ sobre o plano YZ é negligenciável. A diretividade total da antena é de 2.30 dB pois, enquanto uma antena isotrópica distribui energia em todas as direções igualmente, uma antena dipolo redistribui a energia que seria transmitida na direção Y para as direções X e Z.

4 Antenas Yagi-Uda

A Figura 21 ilustra a geometria da antena Yagi-Uda com 2 elementos e a Figura 22 contém os gráficos das propriedades desta antena. Percebe-se um claro aumento da diretividade da antena com a adição do elemento refletor, com diretividade máxima de 6.3 dB para $\phi = 0°$ e $\theta = 90°$ (direção X). Observa-se ainda um aumento dos parâmetros S quando comparado com a antena dipolo. Isto é esperado, pois a adição de elementos parasitas interfere na reflexão das ondas eletromagnéticas. Esta interferência pode ser mitigada por meio de uma nova otimização do Balun da antena dipolo. As Figuras 23, 24, 25 e 26 mostram a mesma caracterização realizada nas demais antenas Yagi-Uda com 3 e 4 elementos.

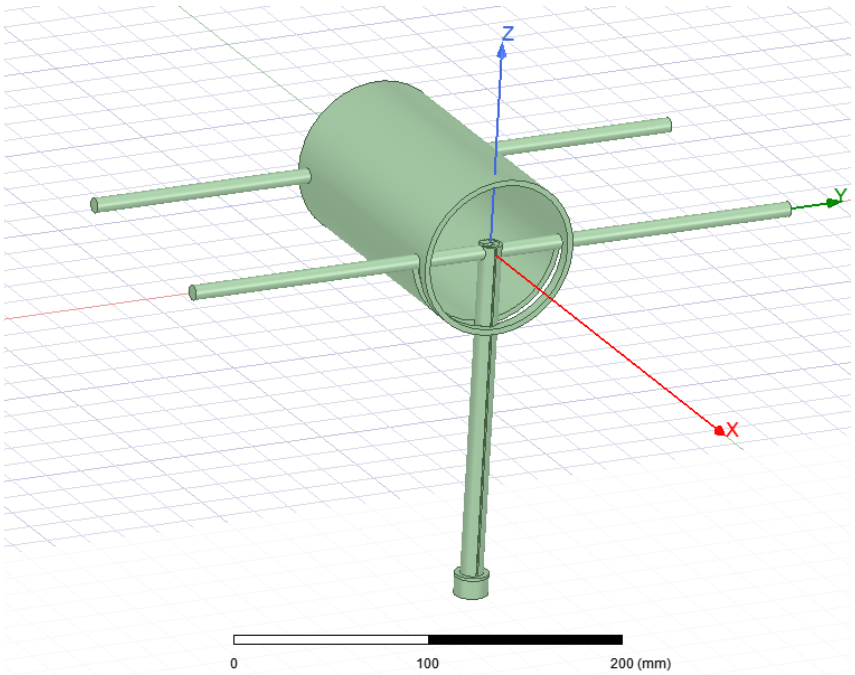
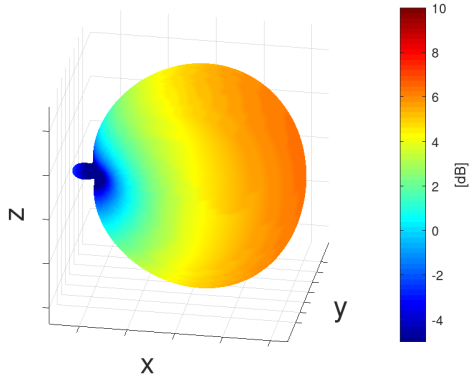
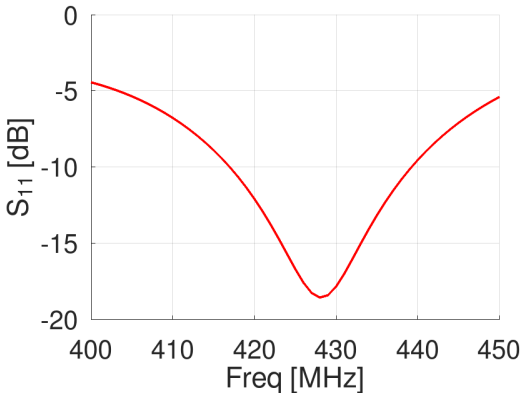


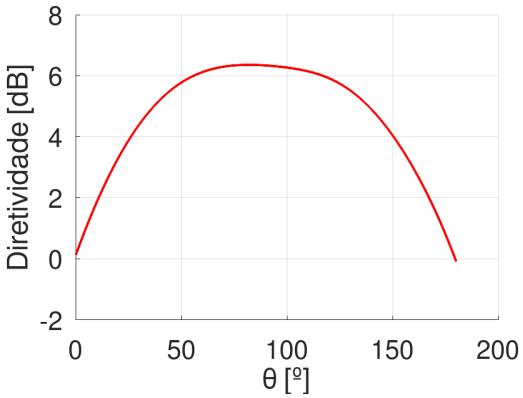
Figura 21 – Geometria da antena Yagi desenvolvida com 2 elementos



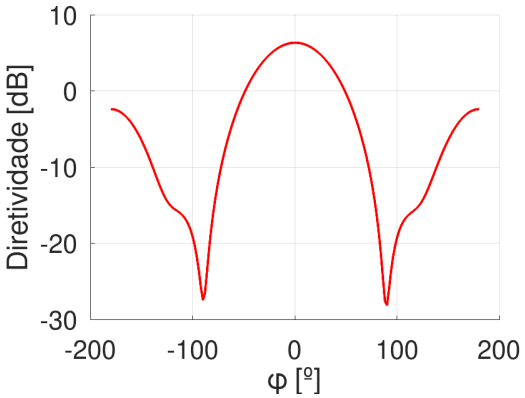
(a) Diretividade tridimensional



(b) Parâmetro S_{11} para frequências de 400 a 450 MHz



(c) Diretividade em θ para $\phi = 0°$



(d) Diretividade em ϕ para $\theta = 90°$

Figura 22 – Propriedades da antena Yagi desenvolvida com 2 elementos

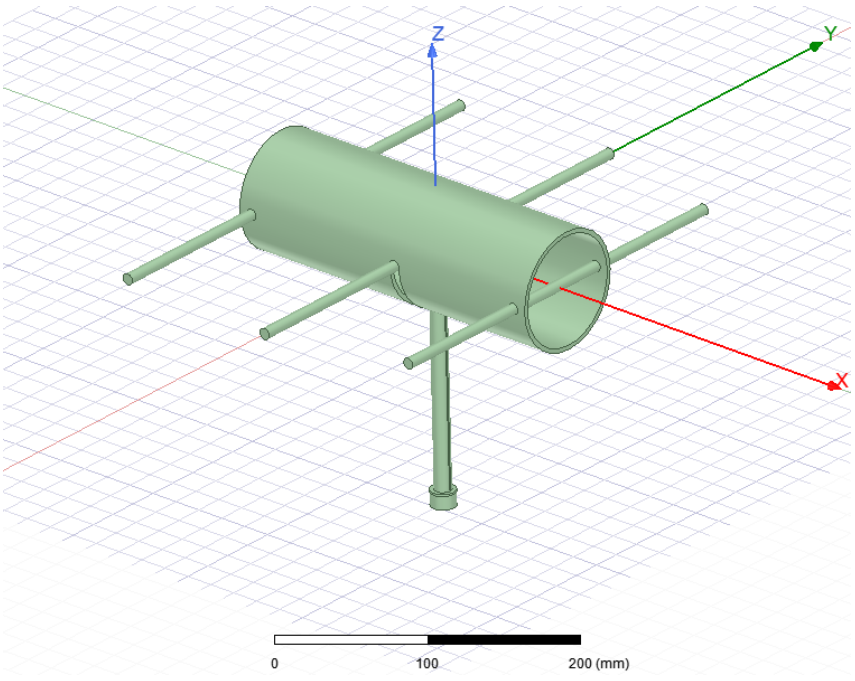
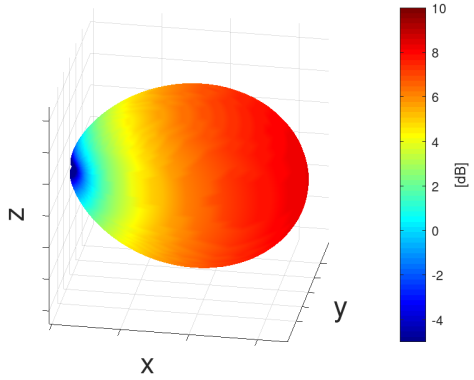
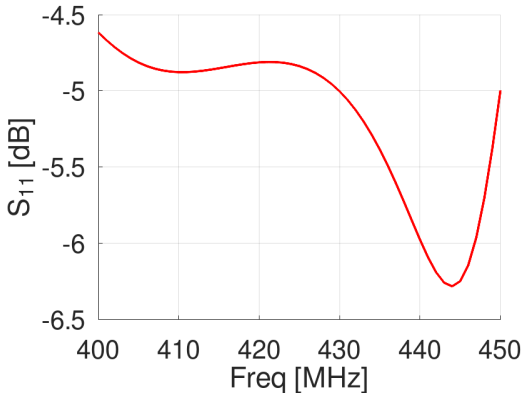


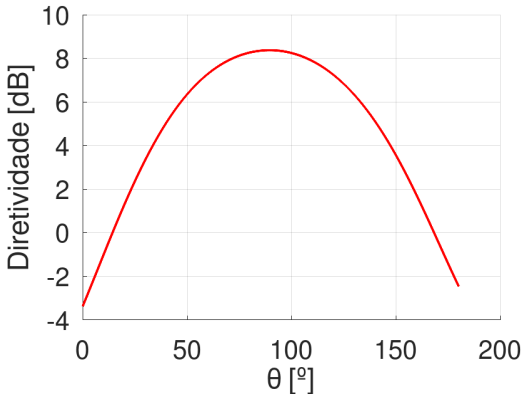
Figura 23 – Geometria da antena Yagi desenvolvida com 3 elementos



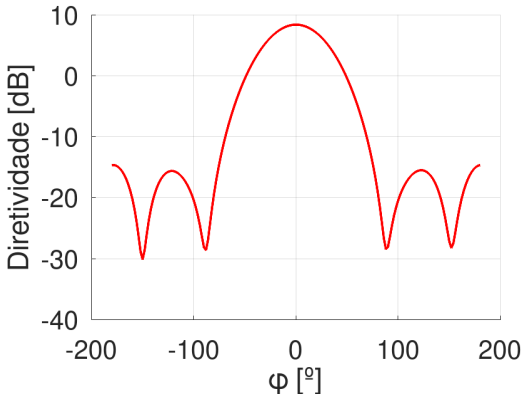
(a) Diretividade tridimensional



(b) Parâmetro S_{11} para frequências de 400 a 450 MHz



(c) Diretividade em θ para $\phi = 0°$



(d) Diretividade em ϕ para $\theta = 90°$

Figura 24 – Propriedades da antena Yagi desenvolvida com 3 elementos

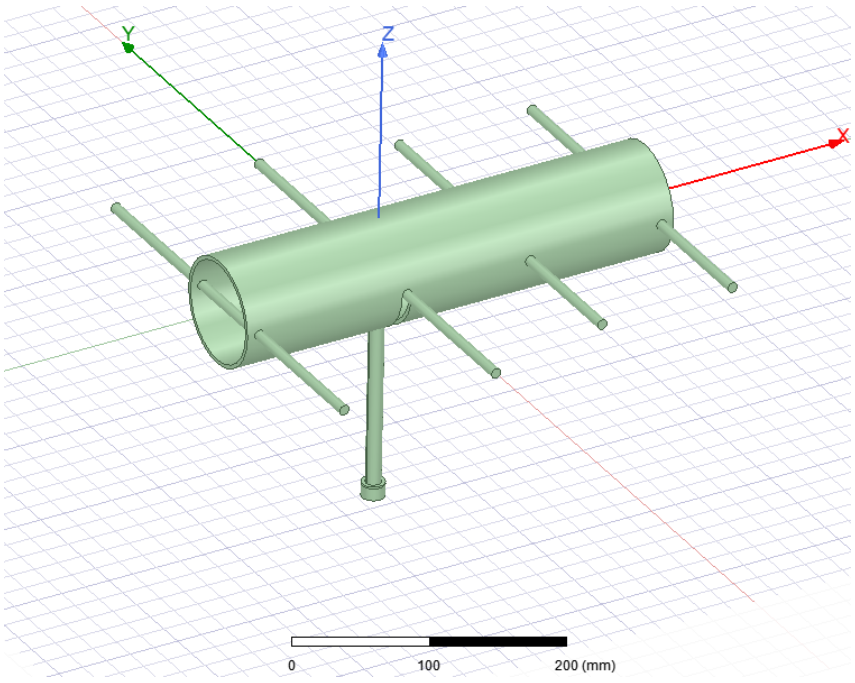
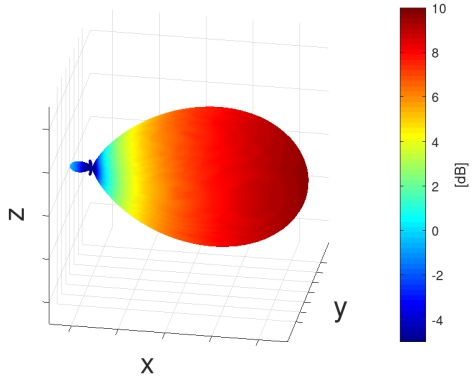
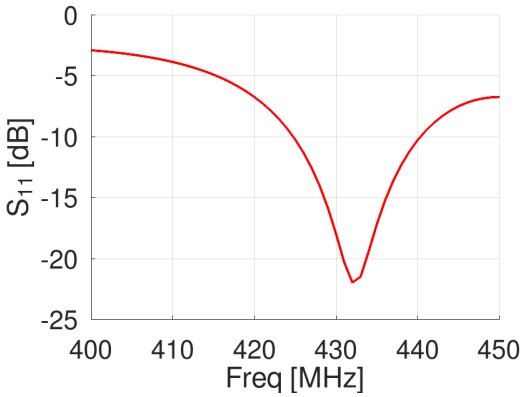


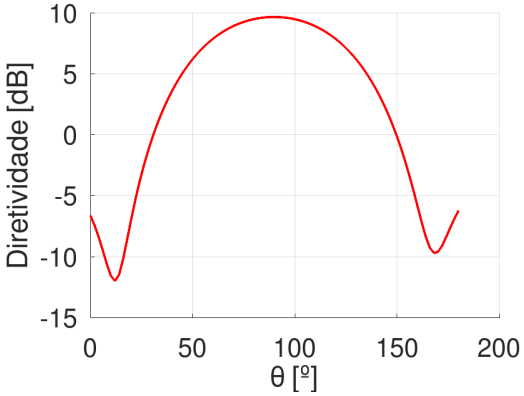
Figura 25 – Geometria da antena Yagi desenvolvida com 4 elementos



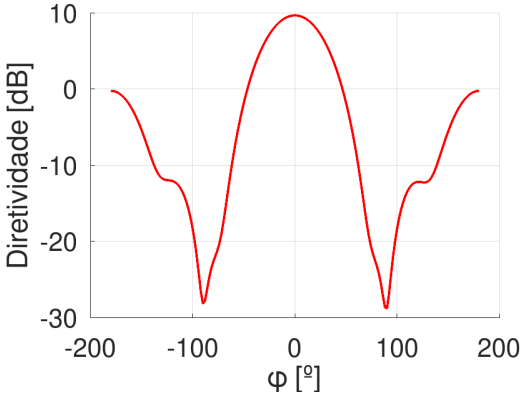
(a) Diretividade tridimensional



(b) Parâmetro S_{11} para frequências de 400 a 450 MHz



(c) Diretividade em θ para $\phi = 0°$



(d) Diretividade em ϕ para $\theta = 90°$

Figura 26 – Propriedades da antena Yagi desenvolvida com 4 elementos

Número de elementos	Parâmetro S_{11} [dB]	Diretividade total [dB]	A_θ [°]	A_ϕ [°]
2	−15.0	6.3	66.0	34.0
3	−5.2	8.4	48.0	30.0
4	−21.5	9.7	38.0	28.0

Tabela 4 – Propriedades das antenas Yagi-Uda projetadas

As propriedades das antenas Yagi-Uda estão presentes na Tabela 4. Percebe-se um aumento gradual na diretividade total da antena à medida que novos elementos são adicionados. O pico de diretividade é em 9.7 dB na antena com 4 elementos. Uma diretividade maior pode ser obtida agrupadando as antenas em arranjos. A abertura de meia potência em θ diminui com novos elementos. Isto acontece porque cada vez mais energia está sendo direcionada em uma direção privilegiada (neste caso, X), enquanto menos energia é direcionada em outras direções (Z e Y). O ângulo de meia potência em ϕ não apresenta muita variação pois a antena dipolo já não direciona muita energia na direção Y e a adição dos novos elementos não afeta tanto a diretividade na direção Y quanto na direção Z. O parâmetro S da antena com 3 elementos se mostrou mais elevado, assim levantando a necessidade de um refinamento em seu *Balun*, mas as demais antenas tem parâmetros S razoáveis, em especial a antena com 4 elementos.

5 Algoritmos desenvolvidos

Foram desenvolvidos algoritmos nas linguagens de programação *Octave* e *Python* para o cálculo do vetor de radiação resultante de arranjos de antenas arbitrários. Os campos elétricos das antenas desenvolvidas no *HFSS* podem ser importados para os *softwares* desenvolvidos, e várias visualizações e análises se tornam disponíveis. Os campos elétricos de cada antena importada são utilizados para calcular, de forma quasi-analítica, o campo elétrico resultante de um arranjo arbitrário. A análise desta forma é muito mais rápida do que uma simulação numérica dos arranjos completos no *HFSS*, o que torna os códigos desenvolvidos uma ferramenta interessante para desenvolver e otimizar arranjos de antenas, principalmente quando o número de antenas se torna elevado e a distância entre elas cresce.

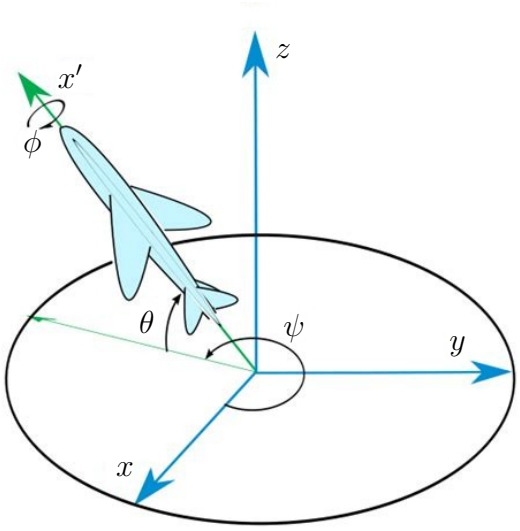
Primeiramente as equações para fator de arranjo e campo elétrico resultante foram implementadas no *Octave*. Foram utilizadas estruturas no código que representam as antenas, armazenando o campo elétrico delas para cada direção θ e ϕ, bem como sua orientação e posição no sistema de coordenadas global. Uma outra estrutura, derivada de antenas, representa um arranjo. Esta contém antenas e, quando solicitado, calcula o campo elétrico resultante do arranjo levando em consideração as posições e orientações de cada antena, bem como a posição e orientação do próprio arranjo. Desta forma, se tornou fácil projetar e editar rapidamente arranjos de antenas, simular seus campos elétricos resultantes, e modificar quaisquer parâmetros de interesse.

5.1 Orientação

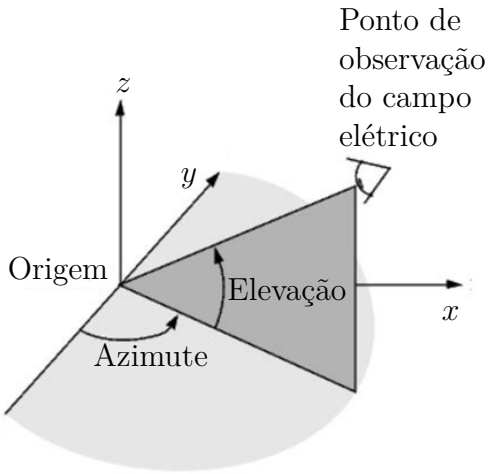
Os arranjos projetados são construídos com antenas arbitrárias e a posição, orientação, e corrente de alimentação de cada antena pode ser modificado individualmente. Existem diversas formas de definir a orientação de um sistema referencial ou de um objeto, as Figuras 27a, 27b, e 27c definem três maneiras muito utilizadas. A primeira figura mostra os ângulos de *roll*, *pitch* e *yaw*, muito utilizados para definir o sistema referencial de aeronaves. A segunda figura mostra a utilização dos ângulos de elevação e azimute, que são comumente utilizados para definição da posição de corpos celestes no céu. A terceira definição é a dos ângulos de Euler, muito utilizados em contextos de desenvolvimento de jogos. Neste trabalho, orientação de cada antena é dada por 3 ângulos chamados de azimute β, elevação α e rolagem γ. NO caso, β, α e γ são as rotações que a antena sofre, enquanto as matrizes de rotação trabalhadas levam um ponto arbitrário $\vec{P}$ do sistema referencial global para o sistema referencial da antena rotacionada. Assim, as matrizes de rotação utilizadas tem argumentos negativos, conforme mencionado na Seção 1.1.

A definição dos ângulos de orientação neste trabalho não segue os padrões normalmente utilizados nos sistemas da Figura 27, e as letras α, β e γ das rotações utilizadas não devem ser confundidos com os ângulos de Euler que utilizam as mesmas letras. São utilizados os ângulos de azimute (*azimuth*) e elevação (*elevation*) conforme os da Figura 27b, com a adição de um ângulo de rotação em torno do eixo x' (rolagem ou *roll*). Os eixos x', y' e z' são os eixos do sistema referencial local representado no sistema referencial global, ou seja, são eixos que acompanham a rotação da antena. Todas as rotações obedecem a convenção da regra da mão direita. Dados 3 ângulos de rotação, a orientação de uma antena se dá por 3 rotações respectivamente em torno dos eixos z, y' e x', então, as rotações são sempre em torno dos eixos do sistema referencial local da antena, seguindo a mesma lógica das rotações mostradas na Figura 27c. Desta forma, uma antena orientada conforme estes ângulos sofre uma rotação pelo ângulo de azimute em torno do eixo z, seguida por uma rotação pelo ângulo de elevação em torno de seu eixo y', e por fim uma rotação pelo ângulo de roll em torno de seu eixo x'. Esta definição permite modificar a orientação da antena de forma mais intuitiva. A Equação 5.1 mostra como é contruída a matriz de mudança de base do sistema referencial global para o sistema referencial local, $\overline{\overline{R}}_L^G$, da antena com orientação conforme definido, utilizando as definições de matrizes de rotação dadas em 1.22, 1.21, 1.20. As Figuras 32, 28b, e 33b mostram respectivamente o sistema referencial da antena rotacionada, a antena rotacionada no *HFSS* e a magnitude de seu campo elétrico, para uma orientação dada pelos ângulos elevação $= -30°$, azimute $= -130$ e roll $= 60°$. As Figuras 28a e 33a mostram a mesma antena e seu campo elétrico antes das rotações, para uma melhor compreensão do efeito da orientação da antena sobre seu campo elétrico. As Figuras 29 a 31 mostram o passo a passo do processo de rotação da antena.

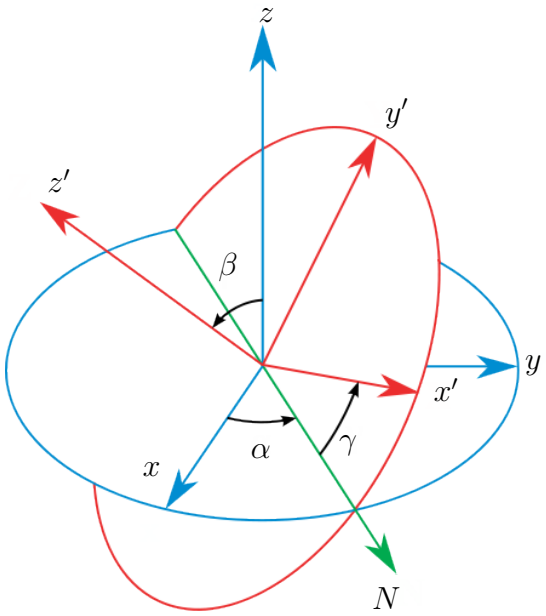
$$\overline{\overline{R}}_L^G(\beta, \alpha, \gamma) = \overline{\overline{R}}_x(-\gamma)\overline{\overline{R}}_y(-\alpha)\overline{\overline{R}}_z(-\beta) \tag{5.1}$$



(a) Ângulos de *yaw*, *pitch* e *roll*

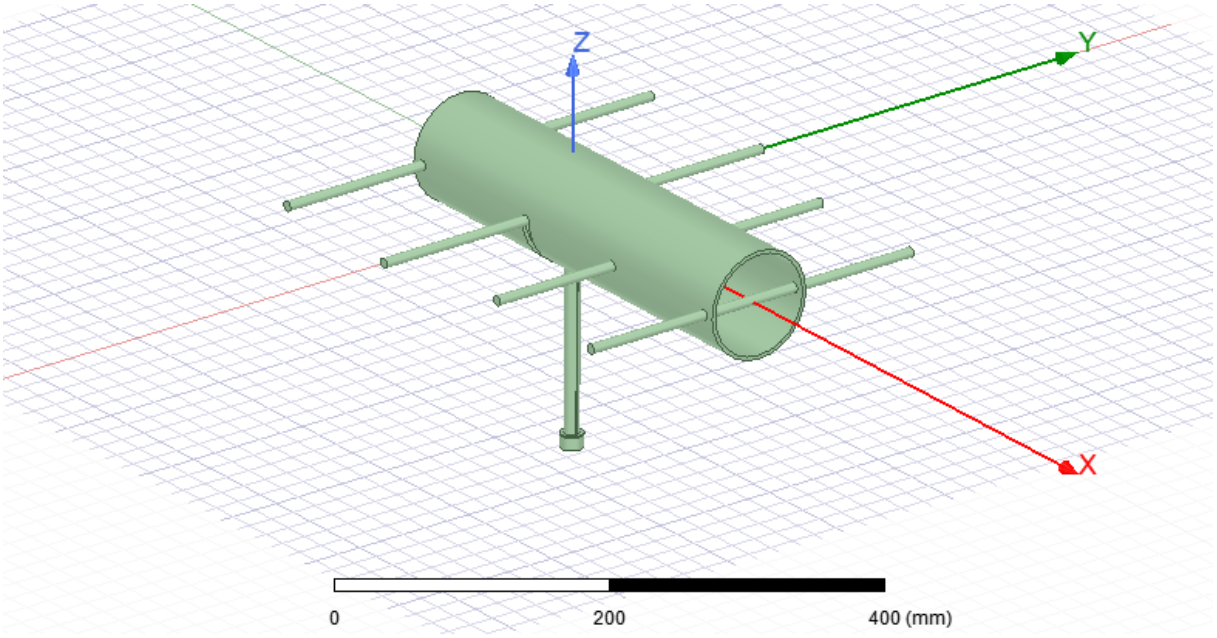


(b) Ângulos de elevação e azimute

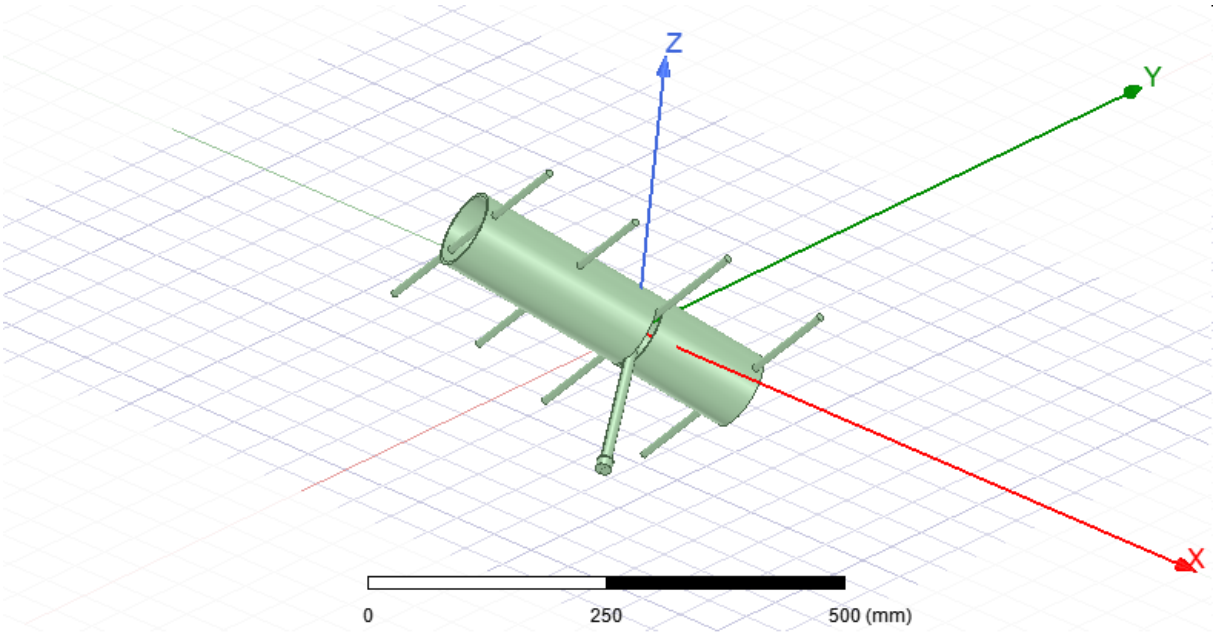


(c) Ângulos de Euler

Figura 27 – Sistemas de orientação



(a) Antena antes da rotação



(b) Antena rotacionada

Figura 28 – Rotação da antena

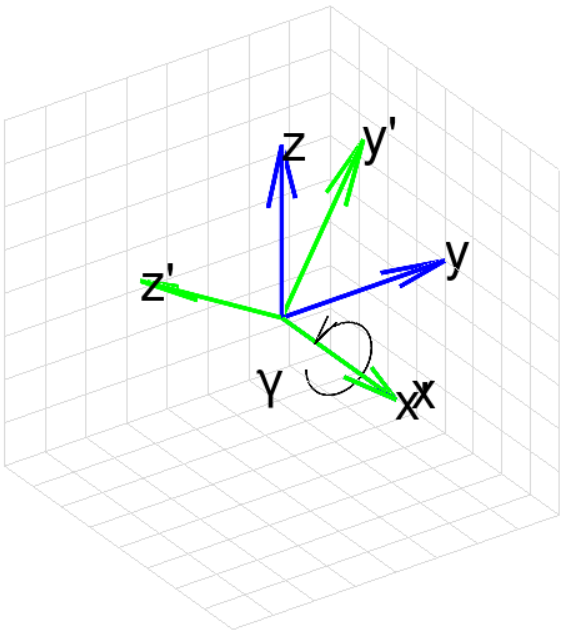


Figura 29 – Rotação em torno do eixo x (rolagem $\gamma = 60°$)

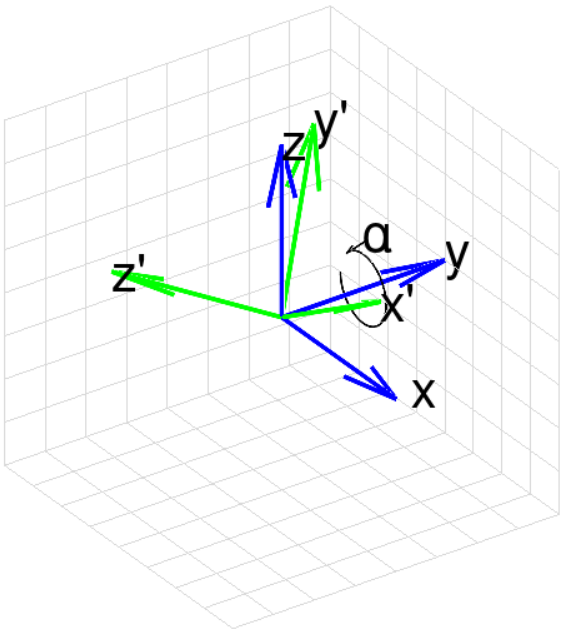


Figura 30 – Rotação em torno do eixo y (elevação $\alpha = -30°$)

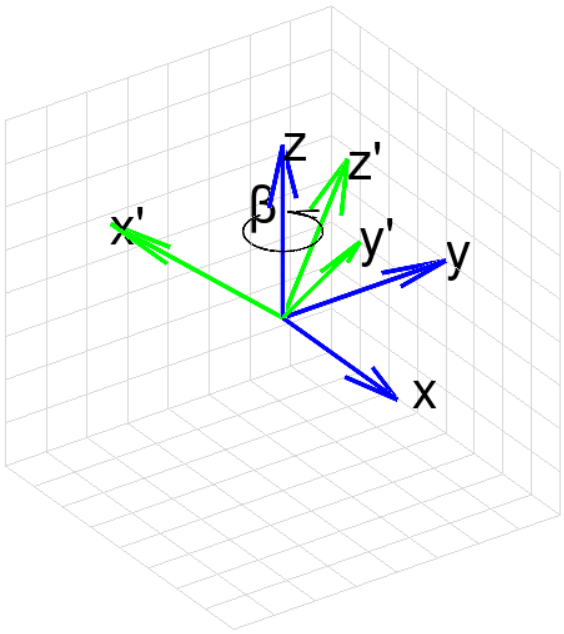


Figura 31 – Rotação em torno do eixo z (azimute $\beta = -130°$)

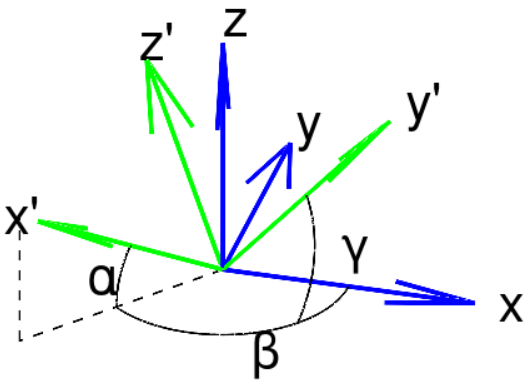
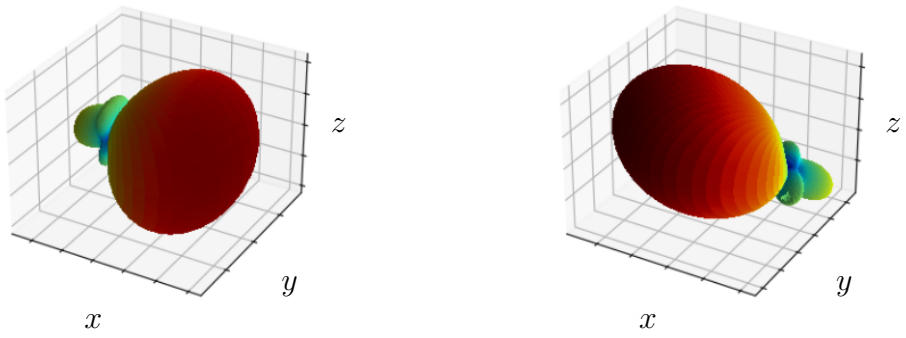


Figura 32 – Sistema referencial da antena rotacionada (verde) e sistema referencial global (azul)



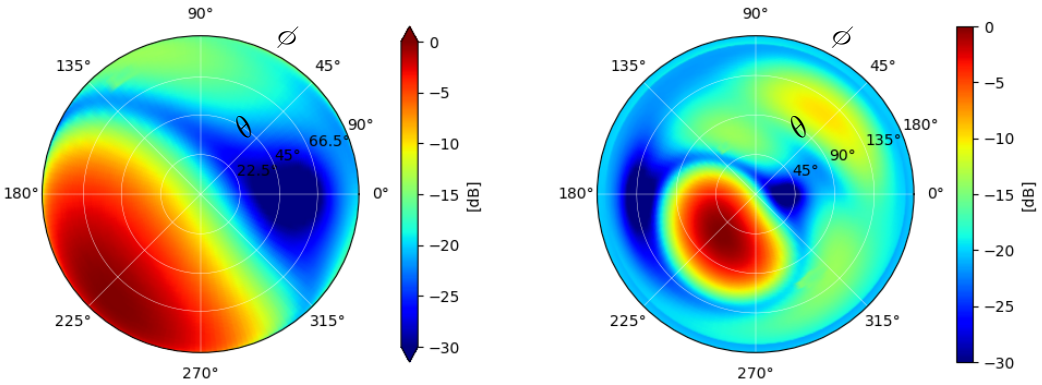
(a) Magnitude do campo elétrico da antena no referencial local

(b) Magnitude do campo elétrico da antena no referencial global

Figura 33 – Rotação do campo elétrico da antena

5.2 Gráfico de esfera invertida

O gráfico de esfera invertida é muito utilizado para analisar o campo elétrico emitido por uma antena em todas as direções em uma única figura. Este trabalho utiliza 2 definições deste gráfico, a primeira e mais comum é definida pela projeção do campo elétrico da antena sobre o plano xy. A segunda definição é a projeção dos campos elétricos da antena sobre uma esfera de raio unitário em torno da antena. A segunda definição é útil pois permite a visualização de todo o campo elétrico da antena em um gráfico bidimensional, enquanto a primeira definição permite apenas a visualização de uma semiesfera da antena por vez, pois ambas semiesferas ocupam a mesma projeção sobre o plano xy, esta definição permite entretanto uma melhor análise dos campos elétricos emitidos nesta semiesféra. Uma comparação das duas definições é ilustrada na Figura 34. Na Figura 34a, as linhas radiais marcam do centro do gráfico para fora os ângulos polares θ de θ_1, θ_2 e θ_3 e a linha mais externa do gráfico marca um ângulo de $\theta = 90°$. É observado que esta é uma escala não linear, assim, surge o efeito de ampliação sobre o centro do gráfico. As linhas radiais da Figura 34b marcam do centro do gráfico para fora os ângulos polares θ de θ_1, θ_2, θ_3, θ_4, θ_5 e θ_6, sendo uma escala linear, e a linha mais externa marca um ângulo $\theta = 180°$. Durante este trabalho, a segunda definição é utilizada por padrão para as figuras e análises. É perceptível que a primeira definição fornece uma ampliação da visualização no centro do gráfico pela segunda definição.



(a) Gráfico de esfera invertida pela definição 1

(b) Gráfico de esfera invertida pela definição 2

Figura 34 – Comparação dos gráficos de esfera invertida

5.3 Funcionamento do código desenvolvido

O código em *Python* é constituido por uma coleção de objetos, classes e métodos que constituem as variáveis e funções necessárias para modelar as antenas e os campos elétricos que elas emitem. As variáveis constituem o modelo que representa a antena e o arranjo de antenas no código. Algumas variáveis são responsáveis por organizar as antenas e ditar seu funcionamento, guardando os valores que representam suas coordenadas e orientação no sistema referencial global. Estas variáveis são chamadas de variáveis parametrizáveis, que são as posições x, y, z, orientação dada pelos ângulos de elevação, azimute e rolagem, e sua corrente de alimentação dada pela sua magnitude e fase. Para representar os campos elétricos da antena, há uma discretização espacial dos ângulos θ e ϕ, assim, o campo elétrico é representado por vetores e matrizes no código que representam o vetor de campo elétrico irradiado pela antena nos dados pontos de discretização.

O primeiro passo para utilização do código é a importação do campo elétrico emitido por uma antena. A geometria de cada antena não é incluída nos cálculos executados numericamente, e o código não é responsável pelo projeto de cada antena individualmente. Não há como realizar simulações de campo elétrico no código, assim, as antenas a serem utilizadas devem ser importadas de outro software. As antenas projetadas no *HFSS* foram utilizadas para todas as simulações realizadas no código. O *HFSS* exporta os valores do vetor de campo elétrico da antena em pontos já discretizados de θ e ϕ, e o código dispõe de um função especificamente para importar os campos elétricos de antenas exportadas diretamente do *HFSS*. A antena modelada herda a discretização do *HFSS*, mas o domínio discretizado pode ser mudado com interpolações bidimensionais dos pontos.

Com as antenas que serão utilizadas nos arranjos importadas para *Python*, o usuário pode construir um arranjo e simular seu campo elétrico resultante. As antenas do arranjo podem ser fornecidas em sua iniciação ou adicionadas posteriormente. Os parâmetros das antenas devem então ser modificados para projetar o arranjo conforme o desejado. As variáveis parametrizáveis são a posição da antena dada pelas coordenadas x, y, e z em relação ao sistema de referência local do arranjo, a orientação da antena dada pelos ângulos de elevação e azimute, e a alimentação da antena dada pela sua magnitude e fase. Cada antena deve ser parametrizada separadamente, é importante que o arranjo final faça sentido fisicamente, pois o código não leva em consideração as dimensões físicas das antenas e o resultado pode ser um arranjo cujas antenas colidem fisicamente. A proximidade entre antenas também pode levar à indução de correntes não desejadas entre as antenas, e este problema não é levado em consideração nas simulações. Desta forma, é importante garantir que as antenas estejam posicionadas no campo distante umas das outras.

Com o arranjo projetado, a simulação é executada. O objeto em Python que representa o arranjo é uma extensão do objeto que representa uma antena. A função que

calcula o campo elétrico do arranjo substitui a respectiva função no objeto da antena, esta função é responsável por calcular o campo elétrico no sistema referencial local do arranjo. O primeiro passo é determinar os campos elétricos de cada antena em seu respectivo sistema referencial e no sistema referencial global. Para cada antena que compõe o arranjo, o código atualiza seu campo elétrico. Bandeiras são utilizadas para sinalizar se uma antena necessita ser atualizada novamente, diminuindo o número de cálculos necessários. Ao atualizar a antena, seu campo elétrico é rotacionado de acordo com sua orientação. O arranjo então adiciona a contribuição do fator de arranjo com a posição de cada antena. O campo elétrico resultante do arranjo é a soma dos campos elétricos de todas suas antenas, ponderado pelo seu respectivo fator de arranjo. Ao final, o campo elétrico do arranjo é rotacionado de acordo com sua orientação, tal qual suas antenas. Assim, seu campo elétrico é representado no sistema referencial global.

Os códigos trabalham com o vetor de radiação $\vec{F}(\theta, \phi)$ das antenas e arranjos. Nos equacionamentos deste trabalho, são utilizados ambos $\vec{F}$ e o vetor de campo elétrico $\vec{E}$ dependendo do contexto. Conforme demonstrado, $\vec{E}(r, \theta, \phi) = \vec{G}(r)\vec{F}(\theta, \phi)$ e todas as equações supõem uma distância à antena r muito grande, de forma que $\vec{G}(r)$ se torne constante para pequenas translações das antenas e então seja possível trabalhar com o campo elétrico ou o vetor de radiação da mesma maneira.

5.4 Rotações das antenas e interpolação do campo elétrico

O arranjo precisa determinar para cada antena qual é o campo elétrico emitido por tal antena na direção dos pontos de amostragem do arranjo. Devido às rotações da antena, os pontos de amostragem fornecidos pela antena estão normalmente completamente desalinhados com os pontos de amostragem requeridos pelo arranjo. Assim, se faz necessário uma interpolação dos campos elétricos sobre os pontos de amostragem de interesse. Uma primeira tentativa é levar os pontos de amostragem da antena de seu sistema referencial local para o sistema referencial global do arranjo, entretanto, esta abordagem cria um problema interessante. A Figura 35 ilustra, para um simples problema com poucos pontos de amostragem para melhor visualização, o que acontece com a malha interpolada da antena quando a matriz de rotação $\overline{\overline{R}}_G^L$ é aplicada sobre esta malha, levando os pontos de interpolação para o sistema global. Para ocorrer interpolação, os pontos a serem amostrados (em azul) devem ser completamente englobada pela malha que contém os pontos de interpolação iniciais, o que é evidente na figura que não ocorre. Um ponto de amostragem da antena, após rotacionado, pode se encontrar em qualquer lugar no sistema global, podendo ser posicionado de forma praticamente aleatória dentro do intervalo de 0° a 90° em θ e de -180° a 180° em ϕ. Raramente haverá algum ponto posicionado sobre a borda do intervalo.

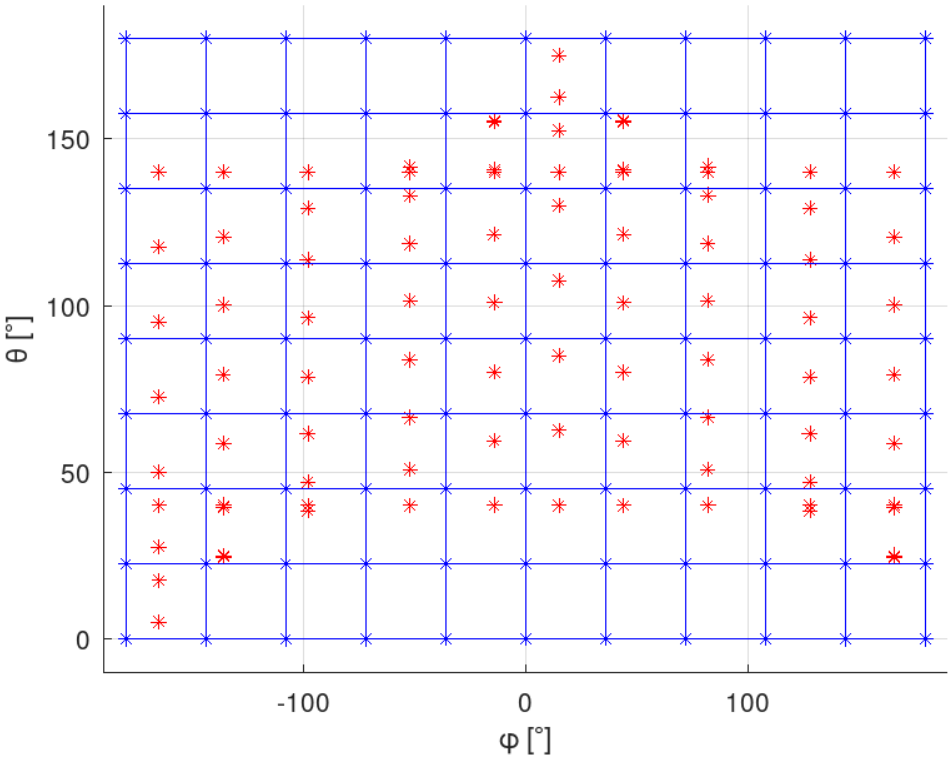


Figura 35 – Malha da antena (vermelho) e malha do arranjo (azul) no sistema referencial do arranjo

Em contrapartida, a malha a ser interpolada do arranjo englobam, geralmente, todo o intervalo de ângulos θ e ϕ possíveis. Assim, naturalmente, os pontos nas bordas do intervalo se encontraram fora da malha da antena. Matematicamente, isto não é um problema, pois a malha apresenta simetrias radiais, o que significa que basta interpolar os pontos utilizando os pontos do outro lado da malha. Na prática, isto é difícil de ser implementado eficientemente, pois demanda um processamento dos dados complicado ou duplicar a malha algumas vezes para garantir que a malha da antena englobe completamente a malha do arranjo.

Existe, entretanto, um método muito mais elegante para contornar este problema. Ao invés de levar a malha da antena ao sistema referencial global, a malha do arranjo é levado ao sistema referencial local, como pode ser visualizado na Figura 36. O mesmo efeito que acontecia anteriormente com a malha da antena agora acontece com a malha do arranjo, que então é sempre englobada pela malha da antena, mesmo no caso extremo de que os pontos de fronteira coincidam com os pontos de fronteira da malha da antena.

O campo elétrico da antena é interpolado, elemento por elemento, nos pontos de amostragem da malha do arranjo, mas as polarizações se referem às polarizações referenciais da antena. Assim, é necessário determinar a polarização equivalente no sistema referencial do arranjo. Isto pode ser feito de duas maneiras com preço computacional

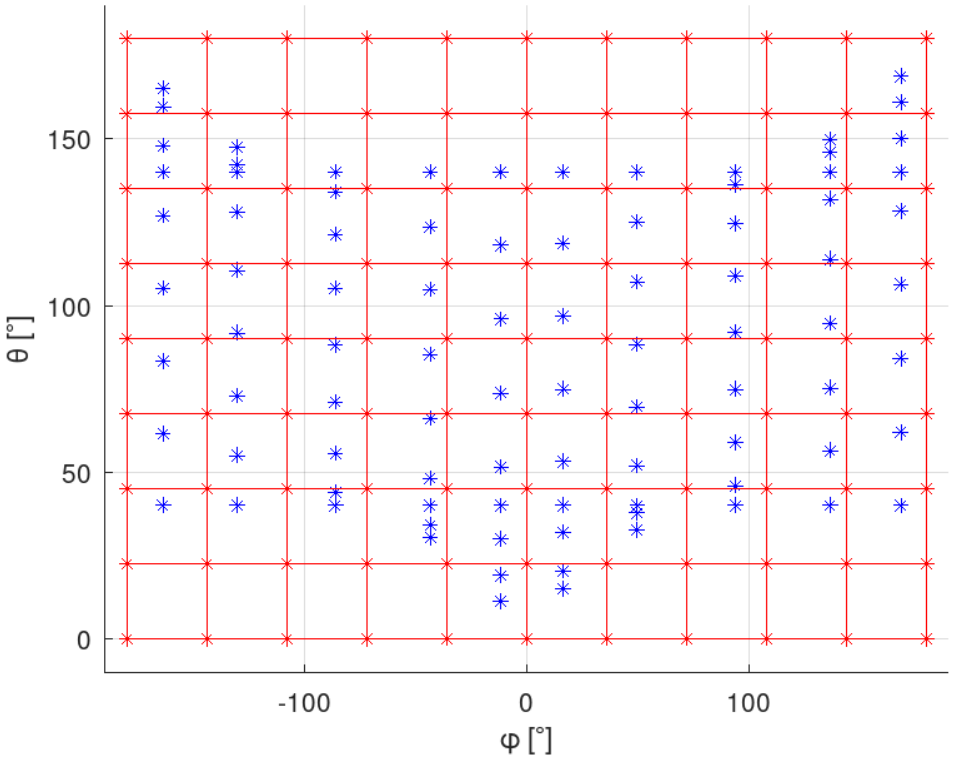


Figura 36 – Malha da antena (vermelho) e malha do arranjo (azul) no sistema referencial da antena

parecido: Levar os vetores unitários de polarização referencial do arranjo ao sistema referencial da antena; ou levar os campos elétricos interpolados ao sistema referencial do arranjo. De qualquer maneira, tem-se o campo elétrico resultante na forma da soma de 2 vetores, e este é comparado com os vetores referenciais de polarização do arranjo por meio de um produto interno. Esta operação é o mesmo que uma mudança de base por meio de um produto interno do vetor com os vetores unitários da nova base. A segunda forma é mais interessante pois realiza a rotação apenas sobre 1 vetor de campo elétrico resultante para depois realizar 2 produtos internos, enquanto a primeira forma realiza 2 rotações e 2 produtos internos. O processo pode ser melhor entendido observando o equacionamento dado nas Equações 5.2 a 5.12.

Primeiro, são definidos os vetores unitários referenciais do arranjo $\vec{r}_G$, $\vec{\theta}_G$ e $\vec{\phi}_G$ no próprio sistema referencial. Então são determinados seus equivalentes no sistema referencial da antena $\hat{r}_{LG}$, $\hat{\theta}_{LG}$ e $\hat{\phi}_{LG}$. Para cada vetor $\vec{u}_{S_1S_2}$, S_1 representa o sistema referencial original do vetor e S_2 representa o sistema referencial onde este está sendo representado, e quando $S_1 = S_2$ a segunda cópia é omitida resultando em $\vec{u}_{S_1}$. Por exemplo, $\hat{\theta}_{GL}$ representa o vetor unitário de $\hat{\theta}_G$ do sistema referencial global, representado no sistema local de uma antena, conforme a Equação 5.4. A necessidade de manter o na notação o sistema referencial original (G neste caso) é para diferenciar os vetores pertencentes ao arranjo como um todo das antenas individuais do arranjo, como por exemplo diferenciar o vetor

$\hat{\theta}_{GL}$ do vetor $\hat{\theta}_L$, sendo o segundo um vetor unitário local de uma antena. Por meio destes vetores, são então determinados os ângulos de apontamento da malha de amostragem do arranjo no sistema referencial da antena θ_{GL} e ϕ_{GL} por meio das operações descritas na Seção 1.1. Assim é possível interpolar os campos elétricos da antena sobre estes ângulos de apontamento determinando o campo elétrico interpolado $\vec{E}_{L_{interp}}$, utilizando uma função de interpolação de vetores $Interp\{\}$. Este vetor é levado ao sistema referencial do arranjo, resultando em $\vec{E}_{L_{interp}G}$. Finalmente, são determinados os componentes do campo elétrico interpolado utilizando a base de referenciais de polarização do arranjo $E_{G_{interp}\theta}$ e $E_{G_{interp}\phi}$, e assim é construído o vetor de campo elétrico da antena interpolado e representado no sistema referencial do arranjo $\vec{E}_{G_{interp}}$.

$$\vec{E}_L = E_{L\theta}\hat{\theta}_L + E_{L\phi}\hat{\phi}_L \tag{5.2}$$

$$\hat{r}_{GL} = \overline{\overline{R}}_L^G \hat{r}_G \tag{5.3}$$

$$\hat{\theta}_{GL} = \overline{\overline{R}}_L^G \hat{\theta}_G \tag{5.4}$$

$$\hat{\phi}_{GL} = \overline{\overline{R}}_L^G \hat{\phi}_G \tag{5.5}$$

$$E_{L_{interp}\theta} = Interp\{E_{L\theta}, \theta_{GL}, \phi_{GL}\} \tag{5.6}$$

$$E_{L_{interp}\phi} = Interp\{E_{L\phi}, \theta_{GL}, \phi_{GL}\} \tag{5.7}$$

$$\vec{E}_{L_{interp}} = E_{L_{interp}\theta}\hat{\theta}_L + E_{L_{interp}\phi}\hat{\phi}_L \tag{5.8}$$

$$\vec{E}_{L_{interp}G} = \overline{\overline{R}}_G^L \vec{E}_{L_{interp}} \tag{5.9}$$

$$E_{G_{interp}\theta} = \hat{\theta}_G \cdot \vec{E}_{L_{interp}G} \tag{5.10}$$

$$E_{G_{interp}\phi} = \hat{\phi}_G \cdot \vec{E}_{L_{interp}G} \tag{5.11}$$

$$\vec{E}_{G_{interp}} = E_{G_{interp}\theta}\hat{\theta}_G + E_{G_{interp}\phi}\hat{\phi}_G \tag{5.12}$$

Este conjunto de operações é o coração deste trabalho, que permite rotacionar e interpolar livremente os campos elétricos das antenas que compõem um arranjo e assim construir arranjos complexos para as mais diversas aplicações. Aplicando estas operações, a Equação 1.130 se torna 5.13, que é a equação completa do campo elétrico resultante do arranjo e define os algoritmos implementados neste trabalho. Nesta equação, i define a i-ésima antena das N antenas que compõem o arranjo, I_i é a sua corrente de alimentação e $\vec{p}_i$ é sua posição no sistema referencial global do arranjo, $\vec{E}_{G_{interp}i}$ é o campo elétrico da i-ésima antena rotacionado e interpolado conforme das operações acima, e $\vec{E}_a$ é o campo elétrico resultante do arranjo.

$$\vec{E}_a = \sum_{i=1}^{N} \mathrm{e}^{-\vec{k}\cdot\vec{p}_i} I_i \vec{E}_{G_{interp}i} \tag{5.13}$$

Fatorando os vetores de campo elétrico em uma parte dependente da distância

$G(r)$ e o vetor de radiação $\vec{F}(\theta,\phi)$, a equação se torna 5.15. Conforme demonstrado anteriormente, $G(r)$ tem a forma da Equação 5.17. Com a simplificação de campos distantes, $G(r)$ é constante para pequenas translações das antenas do arranjo e invariante para as rotações das antenas. Além disso, $G(r)$ é a mesma relação para todas as antenas do arranjo, independente do modelo de cada uma, pois as características individuais de cada antena são contabilizadas em suas distribuições de radiação, que definem os vetore de radiação. Desta maneira, $G(r)$ se torna uma constante para todas as antenas e em todas as operações para a interpolação dos campos elétricos, que pode ser colocada em evidência fora do somatório dos campos elétricos para o arranjo. Assim, igualando a parte dependente de r do campo elétrico com a parte dependente de r das antenas, tem-se que a função $G_a(r)$ do arranjo é igual às funções $G_i(r)$ de cada antena, e igual à um $G(r)$ do sistema. Então ao projetar um arranjo de antenas, a disposição e alimentação das antenas modifica o vetor de radiação do arranjo, e não a função $G(r)$. Por isso o vetor de radiação é utilizado na comparação de antenas e para projetar o arranjo.

$$\vec{E}_i(r,\theta,\phi) = G_i(r) I_i \vec{F}_i(\theta,\phi) \tag{5.14}$$

$$\vec{E}_a(r,\theta,\phi) = G_a(r) I_a \vec{F}_a(\theta,\phi) \tag{5.15}$$

$$G(r) = \frac{E_0}{r} \mathrm{e}^{-jk_0 r} \tag{5.16}$$

$$G_a(r)\vec{F}_a(\theta,\phi) = \sum_{i=1}^{N} \mathrm{e}^{-\vec{k}(\theta,\phi)\cdot\vec{p}_i} I_i G_i(r) \vec{F}_i(\theta,\phi) \tag{5.17}$$

$$G_i(r) = G(r) \forall\ i \tag{5.18}$$

$$G_a(r)\vec{F}_a(\theta,\phi) = G(r) \sum_{i=1}^{N} \mathrm{e}^{-\vec{k}(\theta,\phi)\cdot\vec{p}_i} I_i \vec{F}_i(\theta,\phi) \tag{5.19}$$

$$G_a(r) = G(r) \tag{5.20}$$

$$\vec{F}_a(\theta,\phi) = \sum_{i=1}^{N} \mathrm{e}^{-\vec{k}(\theta,\phi)\cdot\vec{p}_i} I_i \vec{F}_i(\theta,\phi) \tag{5.21}$$

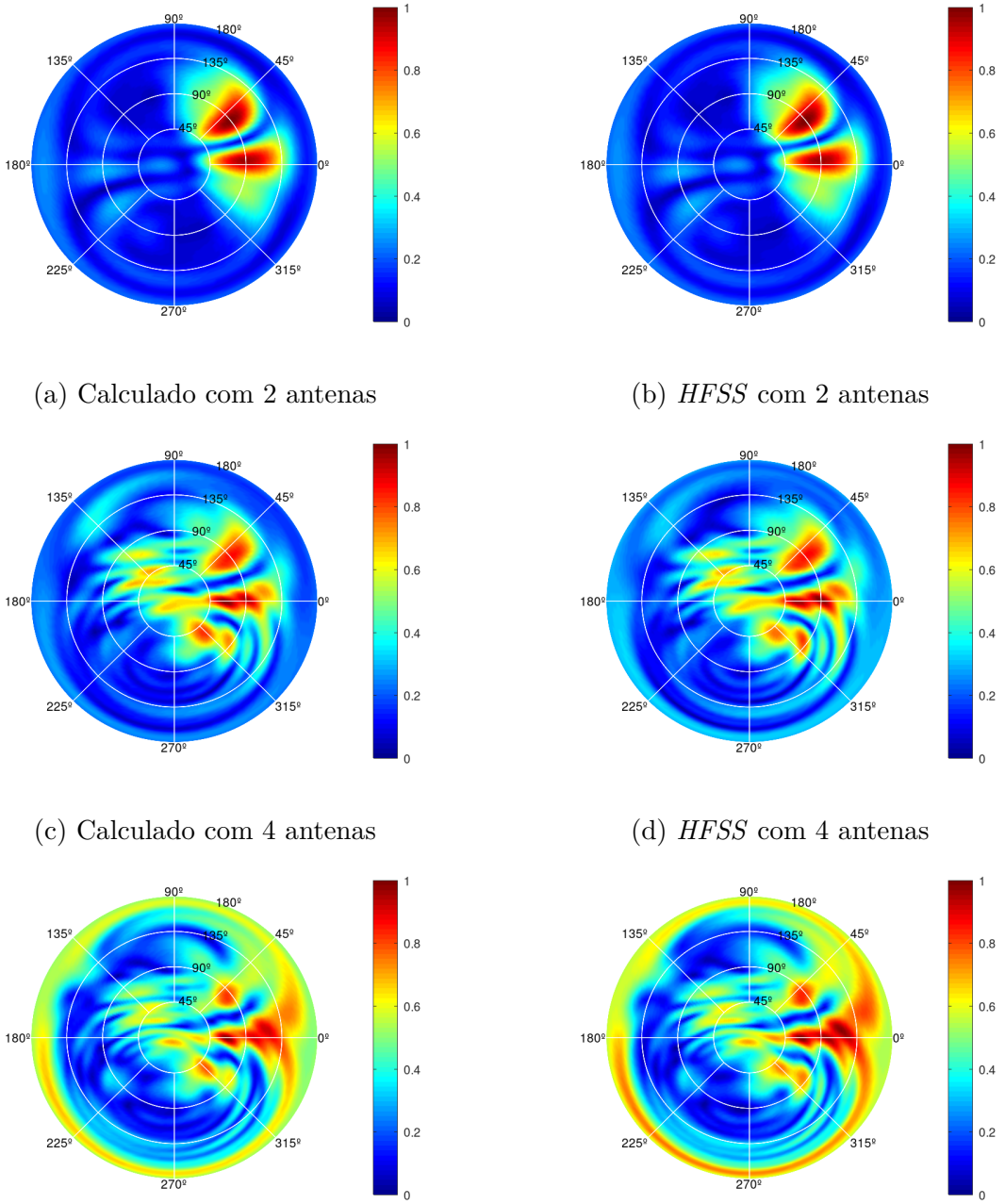
$$\tag{5.22}$$

6 Validação do código desenvolvido

A Figura 37 contém a magnitude do campo elétrico resultante para diversos arranjos projetados e simulados no *HFSS* e no código desenvolvido em *Octave*. Cada par de figuras representa o mesmo arranjo, simulado em ambos os programas. Assim, percebe-se que os resultados dos códigos desenvolvidos foram muito simulares aos resultados obtidos pelas simulações numéricas no *HFSS*, assim, validando a capacidade do código de simulação de encontrar os campos elétricos resultantes de arranjos de antenas reais.

Após validar o código implementado em *Octave*, os mesmos algoritmos foram implementados em *Python*. Isto foi feito para facilitar as operações de criação e modificação de antenas e arranjos de antenas utilizando interfaces gráficas e bibliotecas numéricas disponíveis em *Python*. Assim, houve uma grande flexibilização da ferramenta desenvolvida, além da possibilidade de novas funções e ferramentas serem implementadas de forma mais fácil. Uma das ferramentas desenvolvidas em *Python* foi a implementação de ferramentas de análise e visualização interativa dos resultados de forma mais didática e prática.

Os gráficos do vetor de radiação de polarização θ e ϕ podem ser calculados facilmente com os códigos desenvolvidos para uma antena ou arranjo qualquer. Entretanto, o próprio sistema de coordenadas introduz uma dificuldade na análise dos diagramas de radiação nestas polarizações pela existência de descontinuidades nos pólos (em $\theta = 0°$). Estas descontinuidades se tornam proeminentes quando se está interessado em visualizar o diagrama de uma antena que aponta diretamente sobre os pólos, dificultando a separação do padrão de radiação referencial do padrão de radiação cruzada, como pode ser observado na Figura 38 das polarizações theta e phi de um arranjo com duas antenas Yagi-Uda com 4 elementos.



(a) Calculado com 2 antenas

(b) *HFSS* com 2 antenas

(c) Calculado com 4 antenas

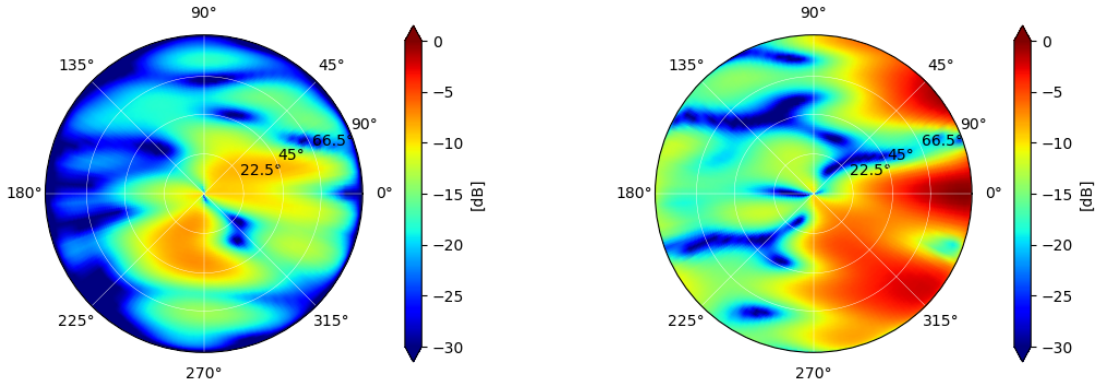
(d) *HFSS* com 4 antenas

(e) Calculado com 5 antenas

(f) *HFSS* com 5 antenas

Figura 37 – Campo elétrico dos arranjos de antenas Yagis de 4 elementos

Baseando-se na terceira definição de polarização cruzada de Ludwig (1973), para melhor analisar os campos elétricos gerados pelas antenas e arranjos de diferentes polarizações, foi implementada uma visualização das polarizações de referência e cruzadas diretamente nos códigos desenvolvidos. Utilizando a matriz de mudança de base disposta na Equação 6.1, as polarizações theta e phi do campo elétrico se tornam as polarizações de referência e cruzada, dadas respectivamente pelas Equações 6.2 e 6.3.



(a) *HFSS* Ftheta (b) *HFSS* Fphi

Figura 38 – Campo elétrico resultante de um arranjo com 2 antenas Yagis de 4 elementos

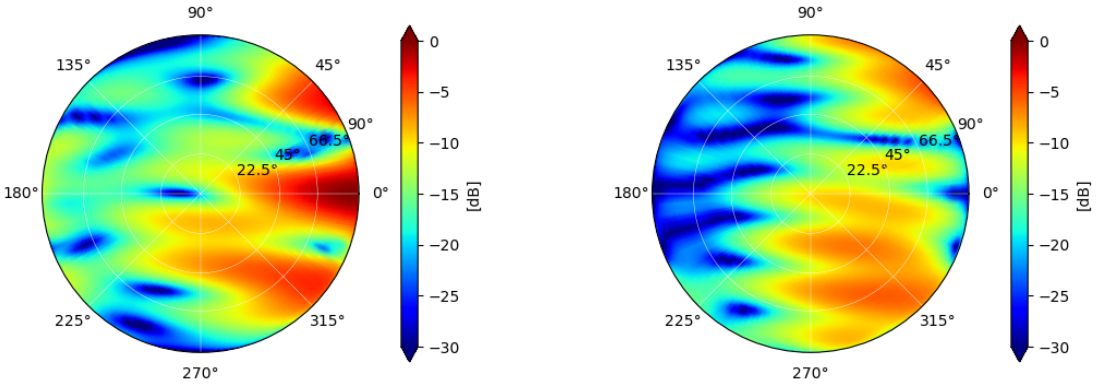
$$\begin{bmatrix} E_x \\ E_y \\ E_z \end{bmatrix} = \begin{bmatrix} 1-\sin^2\theta\cos^2\phi & -\sin^2\theta\sin\phi\cos\phi & -\sin\theta\cos\theta\cos\phi \\ -\sin^2\theta\sin\phi\cos\phi & 1-\sin^2\theta\sin^2\phi & -\sin\theta\cos\theta\sin\phi \\ -\sin\theta\cos\theta\phi & -\sin\theta\cos\theta\sin\phi & 1-\cos^2\theta \end{bmatrix} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} \tag{6.1}$$

$$\vec{E}_{ref}(\theta,\phi) = \vec{E}(\theta,\phi)\cdot(\hat{\theta}\sin\phi + \hat{\phi}\cos\phi) \tag{6.2}$$

$$\vec{E}_{cross}(\theta,\phi) = \vec{E}(\theta,\phi)\cdot(\hat{\theta}\cos\phi - \hat{\phi}\sin\phi) \tag{6.3}$$

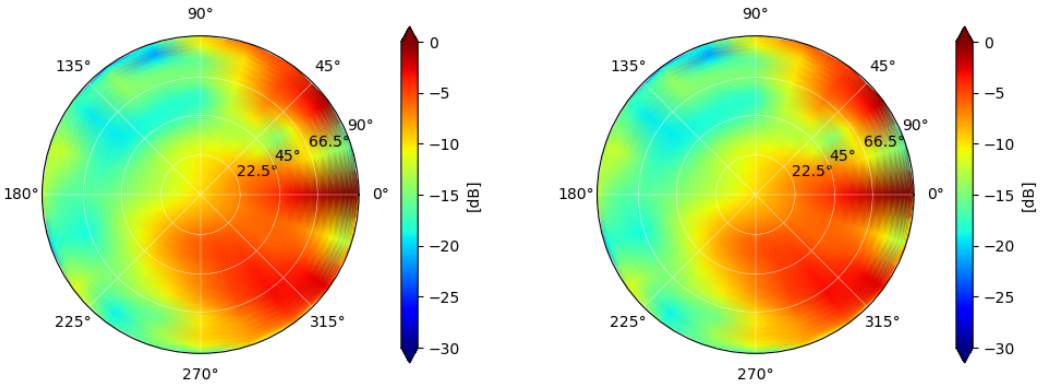
O resultado desta mudança de base é uma melhor visualização dos campos elétricos resultantes das antenas, permitindo uma melhor análise do padrão de radiação da antena em ambas polarizações, conforme pode ser observado na Figura 39, que ilustra as polarizações de referência e cruzada do mesmo arranjo presente na Figura 38.

O cálculo da magnitude do campo elétrico a partir dos novos vetores de polarização referencial e cruzada confirma que não houve uma mudança do campo elétrico resultante, apenas uma mudança de sistema de coordenadas, conforme pode ser visualizado na Figura 40.



(a) HFSS Fref (Polarização referencial) (b) HFSS Fcross (Polarização cruzada)

Figura 39 – Campo elétrico resultante de um arranjo com 2 antenas Yagis de 4 elementos utilizando a definição de polarização referencial e cruzada



(a) Módulo do campo elétrico calculado a partir dos vetores de radiação de polarizações $\hat{\theta}$ e $\hat{\phi}$

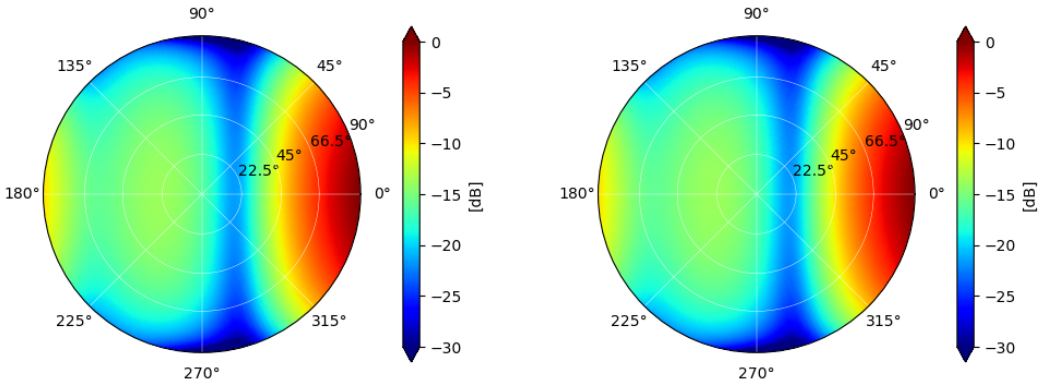
(b) Módulo do campo elétrico calculado a partir dos vetores de radiação de polarizações referencial e cruzada

Figura 40 – Comparação da magnitude do campo elétrico resultante com diferentes definições de polarizações

Utilizando os diagramas de polarização referencial e cruzada para os arranjos de 1 a 5 antenas, dispostos nas Figuras 41, 42, 43, 44 e 45, é possível comparar o resultado do código desenvolvido em Python com as simulações numéricas realizadas no *HFSS*. Obviamente, o "arranjo"com uma única antena é na prática o mesmo que esta antena sozinha, mas este resultado é utilizado para conferir se o objeto de arranjos não modifica as propriedades das antenas individualmente. O arranjo de validação é definido na Tabela 5 onde estão dispostas as posições, orientações e correntes de alimentação de cada antena do arranjo. O arranjo da Figura 41 corresponde à primeira antena da Tabela 5, o arranjo da Figura 42 corresponde às primeiras duas antenas da Tabela 5, e assim por diante.

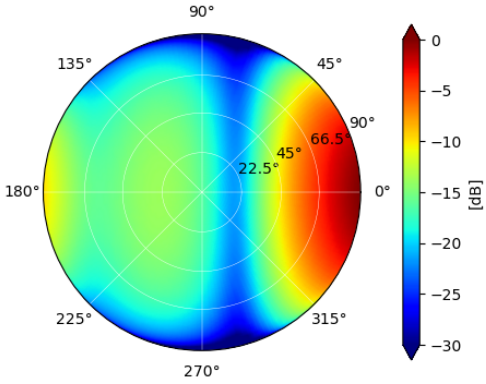
	Posição			Orientação			Alimentação	
Antenna	X	Y	Z	Elevation [o]	Azimuth [°]	Roll [o]	Magnitude	Phase [o]
Yagi 4 elements	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
Yagi 4 elements	0.00	1.50	0.00	0.00	45.00	0.00	1.00	0.00
Yagi 4 elements	0.00	-1.50	0.00	-45.00	-45.00	0.00	1.00	0.00
Yagi 4 elements	0.34	-3.14	1.42	-45.00	135.00	0.00	1.00	0.00
Yagi 4 elements	0.83	1.19	-0.72	72.00	14.00	0.00	1.00	0.00

Tabela 5 – Posição, orientação e alimentação das antenas utilizadas nos arranjos de validação do código

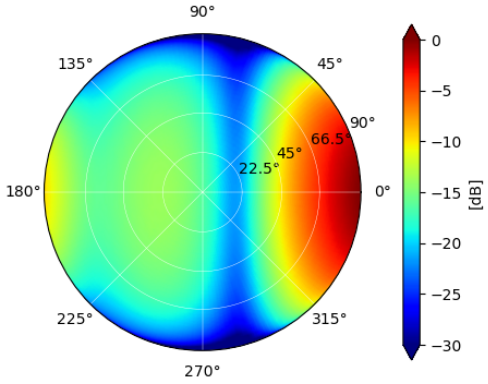


(a) Python F

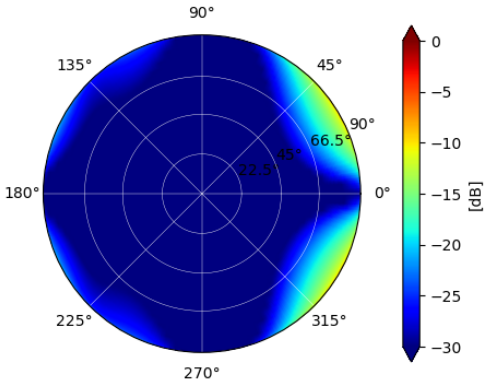
(b) *HFSS* F



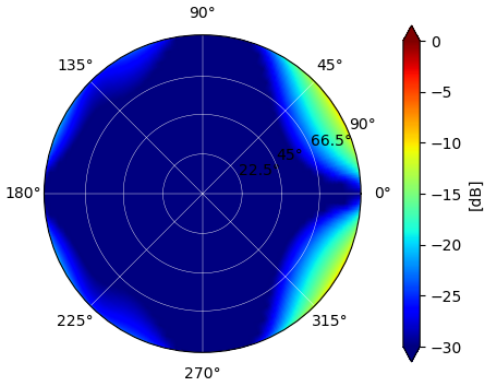
(c) Python Fref



(d) *HFSS* Fref

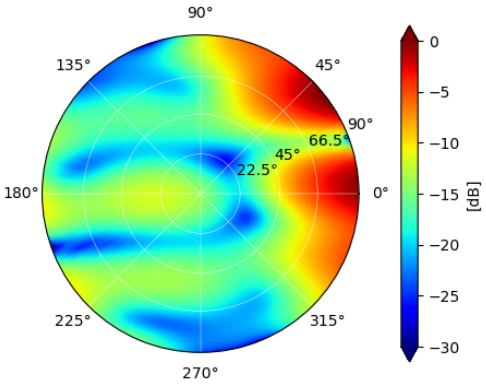


(e) Python Fcross

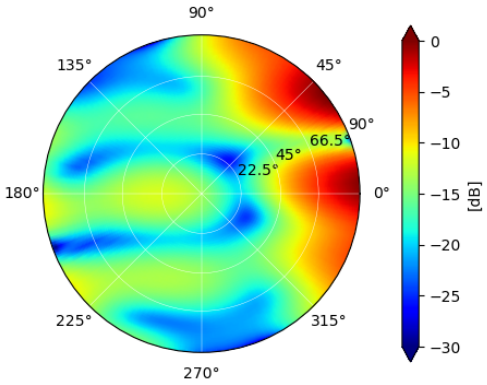


(f) *HFSS* Fcross

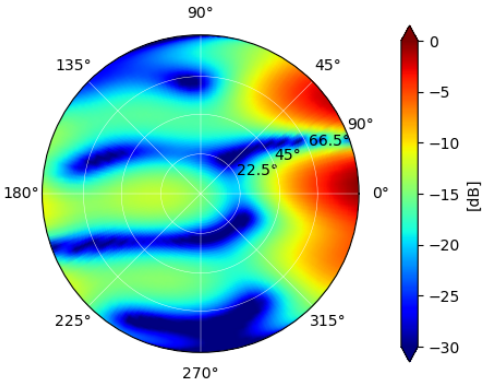
Figura 41 – 1 Yagi com 4 elementos



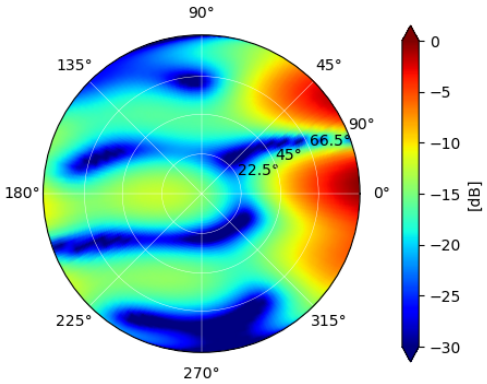
(a) Python F



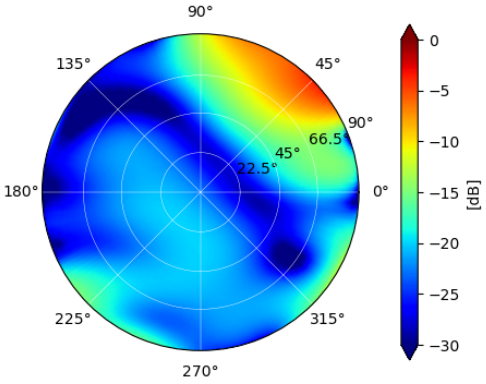
(b) *HFSS* F



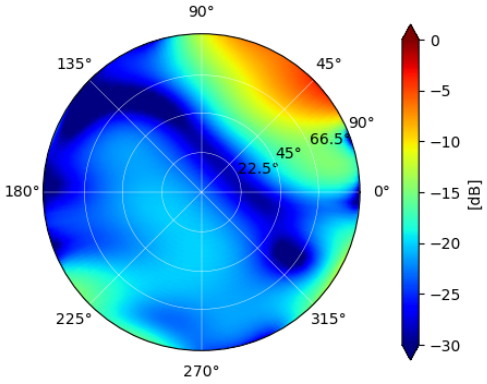
(c) Python Fref



(d) *HFSS* Fref

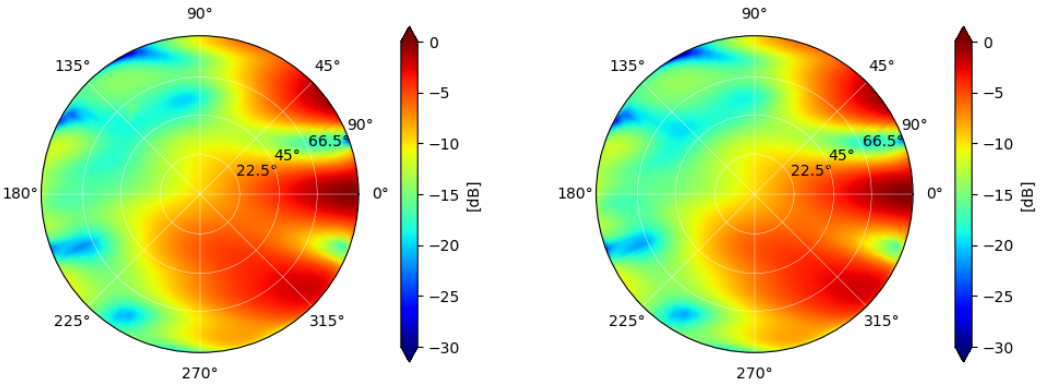


(e) Python Fcross



(f) *HFSS* Fcross

Figura 42 – 2 Yagis com 4 elementos



(a) Python F

(b) *HFSS* F

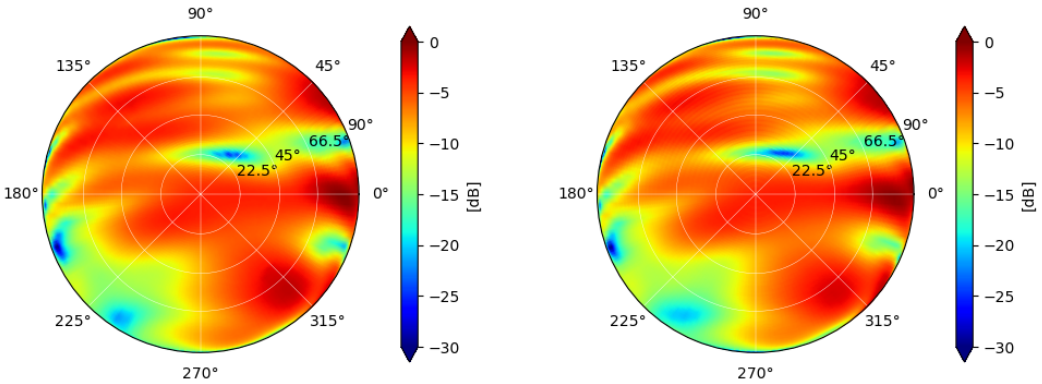
(c) Python Fref

(d) *HFSS* Fref

(e) Python Fcross

(f) *HFSS* Fcross

Figura 43 – 3 Yagis com 4 elementos



(a) Python F

(b) *HFSS* F

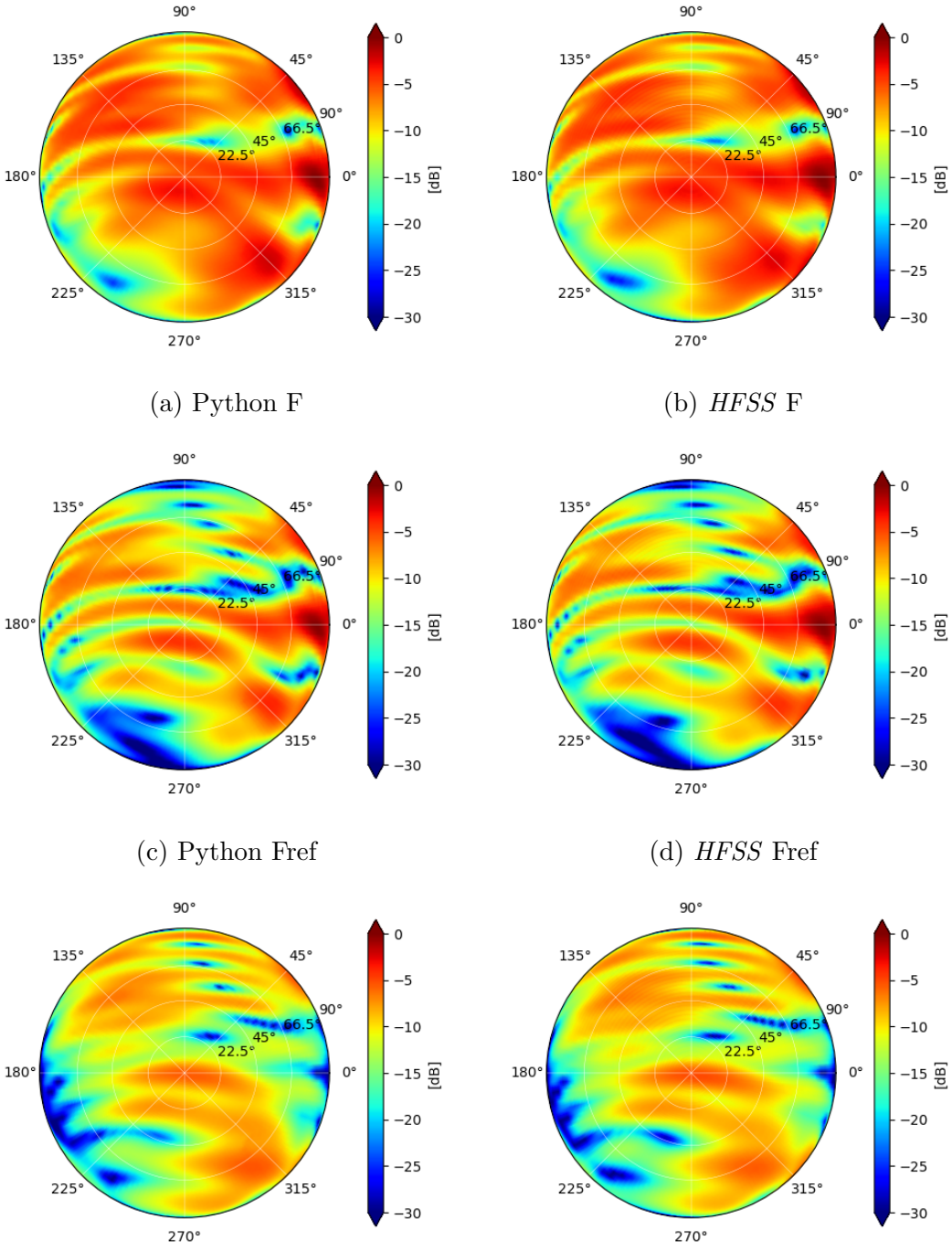
(c) Python Fref

(d) *HFSS* Fref

(e) Python Fcross

(f) *HFSS* Fcross

Figura 44 – 4 Yagis com 4 elementos



(a) Python F

(b) *HFSS* F

(c) Python Fref

(d) *HFSS* Fref

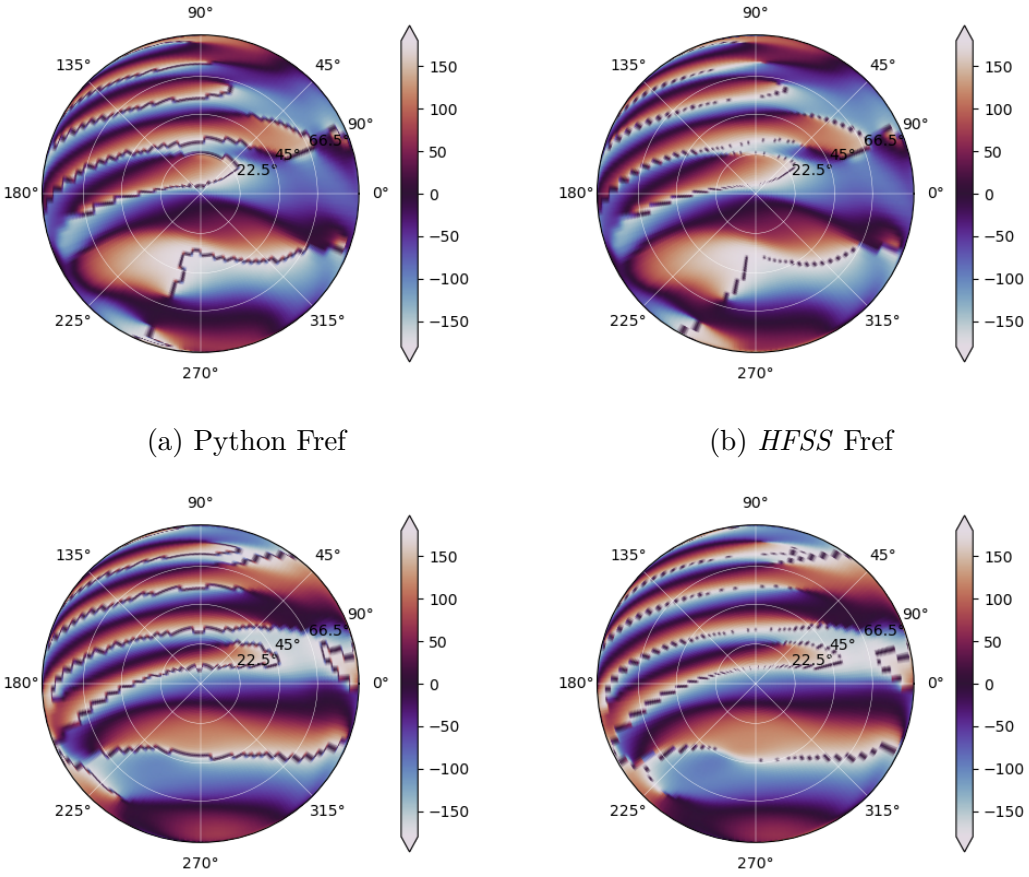
(e) Python Fcross

(f) *HFSS* Fcross

Figura 45 – 5 Yagis com 4 elementos

Por fim, é interessante verificar a fase das polarizações dos campos elétricos produzidos pelos arranjos e comparar com as simulações realizadas no *HFSS*. As Figuras em 46 mostram que, de fato, a fase dos campos elétricos resultantes do arranjo com 5 antenas é a mesma tanto para as simulações realizadas no *HFSS* como nas simulações realizadas em *Python*.

Assim, os resultados mostram que o código desenvolvido consegue reproduzir os



(a) Python Fref

(b) *HFSS* Fref

(c) Python Fcross

(d) *HFSS* Fcross

Figura 46 – 5 Yagis com 4 elementos

resultados das custosas simulações numéricas do *HFSS*, embora, para arranjos com um maior número de antenas, é perceptível ligeiras diferenças entre os resultados.

A tabela 6 contém o tempo medido para calcular o campo elétrico resultante dos arranjos de 2, 3 e 5 antenas Yagi-Uda com 4 elementos no *HFSS* e no código desenvolvido em *Octave*. Nota-se que o código desenvolvido realiza o cálculo do campo elétrico resultante ordens de magnitude mais rapidamente que a simulação numérica realizada no *HFSS* para um mesmo arranjo. O campo elétrico resultante de um arranjo é calculado na ordem de milisegundos no código desenvolvido, enquanto as simulações numéricas podem demorar de minutos a horas. Assim, o código se torna ideal para a realização de milhares de simulações de um arranjo com variações de forma a realizar uma otimização. É perceptível que o tempo de processamento nos códigos desenvolvidos aumenta de forma mais linear conforme o número de antenas do arranjo aumenta, enquanto o tempo de simulação no *HFSS* aumenta exponencialmente. Isso se deve ao fato do número de elementos da simulação no *HFSS* ser proporcional ao volume de controle da simulação, que pode aumentar exponencialmente com a adição de novas antenas.

Número de antenas	*Octave*	*Python*	*HFSS*
1	57.14 ms	43.98 ms	4.03 min
2	76.27 ms	70.96 ms	9.62 min
3	131.67 ms	106.49 ms	10.65 min
4	163.89 ms	144.92 ms	18.13 min
5	204.26 ms	184.88 ms	21.78 min

Tabela 6 – Tempo de simulação

6.1 Considerações sobre o software desenvolvido

Os códigos desenvolvidos apresentam 8 variáveis parametrizáveis que definem individualmente a posição, orientação, e corrente de alimentação de cada antena de um arranjo arbitrário. A adição de cada antena adiciona 8 dimensões ao vetor de variáveis que podem ser parametrizadas. Este é um fator importante para as otimizações que podem ser realizadas nos arranjos.

O código utiliza simplificações das antenas que compõem os arranjos. Uma antena é modelada como uma fonte pontual de ondas eletromagnéticas. Assim, os efeitos indutivos entre antenas próximas são negligenciados. Isto é um requisito importante para a modelagem dos problemas, pois antenas muito próximas podem ter seu campo magnético modificado significantemente devido aos enfeitos indutivos de campo próximo. Assim, deve-se garantir que cada antena esteja posicionada na região de campo distante de todas suas vizinhas para garantir uma boa confiabilidade dos resultados da simulação.

Enquanto a demanda computacional de simulações no *HFSS* aumenta de acordo com o volume de controle da simulação, a demanda computacional do código desenvolvido aumenta linearmente com o número de antenas. O tempo necessário para calcular o campo elétrico de arranjos pequenos no código desenvolvido foi da ordem de milisegundos, enquanto o *HFSS* necessitou de minutos, e simulações com maior precisão ou mais antenas pode facilmente demandar horas. O código desenvolvido resultou ainda em aproximadamente o mesmo campo elétrico resultante para todos os arranjos simulados com precisão satisfatória. A simulação com 4 e 5 arranjos resultou em diferenças perceptíveis nos gráficos, mas que não são preocupantes para as aplicações atuais do trabalho. Estas diferenças provavelmente acontecem devido à proximidade entre as antenas do arranjo, que já começam a introduzir os efeitos de indução de corrente nas antenas vizinhas. Estas diferenças são aceitaveis até certo ponto. Para arranjos como o utilizado neste trabalho, é recomendado uma simulação numérica final em um *software* comercial após o desenvolvimento do arranjo em softwares de simulação de volume de controle como o *HFSS* para averiguar que o arranjo final continua atendendo aos requisitos do projeto. Mesmo com esta simulação final, a utilização dos códigos desenvolvidos pode ainda acelerar muito o desenvolvimento dos arranjos pois elimina a necessidade da maioria das simulações numéricas nos *softwares* comerciais.

O código desenvolvido se mostra ideal para aplicações com grande número de antenas, onde uma grande customização da posição, orientação, e corrente de alimentação de cada antena é necessária, e quando a distância entre cada par de antenas às coloca na região de campo distante de suas vizinhas, oferencendo mais flexibilzação, praticidade e rapidez no cálculo do campo elétrico resultante dos arranjos. Em outros casos, *softwares* de simulações de volume de controle se mostram mais adequados pois levam em consideração a geometria de cada antena e os campos elétricos de campo próximo destas, que podem afetar as simulações do código desenvolvido de forma a invalidar seus resultados. Desta forma, os códigos desenvolvidos se mostram como uma ferramenta adicional à disposição dos projetistas de arranjos, não se posicionando como concorrente direto de *softwares* comerciais, pois suas aplicações são distintas, mas complementando em aplicações onde estes softwares comerciais podem se mostrar ineficazes.

7 Algoritmo de otimização

Para desenvolver um arranjo que atenda os requisitos de um projeto, se faz necessária uma otimização de forma a definir a posição e orientação de cada antena, bem como a magnitude e fase de sua alimentação. Assim, utilizando os códigos desenvolvidos neste trabalho foram realizadas otimizações para projetar arranjos (arranjo alvo) de antenas cujo diagrama de distribuição de radiação seja semelhante a um diagrama arbitrário (arranjo objetivo), construído com funções degrau unitário cujo argumento são os ângulos theta e phi de cada direção.

Tendo em mente um sistema com entradas e saídas, um algoritmo de otimização é um algoritmo que modifica as variáveis de entrada de um sistema arbitrário para que sua saída seja igual à saídas pré-determinadas. Quando as variáveis de saída do sistema são suficientemente semelhanes às variáveis objetivo, o algoritmo é interrompido e as variáveis de entrada do sistema são a solução encontrada para a otimização. Caso contrário, o algoritmo pode ser interrompido quando determinar que não é capaz de chegar mais perto do que seria uma solução boa o suficiente. Bisneto (2022) e Nocedal e Wright (1999) mostram diveros métodos de otimização numérica disponíveis. Dentre eles, métodos de otimização convexa, como mostrado por Boyd e Vandenberghe (2004) e Sousa (2021), se destacam por serem capazes de determinar rapidamente o mínimo global de uma função convexa.

7.1 Como funcionam otimizações

Uma otimização numérica é um processo que iterativamente tenta igualar o conjunto de saídas de um sistema potencialmente desconhecido à um conjunto de valores desejados quaisquer. As saídas do sistema podem ser valores discretos ou funções, em que então deve se igualar à uma função dada. Estas saídas são controladas indiretamente pelas entradas, e a relação entre entrada e saída pode ser muito dinâmica ou até mesmo desconhecida. Quando a saída do sistema é uma função, a diferença $Diff$ entre a função de saída e a função desejada é muitas vezes quantificada pela integral do quadrado da diferença entre as funções sobre um domínio S de interesse dado, conforme a Equação 7.1.

Para quantificar o quão similar o conjunto de saídas do sistema é do conjunto de saídas desejadas, o algoritmo de otimização utiliza uma função denominada função de custo (*Cost function*), cuja entrada são as saídas do sistema e sua saída, o custo (*Cost*), quantifica o quão perto ou longe a otimização está de ser finalizada. O algoritmo de otimização é responsável então por utilizar o custo atual do sistema para determinar qual

seleção das variáveis de entrada irá produzir o menor custo. Desta forma, o algoritmo de otimização deve minimizar a função de custo. O custo representa todas as diferenças entre a solução atual e a solução desejada em uma única variável, e pode ser definido de diversas maneiras dependendo do sistema a ser otimizado.

Muitas vezes o custo é definido como a soma do módulo das diferenças entre as variáveis de saída do sistema e seus respectivos valores ideais, como na Equação 7.2 da definição 1 da função de custo, ou também pode ser encontrado como a soma do quadrado do módulo das diferenças, como na equação 7.3 da definição 2 da função de custo.

$$Diff = \int_S (f_1(s) - f_2(s))^2 ds \tag{7.1}$$

$$Cost_1 = \sum_{n=1}^{N} |S_n - S_{in}| \tag{7.2}$$

$$Cost_2 = \sum_{n=1}^{N} (|S_n - S_{in}|)^2 \tag{7.3}$$

Ambas equações resultam em um número real positivo que o algoritmo de otimização deseja minimizar, idealmente zerando o custo. Para uma função ser uma função de custo válida, sua saída para qualquer entrada deve ser um número real maior ou igual a zero, pois caso contrário não possível definir um "mínimo"para a saída. As definições dadas são somas de número reais positivos, mas isso nem sempre é o caso. Uma função de custo poderia ser o módulo da soma de números quaisquer, até mesmo complexos. Normalmente as funções de custo não são definidas desta maneira pois a soma de números não estritamente reais e positivos poderia resultar em um número com módulo próximo de zero, mesmo com as saídas do sistema sendo muito diferentes das saídas desejadas.

A escolha da função de custo deve refletir os requisitos da solução que se deseja atingir. Assim, é necessária uma escolha criteriosa da função de custo, pois, uma boa função de custo pode até mesmo acelerar a execução da otimização, enquanto funções de custo não adequadas podem resultar na otimização finalizando em soluções que não atendem os requisitos do projeto.

7.2 Código de otimização

A otimização de um problema começa pela definição ds objetivos a serem atingidos pela mesma. Objetivos sendo os campos elétricos desejados ao final da otimização. Nos códigos desenvolvidos, uma antena é construída com os campos elétricos desejados. Os campos elétricos desta antena são definidos matematicamente conforme o necessário, assim, esta antena se torna apenas um artefato matemático utilizado para projetar uma

antena real. Esta antena abstrada pode ser também uma antena real, o que é utilizado para testar os algoritmos de otimização. O próximo passo da otimização é definir a função de custo mais adequada para o problema. O processo de otimização é melhor ilustrado no diagrama da Figura 47.

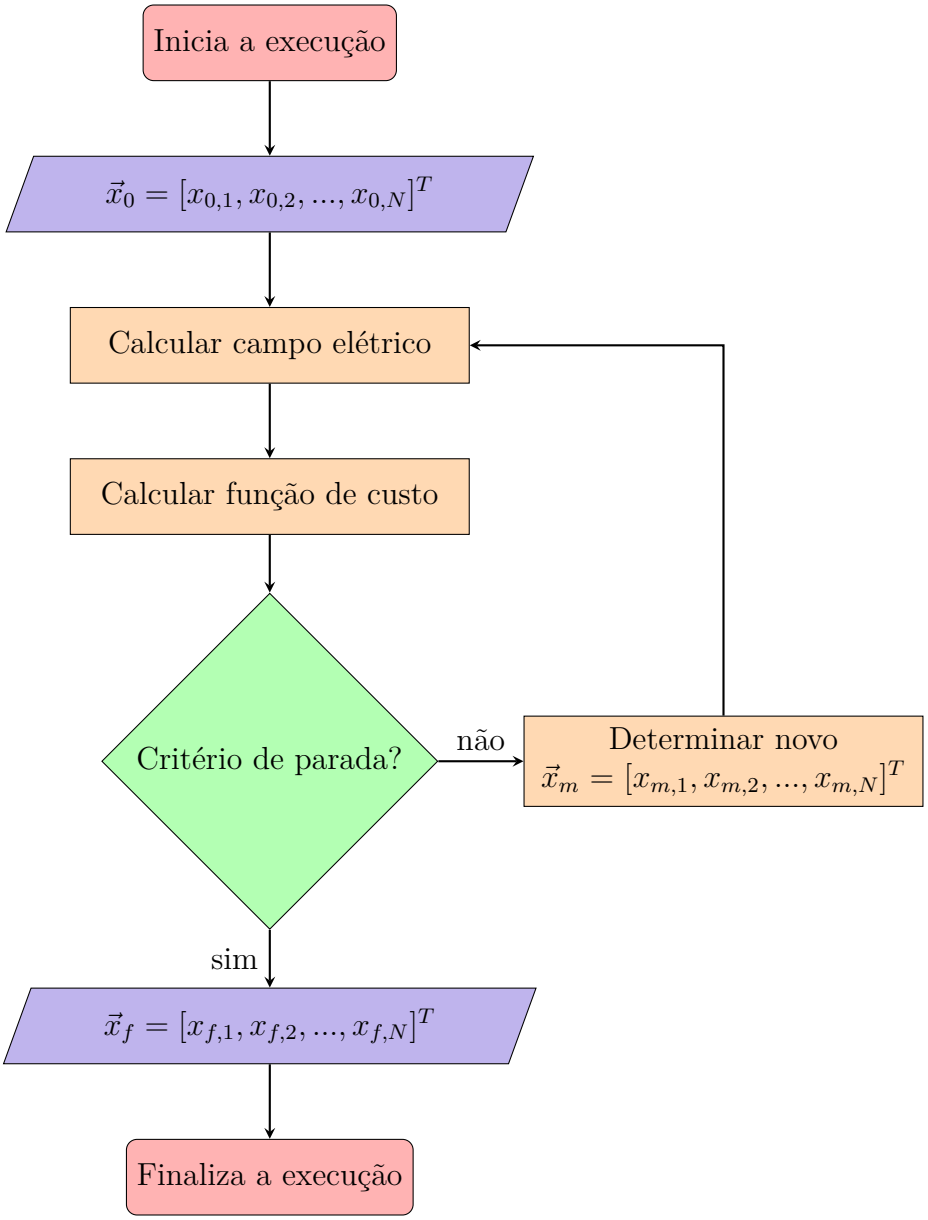


Figura 47 – Algoritmo de otimização numérica dos arranjos de antenas

As entradas do sistema podem ser definidas na forma de um vetor $\vec{x}$ de dimensão N igual ao número de variáveis parametrizadas a serem otimizadas. A otimização necessita de um valor inicial para o vetor $\vec{x}$, que pode ser chamado de $\vec{x}_0$. A partir da primeira iteração calculada com este vetor inicial, cada uma das seguintes iterações m resulta em um novo vetor $\vec{x}_m$, para o qual a função de custo é calculada novamente até que o algoritmo termine sua execução, conforme a Equação 7.5 onde uma iteração do algoritmo de otimização é dada pela função *Optim*. A Equação 7.4 mostra o vetor $\vec{x}$, onde cada um

de seus elementos é uma variável sendo otimizada.

$$\vec{x}_0 = \begin{bmatrix} x_{0,1} \\ x_{0,2} \\ x_{0,3} \\ ... \\ x_{0,N} \end{bmatrix} \tag{7.4}$$

$$\vec{x}_{n+1} = Optim\{\vec{x}_n\} \tag{7.5}$$

O código termina sua execução uma vez que um custo máximo aceitável é encontrado, a função de custo encontra um ponto de inflexão, ou outros critérios de parada são atingidos como número máximo de iterações do algoritmo ou tempo limite de execução. Ao final, é retornado um vetor $\vec{x}_f$ contendo o valor otimizado de cada uma das variáveis de otimização. Um valor máximo aceitável da função de custo é um valor um tanto arbitrário, pois depende muito de como a função de custo é definida e pode ser influenciado pelo tamanho do domínio de integração e número de funções e variáveis que participam da função de custo. Para as aplicações deste trabalho, após realizados vários testes, foi determinado que em geral uma função de custo com valor inferior a 200 significa uma solução ótima, enquanto soluções com custos inferiores a 800 são muito boas e inferiores a 3000 são aceitáveis. Entretanto, estes valores foram estabelecidos apenas para as simulações deste trabalho.

7.3 Otimizações realizadas

O algoritmo de otimização utilizado neste trabalho foi implementado utilizando o módulo *scipy.optimize.minimize* em *Python*. O argumento desta função é o resultado de uma função de custo calculada a partir das distribuições de radiação do arranjo alvo e objetivo. A função de custo neste caso quantifica a diferença entre as distribuições de radiação dos dois arranjos, e custo zero é obtido quando ambos arranjos tem a mesma distribuição, ou seja, quando o arranjo a ser optimizado produz uma distribuição de radiação idêntica à distribuição de radiação desejada.

Para a otimização de um arranjo de antenas, as possíveis variáveis parametrizadas são a posição, orientação e corrente de alimentação de cada antena que compõe o arranjo, somando 8 variáveis para cada antena (3 variáveis posição, 3 variáveis de orientação, magnitude e fase de corrente). Alguns algoritmos de otimização funcionam melhor quando as variáveis parametrizadas são contidas no intervalo entre 0 e 1, assim, quando possível, pode ser interessante limitar as variáveis parametrizadas do algoritmo de otimização dentro de um intervalo fixo, pois isso pode agilizar o processo de otimização. Dentre as variáveis parametrizadas de antenas, os valores dos ângulos de orientação e fase são na-

turalmente limitadas conforme ilustrado nas Equações 7.6, 7.7, 7.8 e 7.9. Dados intervalos fíxos, é possível mapear os valores para que fiquem entre 0 e 1. As variáveis de posição e magnitude da corrente das antenas não são limitadas naturalmente, mas, dependendo da otimização, também podem ser mapeadas em um intervalo entre 0 e 1 quando divididas por um valor máximo esperado, que fica a critério do projetista. Desta forma, cada variável parametrizada de posição, orientação e corrente de alimentação é mapeada para um elemento do vetor de parâmetros $\vec{x}$ na otimização. Desta forma, a função de custo fica dependente apenas do vetor de parâmetros $\vec{x}$, e o problema de otimização se torna na verdade um problema de minimização da função de custo, ou seja, encontrar qual o conjunto de variáveis parametrizadas $\vec{x}$ que minimizam a função de custo. Claro, nem todas as possíveis variáveis otimizáveis tem que participar da otimização. Fica a critério do projetista determinar quais variáveis tem maior efeito na função de custo para a aplicação desejada, e então quais variáveis devem ser otimizadas. Cada variável parametrizada em cada antena adiciona uma nova dimensão no problema de otimização, aumentando polinialmente a complexidade e o tempo necessário para o término do algoritmo.

$$-\pi \leq \beta \leq \pi \tag{7.6}$$

$$-\pi/2 \leq \alpha \leq \pi/2 \tag{7.7}$$

$$-\pi \leq \beta \leq \pi \tag{7.8}$$

$$-\pi \leq \angle I \leq \pi \tag{7.9}$$

Neste trabalho, uma possível função de custo é definida como a integral do módulo da diferença entre o vetor de radiação resultantes do arranjo e o vetor de radiação desejado préviamente definido, como ilustrada na Equação 7.10 da definição 3 da função de custo, onde S é o domínio de ângulos sólidos formado pelos ângulos ϕ e θ de interesse, possivelmente integrando em todos os ângulos ϕ e θ possíveis, ou seja, sobre a superfície de uma esfera de raio unitário formada pelos ângulos θ de 0 a π e ϕ de $-\pi$ a π. Esta definição de custo leva em consideração também a fase entre as componentes do vetor de radiação, ou seja, a fase entre as componentes ϕ e θ do vetor. Em alguns casos, como otimizações de arranjos de polarização linear, a fase não é de grande interesse, e sim a magnitude da radiação irradiada em uma dada direção, assim, é utilizada a função de custo dada pela Equação 7.11 da definição 4 da função de custo, onde é levada em consideração apenas o módulo do vetor de radiação. Esta definição pode acelerar o processo de otimização aceitando arranjos cuja magnitude do vetor de radiação se assemelha à magnitude do vetor de radiação desejado enquanto sua fase não necessariamente é parecida. A Equação 7.10 pode ser melhor utilizada ao trabalhar com arranjos de polarização circular, onde a fase das componentes do vetor de radiação são importantes. Nestas equações, F denota o vetor de radiação do arranjo para uma configuração de antenas dada pelo vetor de parâmetros

$\vec{x}$ e F^t denota o vetor de radiação desejado ao final da otimização, ou seja, o alvo (*target*) da otimização.

$$Cost_3(\vec{x}) = \int_S |F^t(\theta,\phi) - F(\theta,\phi,\vec{x})|^2 d\theta d\phi \tag{7.10}$$

$$Cost_4(\vec{x}) = \int_S (|F^t(\theta,\phi)| - |F(\theta,\phi,\vec{x})|)^2 d\theta d\phi \tag{7.11}$$

Considerando que a integração se dá sobre uma superfície esférica, é necessário levar em consideração que os pontos de amostragem próximos dos pólos representam uma área menor em relação aos pontos de amostragem próximos ao equador, pois em uma malha regular dos ângulos θ e ϕ os pontos próximos aos pólos estão posicionados mais próximos entre si. Este efeito é compensado introduzindo na integral um fator de $\sin(\theta)$, resultando na definição 5 da função de custo 7.12.

$$Cost_5(\vec{x}) = \int_S (|F^t(\theta,\phi)| - |F(\theta,\phi,\vec{x})|)^2 \sin(\theta) d\theta d\phi \tag{7.12}$$

Finalmente, nem todas as direções de apontamento precisam ter a mesma importantes para a otimização, desta forma, pode ser interessante acrescentar uma função de peso $W(\theta,\phi)$ em função dos ângulos de apontamento para que a otimização seja focalizada na minimização do erro com mais ênfase em certas direções que em outras. Com esta função de peso, a equação de custo se torna 7.13, que é uma definição completa para a função de custo. A função de peso é definida pelo projetista de forma a ajustar o custo às necessidades e prioridades do sistema sendo desenvolvido. No escopo deste trabalho, entretanto, considerando que nas aplicações deste trabalho a direção de apontamento mais importante seja com $\theta = 0°$, pois o sistema é para comunicações com satélites que normalmente se econtram acima do arrajo de antenas em solo, é interessante que as amostragens nesta direção sejam preferenciadas. As direções que apontam para o solo não são de interesse para o arranjo, e a radiação emitida em direção ao solo pode ser ignorada, assim, pode-se definir uma função de peso de forma que $W(\theta,\phi) = 0, \theta > 90°$. De fato, as direções de apontamento que apontam para o solo podem ser completamente retiradas da equação do custo, o que diminui pela metade a superfície de integração e acelera o cálculo da função de custo de uma iteração. Levando isto em consideração, a definição da função de custo em 7.11 pode ainda ser interessante, pois naturalmente privilegia as direções de apontamento para cima e economiza na quantidade de computações a serem efetuadas durante uma iteração. Mesmo assim, neste trabalho foram utilizadas definições de custo baseada em 7.12, levando em consideração o jacobiano da superfície esférica, e um peso foi aplicado ao desconsiderar na superfície de integração pontos com ângulo $\theta > 90°$.

$$Cost_6(\vec{x}) = \int_S W(\theta,\phi)(|F^t(\theta,\phi)| - |F(\theta,\phi,\vec{x})|)^2 \sin(\theta) d\theta d\phi \tag{7.13}$$

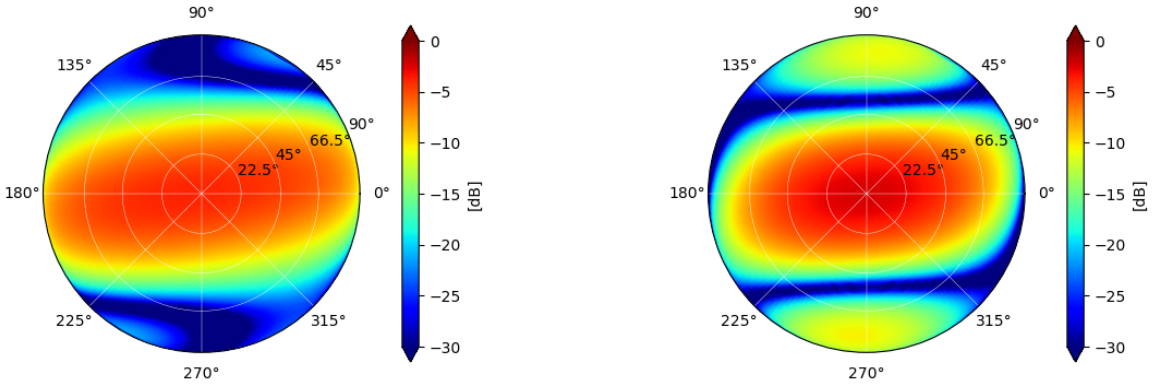
Um custo zero na prática é impossível de ser obtido em aplicações reais, desta forma, a função de minimização tem o objetivo de determinar parâmetros que minimizam a função de custo, assim forçando com que a distribuição de radiação do arranjo optimizado seja o mais semelhante possível com a distribuição de radiação desejada, e diversos algoritmos disponíveis nas bibliotecas padrões de *Python* podem ser utilizados para este fim. Como o objetivo deste trabalho não é realizar longas otimizações para um problema específico, e sim demonstrar os algoritmos desenvolvidos e suas aplicações, as otimizações realizadas neste trabalho utilizaram o algoritmo *BFGS*, por sua simplicidade, praticidade, e por ser um algoritmo padrão para otimizações simples.

As seções seguintes ilustram os resultados de optimizações teste simples realizadas com o objetivo de construir distribuições de radiações arbitrárias utilizando as antenas Dipolo e Yagi-Uda desenvolvidas neste trabalho. Desta forma foi avaliada a capacidade do algoritmo de otimizar arranjos arbitrários de antenas, assim como o tempo decorrido e o desempenho dos resultados obtidos.

7.3.1 Arranjos de polarização linear

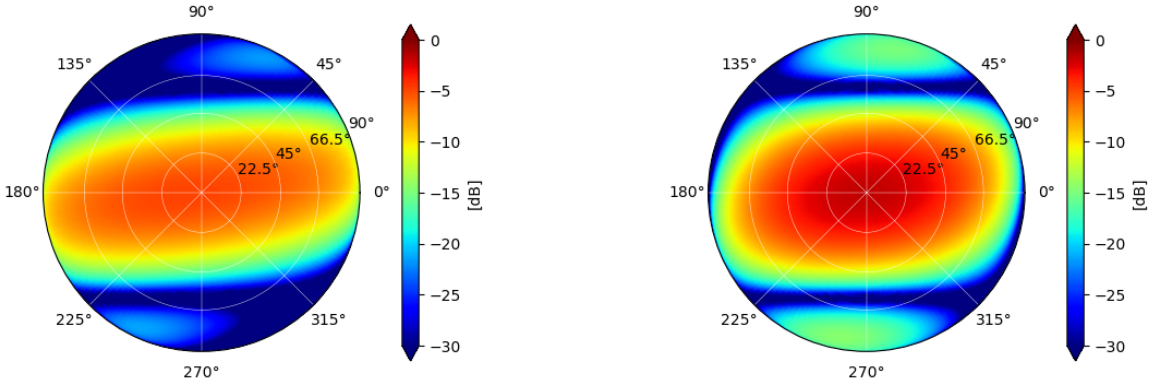
A seguinte otimização foi realizada em um arranjo com 3 antenas Yagis de 2 elementos. Duas cópias do mesmo arranjo foram criadas, com alimentação e posição iguais. A orientação das antenas de um dos arranjos foi então modificada, de forma que o campo elétrico resultante do arranjo se tornou ligeiramente diferente do campo elétrico do outro arranjo. Um dos dois arranjos foi denominado arranjo alvo, enquanto o outro será o arranjo que será otimizado. Assim, esta otimização serviu para averiguar a capacidade de encontrar um mínimo local da função de custo de um problema simples. Neste caso, isso pode ser feito realinhando as antenas do arranjo para que os dois arranjos se tornem novamente idênticos. A função de custo utilizada é dada na Equação 7.14, neste caso utilizando como domínio apenas a semiesfera superior do domínio de integração, visto que neste caso as direções de apontamento com ângulos $\theta > \frac{\pi}{2}$ apontam para o chão e podem ser ignoradas. Isto permite diminuir pela metade o domínio de integração, agilizando o processo de otimização, sem comprometer o resultado final. Obviamente, a função de custo desta otimização pode ser zero se as orientações das antenas de um arranjo forem idênticas às orientações das antenas do arranjo alvo. Como os arranjos neste problema são muito similares, a solução ótima se encontra bem próxima das condições iniciais da otimização.

$$Cost_{linear} = \int_{-\pi}^{\pi} \int_{0}^{\frac{\pi}{2}} \left(\left| |F_{ref}| - |F^t_{ref}| \right| + \left| |F_{cross}| - |F^t_{cross}| \right| \right) \sin(\theta) d\theta d\phi \tag{7.14}$$



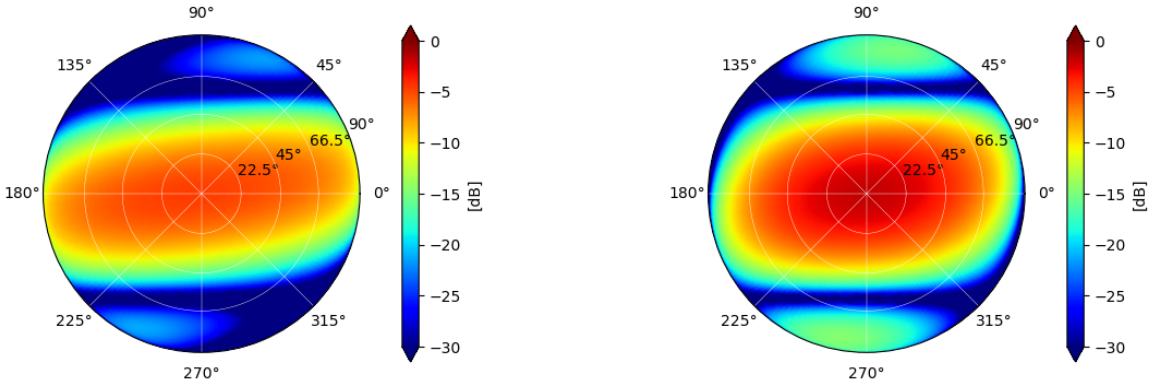
(a) Polarização Linear Referencial (b) Polarização Linear Cruzada

Figura 48 – Arranjo antes da otimização



(a) Polarização Linear Referencial (b) Polarização Linear Cruzada

Figura 49 – Campo elétrico alvo da otimização



(a) Polarização Linear Referencial (b) Polarização Linear Cruzada

Figura 50 – Arranjo após a otimização

As Figuras 48, 49, e 50 apresentam respectivamente o campo elétrico arranjo inicial

da otimização, o campo elétrico alvo e o campo elétrico do arranjo após a otimização. De fato, como é visível nas figuras, a otimização foi capaz de alinhar as antenas dos arranjos, minimizando a função de custo. Assim, a otimização obteve sucesso em encontrar o mínimo local mais perto da função de custo. Com as antenas alinhadas, o campo elétrico dos dois arranjos se tornou idêntico em âmbas polarizações lineares, e assim a função de custo final foi próxima de zero.

As Tabelas 7, 8, e 9 contém a posição, orientação e alimentação de todas as antenas dos arranjos da otimização. É observado na posição e orientação das antenas do arranjo otimizado que estas não são exatamente iguais às posições e orientações do arranjo alvo

	Posição			Orientação			Alimentação	
Antenna	X	Y	Z	Elevation [o]	Azimuth [o]	Roll [o]	Magnitude	Phase [o]
Yagi 2 elementos	0.00	0.00	0.00	-90.00	60.00	0.00	1.00	0.00
Yagi 2 elementos	0.00	0.50	0.00	-90.00	30.00	0.00	1.00	0.00
Yagi 2 elementos	0.00	1.00	0.00	-90.00	60.00	0.00	1.00	0.00

Tabela 7 – Arranjo inicial

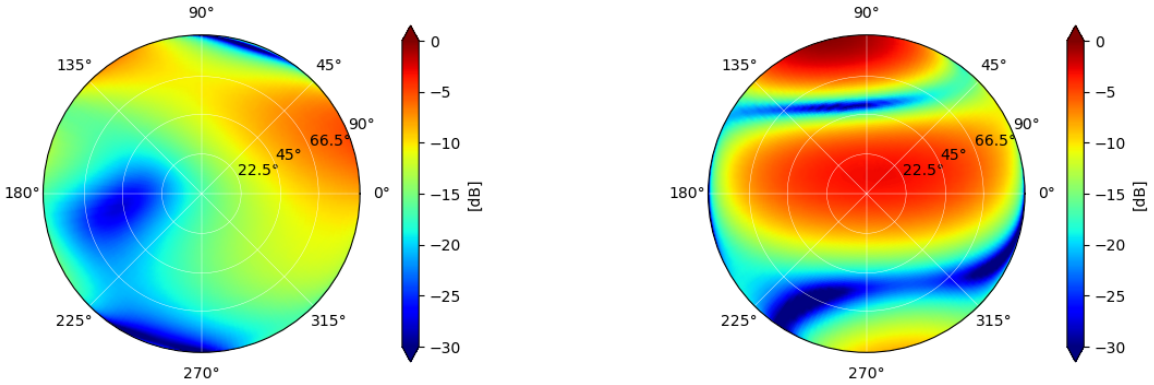
	Posição			Orientação			Alimentação	
Antenna	X	Y	Z	Elevation [o]	Azimuth [o]	Roll [o]	Magnitude	Phase [o]
Yagi 2 elementos	0.00	0.00	0.00	-90.00	55.00	0.00	1.00	0.00
Yagi 2 elementos	0.00	0.50	0.00	-90.00	55.00	0.00	1.00	0.00
Yagi 2 elementos	0.00	1.00	0.00	-90.00	55.00	0.00	1.00	0.00

Tabela 8 – Arranjo alvo

	Posição			Orientação			Alimentação	
Antenna	X	Y	Z	Elevation [o]	Azimuth [o]	Roll [o]	Magnitude	Phase [o]
Yagi 2 elementos	0.00	0.00	0.00	-85.71	54.56	0.00	1.00	0.00
Yagi 2 elementos	0.00	0.50	0.00	-86.12	54.19	0.00	1.00	0.00
Yagi 2 elementos	0.00	1.00	0.00	-90.00	53.93	0.00	1.00	0.00

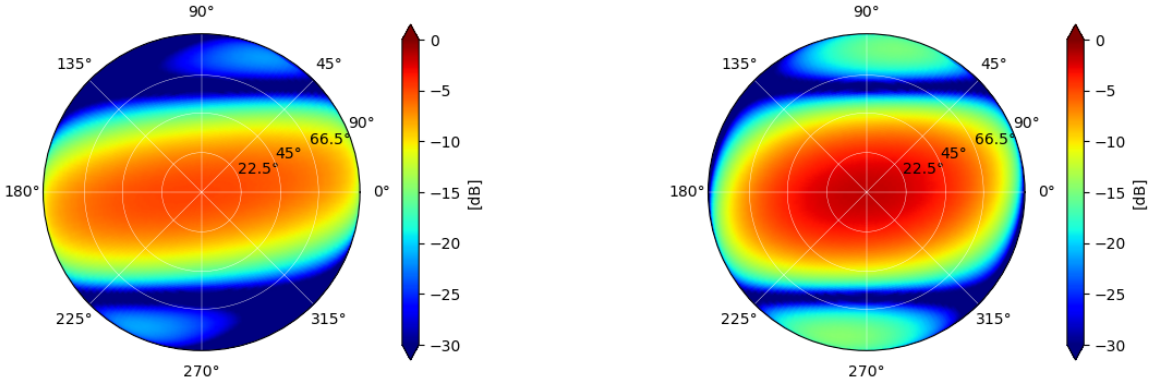
Tabela 9 – Resultados da otimização de polarização linear com custo final de 73.18

Para demonstrar o efeito que as condições iniciais tem sobre a convergência da otimização para uma boa solução, foi realizada uma nova otimização com os mesmos arranjos utilizados na otimização anterior. Entretanto, a orientação do arranjo a ser otimizado foi modificada de forma a não favorecer a convergência para o arranjo alvo. Como pode ser visualizado nas Figuras 48, 49, e 50, embora a otimização convergiu para um mínimo local resultando em um campo elétrico parecido com o campo elétrico alvo, este claramente não é o mínimo global ótimo atingido na otimização anterior. As Tabelas 10, 11, 12 contém os parâmetros dos arranjos desta otimização, e é observado que o custo obtido nesta otimização é muito superior ao custo obtido anteriormente.



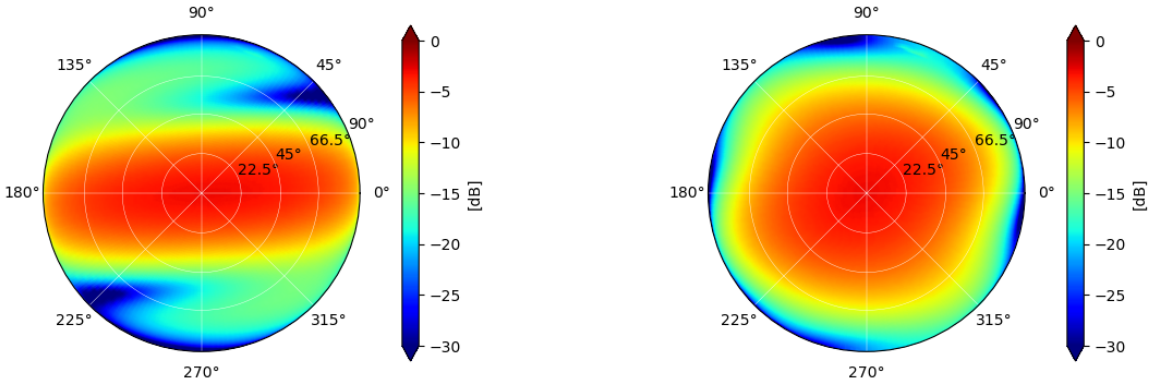
(a) Polarização Linear Referencial (b) Polarização Linear Cruzada

Figura 51 – Arranjo antes da otimização



(a) Polarização Linear Referencial (b) Polarização Linear Cruzada

Figura 52 – Campo elétrico alvo da otimização



(a) Polarização Linear Referencial (b) Polarização Linear Cruzada

Figura 53 – Arranjo após a otimização

	Posição			Orientação			Alimentação	
Antenna	X	Y	Z	Elevation [o]	Azimuth [o]	Roll [o]	Magnitude	Phase [o]
Yagi 2 elementos	0.00	0.00	0.00	0.00	90.00	0.00	1.00	0.00
Yagi 2 elementos	0.00	0.50	0.00	0.00	30.00	0.00	1.00	0.00
Yagi 2 elementos	0.00	1.00	0.00	0.00	110.00	0.00	1.00	0.00

Tabela 10 – Arranjo inicial

	Posição			Orientação			Alimentação	
Antenna	X	Y	Z	Elevation [o]	Azimuth [o]	Roll [o]	Magnitude	Phase [o]
Yagi 2 elementos	0.00	0.00	0.00	-90.00	55.00	0.00	1.00	0.00
Yagi 2 elementos	0.00	0.50	0.00	-90.00	55.00	0.00	1.00	0.00
Yagi 2 elementos	0.00	1.00	0.00	-90.00	55.00	0.00	1.00	0.00

Tabela 11 – Arranjo alvo

	Posição			Orientação			Alimentação	
Antenna	X	Y	Z	Elevation [o]	Azimuth [o]	Roll [o]	Magnitude	Phase [o]
Yagi 2 elementos	0.00	0.00	0.00	-86.93	180.00	0.00	1.00	0.00
Yagi 2 elementos	0.00	0.50	0.00	-74.56	-109.09	0.00	1.00	0.00
Yagi 2 elementos	0.00	1.00	0.00	-90.00	-119.11	0.00	1.00	0.00

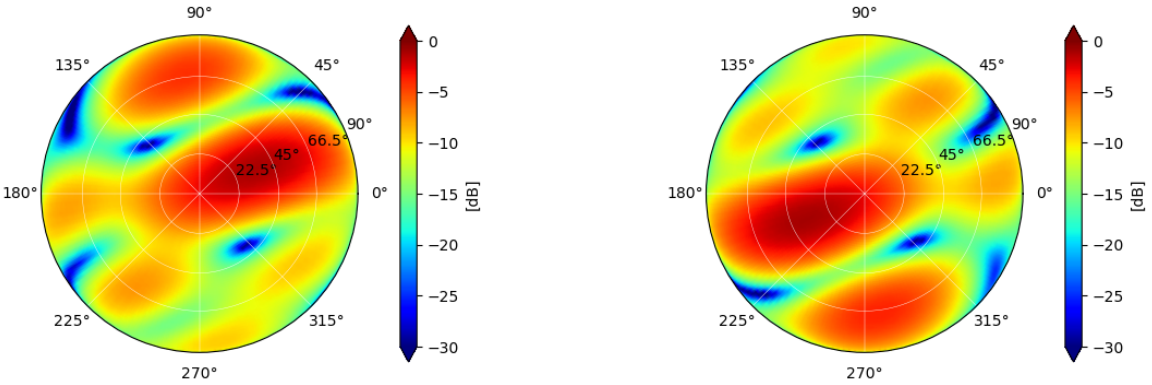
Tabela 12 – Resultados da otimização de polarização linear com custo final de 1168.11

7.3.2 Arranjos de polarização circular

Outra otimização de muito interesse é a das polarizações circulares dos arranjos. Esta otimização tem como objetivo mostrar que, tal como com as polarizações lineares, é possível modelar os arranjos com polarizações circulares. Neste caso, os arranjos da otimização com polarização linear foram reorientados e a fase de suas alimentações foi modificada para produzirem campos elétricos de polarizações circulares. A orientação das antenas novamente difere levemente de um arranjo para outro, produzindo distribuições de campo elétrico ligeiramente distintas. A otimização dos arranjos alinhou as orientações das antenas dos dois arranjos, resultando em distribuições de campo elétrico de polarização circular idênticos, minimizando a função de custo.

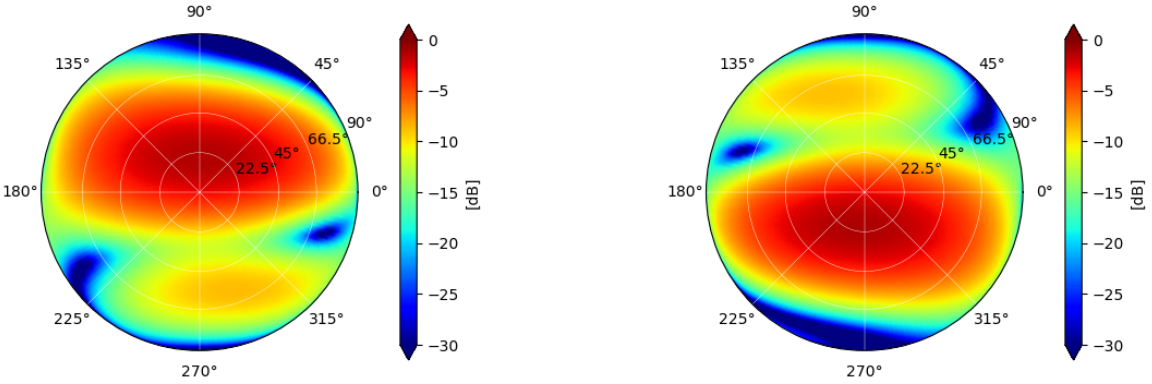
A função de custo desta otimização deve então levar em consideração os campos elétricos de polarização circular, para tanto, a função de custo utilizada nesta otimização é dada pela Equação 7.15 e os resultados são ilustrados nas Figuras 54, 55, e 56. A função de custo considera a diferença dos módulos dos campos elétricos pois a direfença em fase não é de interesse neste problema. Este é um fator importante a ser levado em consideração nas otimizações, pois pode aumentar o número de mínimos locais e simplificar a otimização. A função de custo dada pela Equação 7.15 é adequada para polarizações lineares pois leva diretamente em conta o vetor de radiação de polarização circular do arranjo, que é fornecido por padrão pelo código sempre que é solicitado calcular o campo elétrico resultante de um arranjo.

$$Cost_{circular} = \int_{-\pi}^{\pi} \int_{0}^{\frac{\pi}{2}} \left(\left| |F_{rhcp}| - |F^t_{rhcp}| \right| + \left| |F_{lhcp}| - |F^t_{lhcp}| \right| \right) \sin(\theta) d\theta d\phi \tag{7.15}$$



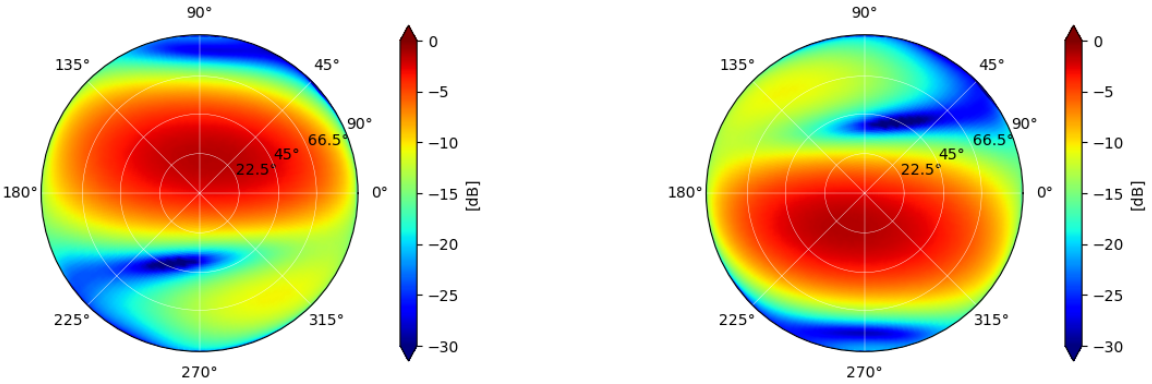
(a) Polarização circular direita (b) Polarização circular esquerda

Figura 54 – Arranjo antes da otimização



(a) Polarização circular direita (b) Polarização circular esquerda

Figura 55 – Campo elétrico alvo da otimização



(a) Polarização circular direita (b) Polarização circular esquerda

Figura 56 – Arranjo após a otimização

As Figuras 54, 55, e 56 apresentam respectivamente o campo elétrico arranjo inicial da otimização, o campo elétrico alvo e o campo elétrico do arranjo após a otimização.

Destas otimizações, conclui-se que o processo de otimização consegue de fato encontrar soluções para problemas simples, dado que as condições iniciais do problema se encontram perto de uma solução adequada. Os casos simulados tem soluções óbvias onde é possível chegar em uma função de custo igual a zero. Problemas encontrados na realidade não tem na prática uma solução ideal possibilitando uma função de custo zero, desta forma, estes problemas tem como objetivo apenas minimizar a função de custo, produzindo campos elétricos resultantes que sejam aceitáveis para a aplicação em mente.

	Posição			Orientação			Alimentação	
Antenna	X	Y	Z	Elevation [o]	Azimuth [o]	Roll [o]	Magnitude	Phase [o]
Yagi 2 elementos	0.00	0.00	0.00	-90.00	0.00	0.00	1.00	0.00
Yagi 2 elementos	0.30	0.30	0.00	-90.00	120.00	0.00	1.00	0.00
Yagi 2 elementos	-0.30	1.30	0.00	-90.00	60.00	0.00	1.00	0.00

Tabela 13 – Arranjo inicial

	Posição			Orientação			Alimentação	
Antenna	X	Y	Z	Elevation [o]	Azimuth [o]	Roll [o]	Magnitude	Phase [o]
Yagi 2 elementos	0.00	0.00	0.00	-90.00	0.00	0.00	1.00	0.00
Yagi 2 elementos	0.00	0.50	0.00	-90.00	90.00	0.00	1.00	0.00
Yagi 2 elementos	0.00	1.00	0.00	-90.00	90.00	0.00	1.00	0.00

Tabela 14 – Arranjo alvo

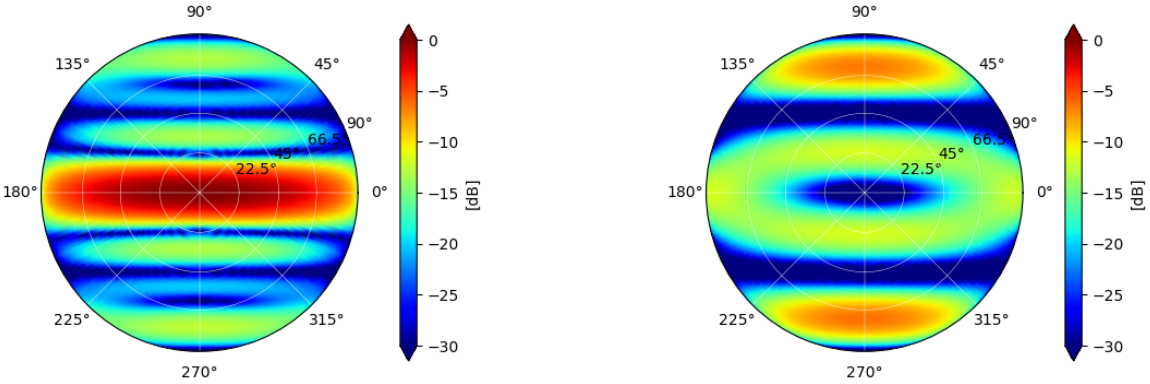
	Posição			Orientação			Alimentação	
Antenna	X	Y	Z	Elevation [o]	Azimuth [o]	Roll [o]	Magnitude	Phase [o]
Yagi 2 elementos	0.00	0.00	0.00	-90.00	0.00	0.00	1.00	0.00
Yagi 2 elementos	0.22	0.39	0.00	-90.00	120.00	-80.83	1.00	0.00
Yagi 2 elementos	0.08	0.95	0.00	-90.00	60.00	30.48	1.00	0.00

Tabela 15 – Resultado da otimização de polarização circular com custo final de 535.06

7.3.3 Otimização com arranjo grande e um objetivo arbitrário

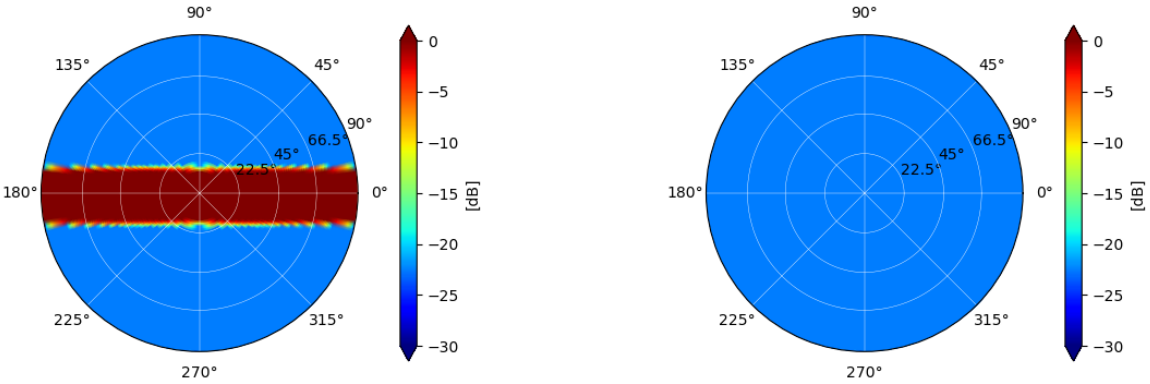
Para testar a capacidade de otimização do código para um problema mais prático, foi realizada uma otimização utilizando como objetivo uma antena cujos campos elétricos são calculados a partir de uma função arbitrária. O campo elétrico produzido por esta antena não vem de uma antena real pois sua radiação é modelada matematicamente, assim, não é esperado que a otimização seja capaz de reproduzir fielmente o campo elétrico objetivo, mas uma aproximação deste é possível quando as variáveis otimizadas possibilitam um alto grau de liberdade de modificação do campo elétrico resultante. É estipulado que, conforme o número de antenas do arranjo a ser otimizado aumente, seja possível reproduzir cada vez mais fielmente o campo elétrico objetivo arbitrário, em uma relação análoga à reconstrução de uma função arbitrária a partir de um somatório ponderado dado por uma série de Fourier truncada, onde, conforme mais termos são adicionados, o somatório resultante se aproxima da função desejada.

Desta forma, o seguinte teste foi realizado utilizando um arranjo linear simétrico com 8 antenas Yagi de 2 elementos de polarização linear. Todas as antenas foram posicionadas com espaçamento $0.5\lambda_0$ entre si na direção $\hat{y}$, orientadas com elevação de $-90°$, com ângulos de *azimuth*, *roll* e fase da corrente de alimentação variando de forma a produzir um campo elétrico resultante de polarização circular direita. A variável otimizada foi a magnitude da alimentação e o azimuth para cada par de antena, mantendo simetria no arranjo, totalizando 8 variáveis parametrizadas. A função de custo utilizada neste problema foi a Equação 7.15. O custo inicial da otimização, com o arranjo inicial, foi de 1700.39, e o custo final foi de 1559.88.



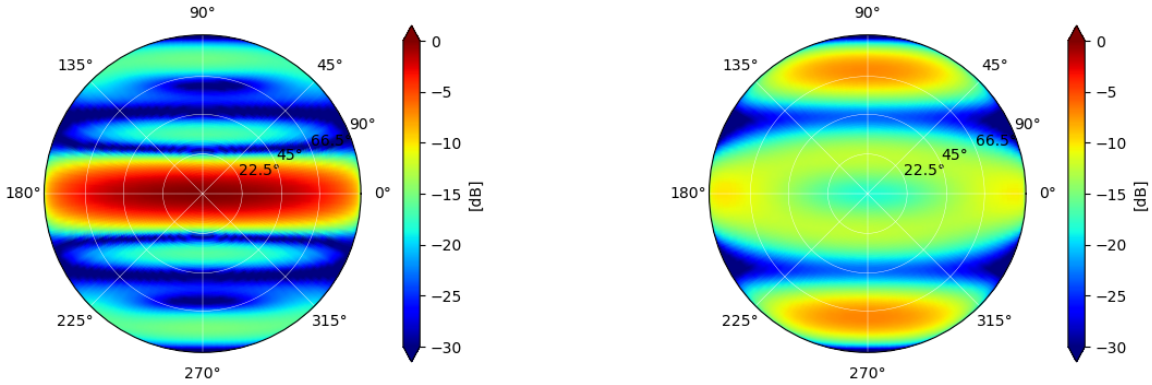
(a) Polarização circular direita (b) Polarização circular direita

Figura 57 – Arranjo antes da otimização



(a) Polarização circular direita (b) Polarização circular direita

Figura 58 – Campo elétrico alvo da otimização



(a) Polarização circular direita (b) Polarização circular direita

Figura 59 – Arranjo após a otimização

As Figuras 57 a 59 demonstram o resultado da otimização do estado inicial do arranjo até o arranjo final. Das figuras, conclui-se que o campo elétrico resultante do arranjo após a otimização é mais similar ao campo elétrico alvo. Menos energia é liberada nos ângulos sólidos com baixa energia no campo elétrico alvo, enquanto mais energia é liberada nas regiões de alta energia. Desta forma, a função de custo apresenta um mínimo local, e por isso a otimização convergiu para esta configuração das antenas. O arranjo resultante ainda apresenta um pico alto de energia na faixa horizontal, alinhada à faixa do campo alvo, enquanto o desejado seria uma distribuição de energia mais constante.

A otimização foi capaz de focalizar a energia do arranjo para a faixa desejada, entretanto, se faz necessário um estudo para descobrir condições iniciais que providenciem uma distribuição mais constante, conforme desejado, e assim um custo menor. A adição de novas antenas ao arranjo na configuração atual de otimização tende a exaltar o pico de energia na faixa de frequências, o que não é desejado. A semelhança dos resultados com uma somatória de senos e cossenos não é coincidência. De fato, o fator de arranjo das antenas em um arranjo linear regular se torna uma série de funções sinc, assim, é indicado que a solução ideal do problema esteja relacionada com um somatório ponderado de funções sinc neste problema.

	Posição			Orientação			Alimentação	
Antenna	X	Y	Z	Elevation [o]	Azimuth [o]	Roll [o]	Magnitude	Phase [o]
Yagi 2 elementos	0.00	0.25	0.00	-90.00	0.00	0.00	1.00	0.00
Yagi 2 elementos	0.00	0.75	0.00	-90.00	0.00	90.00	1.00	90.00
Yagi 2 elementos	0.00	1.25	0.00	-90.00	0.00	0.00	1.00	0.00
Yagi 2 elementos	0.00	1.75	0.00	-90.00	0.00	90.00	1.00	90.00
Yagi 2 elementos	0.00	-0.25	0.00	-90.00	0.00	0.00	1.00	0.00
Yagi 2 elementos	0.00	-0.75	0.00	-90.00	0.00	90.00	1.00	90.00
Yagi 2 elementos	0.00	-1.25	0.00	-90.00	0.00	0.00	1.00	0.00
Yagi 2 elementos	0.00	-1.75	0.00	-90.00	0.00	90.00	1.00	90.00

Tabela 16 – Arranjo inicial

	Posição			Orientação			Alimentação	
Antenna	X	Y	Z	Elevation [◦]	Azimuth [◦]	Roll [◦]	Magnitude	Phase [◦]
Yagi 2 elementos	0.00	0.25	0.00	-90.00	0.00	0.00	1.54	0.00
Yagi 2 elementos	0.00	0.75	0.00	-90.00	0.00	90.00	1.30	90.00
Yagi 2 elementos	0.00	1.25	0.00	-90.00	0.00	0.00	1.33	0.00
Yagi 2 elementos	0.00	1.75	0.00	-90.00	0.00	90.00	0.94	90.00
Yagi 2 elementos	0.00	-0.25	0.00	-90.00	0.00	0.00	1.54	0.00
Yagi 2 elementos	0.00	-0.75	0.00	-90.00	0.00	90.00	1.30	90.00
Yagi 2 elementos	0.00	-1.25	0.00	-90.00	0.00	0.00	1.33	0.00
Yagi 2 elementos	0.00	-1.75	0.00	-90.00	0.00	90.00	0.94	90.00

Tabela 17 – Resultados da otimização de polarização linear com custo final de 1645.46

7.4 Considerações sobre otimizações numéricas

As simulações e otimizações realizadas deixam clara uma grande limitação do código: a enorme quantidade de possibilidades demanda uma enorme quantidade de simulações e tempo para encontrar a solução ótima para um problema qualquer. Este problema é exaltado na otimização com 30 antenas e um campo elétrico alvo definido matematicamente. A adição de cada antena aumenta a complexidade do problema, o campo elétrico gerado por uma antena afeta o campo elétrico resultante em todas as direções de forma periódica, contribuindo para a solução ótima em múltiplas direções e prejudicando em múltiplas outras direções.

Apesar da velocidade do algoritmo, as otimizações necessitam de tantas simulações que uma otimização com arranjos grandes e muitas variáveis parametrizadas pode demorar consideravelmente. Desta forma, se faz necessário utilizar algoritmos de otimização mais adequados para o problema a ser otimizado, escolhendo de forma mais eficiente as variações do arranjo a serem simuladas. Uma escolha inteligente do algoritmo de otimização pode diminuir consideravelmente o número de simulações a serem efetuadas, acelerando o processo de otimização e, possivelmente, melhorando o resultado final encontrando um mínimo local menor da função de custo. A escolha das condições iniciais da otimização também afeta o resultado final, principalmente quando algoritmos de otimização baseados no gradiente são utilizados, pois diferentes escolhas de condições iniciais vão convergir em diferentes mínimos locais da função de custo.

Para contornar este problema, diversas otimizações podem ser realizadas escolhendo para cada uma um conjunto de condições iniciais distinto, possivelmente escolhido aleatóriamente. Entretanto, conforme o número de variáveis parametrizadas aumenta, o número de condições iniciais necessárias para se representar o espaço de possibilidades total aumenta exponencialmente, rapidamente se tornando inviável simular uma porção significativa das possibilidades. Sendo assim, a qualidade da solução encontrada para a maioria dos problemas práticos depende da escolha inteligente das condições iniciais da otimização, garantindo que estas estejam razoavelmente próximas do mínimo global ou de um mínimo local que seja suficiente para a aplicação.

A busca pelo mínimo global da função de custo por meio de algoritmos de otimização sempre será computacionalmente custosa, e não há garantia de que qualquer mínimo local encontrado seja o mínimo global ou que haja outra solução mais otimizada a não ser nos problemas mais simples. Desta forma, aumentar o número de variáveis parametrizadas e o número de antenas da otimização nem sempre se mostra a estratégia mais adequada para resolver um problema. Em verdade muitas vezes isto pode dificultar a resolução do problema aumentando a complexidade, demanda computacional e tempo, sem necessidade. Em vez disso, a melhor estratégia é avaliar cada problema e escolher quais variáveis parametrizáveis serão mais influentes na equação de custo e quais condições iniciais irão garantir uma convergência mais rápida para um mínimo local aceitável. Estas análises devem ser realizadas individualmente para cada problema, e cada problema irá demandar uma aproximação diferente.

Outro aspecto importante a ser considerado é que o cálculo do campo elétrico resultante de arranjos por meio do fator de arranjo se torna impreciso quando a distância entre as antenas é menor que seu comprimento característico. Em outras palavras, quando existem elementos metálicos dentro da região de campo próximo de uma antena, a disribuição de radiação desta antena é modificado. Este artefato geométrico não é considerado no cálculo executado pelos algoritmos desenvolvidos. Visto que as variáveis otimizáveis são, em geral, posição e orientação das antenas, deve-se ter cuidado para restringir a posição das antenas de um arranjo em relação às suas vizinhas durante a otimização, de forma a evitar que as antenas entrem na região de campo próximo de suas vizinhas, resultando em uma distribuição de radiação diferente da observada na realidade.

Isto pode ser feito efetuando uma rápida verificação da geometria do problema a ser simulado. O comprimento característico de cada antena pode ser dado como um parâmetro e, antes de cada simulação, o algoritmo calcula a distância entre cada par de antenas e verifica que a distância não é menor que a soma de seus comprimentos característicos. Desta forma, é garantida que não são realizadas simulações cujo resultado não é utilizável. O processo a ser executado quando duas antenas estão muito próximas pode ser definido arbitráriamente, parando a otimização ou apenas afastando as antenas até que todos os requisitos para prosseguir com a otimização sejam atendidos.

8 Considerações Finais

8.1 Códigos nos apêndices

Durante a criação deste documento, os *softwares* ainda estão em desenvolvimento, assim, a versão mais recente pode ser encontrada no repositório de *GitHub* disponível em VitinLima/EletromagArrays. Os trabalhos futuros a serem realizados serão refletidos no repositório online, mesmo assim é disponibilizado a versão atual do código nos apêndices para garantir a disponibilidade dos códigos independente do estado do repositório. Estão também no repositório do *GitHub* os arquivos .csv exportados nas simulações do *ANSYS Electronics*. Para geração destes arquivos, é realizada uma simulação eletromagnética da antena de interesse e seus campos são exportados em arquivos .csv conforme mostrado na Metodologia deste trabalho. O arquivo .csv é então importado para os códigos em *Octave* e *Python* por meio de funções criadas para interpretar estes arquivos específicos, e a partir dos campos elétricos importados os modelos das antenas são criados para serem utilizados. Por fim, o código desenvolvido está sendo protegido pelo Instituto Nacional da Propriedade Industrial (INPI).

8.2 Trabalhos Futuros

Simulações futuras devem modelar presilhas e parafusos a serem utilizados na fabricação para aferir seus efeitos na propriedades das antenas. É esperado que estes não afetem significativamente as propriedades, mas caso isto aconteça, uma otimização do *Balun* deve ser suficiente para recuperar a performance.

Simulações de sensibilidade devem ser realizadas para aferir os efeitos dos defeitos de fabricação dos métodos disponíveis e garantir que os defeitos não alterem as propriedades da antena significativamente. Caso isto aconteça, outros processos de fabricação devem ser escolhidos para garantir a precisão mínima requerida pelas simulações de sensibilidade.

Após as simulações de sensibilidade e geometria detalhada das antenas e os arranjos de antenas serem projetados, as antenas a serem utilizadas serão fabricadas e suas propriedades medidas experimentalmente. Espera-se uma queda na performance das antenas reais quando comparadas com as antenas projetadas, pois existem defeitos de fabricação e a precisão dos métodos de fabricação disponíveis é limitada. Assim, deve ser garantido que o projeto das antenas tenha performance superior à necessária para o correto funcionamento do arranjo de antenas, de modo a ter um fator de segurança adequado.

O próximo passo do projeto será desenvolver uma família de antenas do tipo dipolo dobrado no HFSS, capaz de ser utilizado em uma banda de frequências mais ampla. O dipolo dobrado tem uma banda de operação maior do que o dipolo simples, e então é mais interessante para o desenvolvimento de um arranjo de antenas, podendo atender a uma gama maior de sistemas de comunicação sem aumentar consideravelmente a complexidade e custo do sistema.

A família de dipolos dobrados a ser desenvolvida será então utilizada na construção de um arranjo de antenas. Este arranjo será otimizado utilizando algoritmos a serem desenvolvidos e, por fim, as antenas serão fabricadas e testadas, validando a simulações numéricas no *HFSS*, e dando início à fabricação do arranjo de antenas.

No caso da comunicação do solo com satélites, as ondas eletromagnéticas eventualmente atravessam uma região da atmosfera com alta concentração de ions e partículas carregadas pela energia solar. Tal região é denominada Ionosfera, e tem o efeito de rotacionar a polarização da onda que a atravessa. Em ondas com polarização linear, o resultado é uma troca parcial ou total de polarização referencial para cruzada e vice-versa. Para recuperar a informação transmitida, é necessário ajustar a orientação das antenas transmissoras e receptoras a fim de corrigir a polarização, adicionando partes móveis ao sistema o que gera novos pontos de risco para o projeto, ou captar ambas polarizações e reconstruir a informação original em hardware ou software, o que naturalmente aumenta a complexidade e custo do sistema. Assim, ondas com polarização circular se fazem interessantes em sistemas de comunicação entre terra e espaço, pois o efeito da ionosfera se resume a uma mudança de fase da onda, a qual não porta informações. As antenas projetadas neste trabalho são de polarização linear, mas o sistema de comunicação desenvolvido deve ter polarização circular. A polarização circular pode ser obtida com dois conjuntos de antenas com polarização linear, orientados a um ângulo de 90^{o} entre si, e com uma defasagem elétrica do sinal de 90^{o}, conforme demonstrado nos arranjos criados.

Utilizando o referencial teórico apresentado neste trabalho, é possível desenvolver um algoritmo de otimização de um arranjo de antenas. Para ser capaz de captar de forma eficiente os sinais eletromagnéticos de um satélite em órbita terrestre baixa, um arranjo de antenas deve manter uma diretividade relativamente constante para todo o plano orbital visível de sua localização. Dadas as otimizações realizadas neste trabalho com os códigos desenvolvidos, fica evidente a necessidade de estudar e desenvolver melhor os métodos de otimização. Cada antena que compõe um arranjo contribui com até 8 variáveis parametrizáveis, ou seja, 8 novas dimensões para o espaço de variáveis a serem otimizadas, o que aumenta exponencialmente a quantidade de simulações necessárias em uma otimização. Assim, a complexidade da otimização aumenta muito rapidamente conforme os arranjos aumentam. A melhor estratégia é garantir que os arranjos inicialmente fornecidos aos algoritmos de otimização já providenciem uma função de custo próxima ao mínimo global

possível, desta forma, algoritmos de otimização convexa podem ser empregados. Sousa (2021) mostra que estes algoritmos são capazes de convergir rapidamente para o mínimo global de uma função de custo convexa. A decomposição dos campos elétricos das antenas em harmônicos de esféras e o uso de produtos internos são ferramentas que tem potêncial de simplificar as otimizações e cálculos dos campos elétricos, assim são ferramentas que devem ser estudas para as aplicações deste trabalho.

É possível determinar qual o melhor diagrama de radiação para manter o satélite dentro do ângulo de abertura a meia potência do arranjo de antenas durante uma passagem de sua órbita sobre o arranjo. Com a distribuição de radiação desejada, os algoritmos desenvolvidos neste trabalho serão utilizados com algoritmos de otimização mais eficientes a serem desenvolvidos de forma que o campo elétrico de um arranjo a ser projetado se assemelhe ao campo elétrico resultante desejado.

Utilizando as famílias de antenas desenvolvidas e os códigos de otimização de arranjo, será desenvolvido um arranjo linear de antenas capaz de se comunicar com um satélite em *LEO* em um plano orbital fixo. Em seguida, este arranjo linear será utilizado para construir um arranjo planar com o intuito de estender a cobertura do arranjo para diversos planos orbitais. Desta maneira, um único arranjo planar final poderá ser capaz de receber sinais de satélites em uma variedade de *LEO*s, resultando em um projeto interessante pela flexibilidade, simplicidade e baixo custo do arranjo a ser produzido.

9 Conclusão

As antenas projetadas no presente trabalho apresentam performance simulada satisfatória, atingindo baixa reflexão na frequência de 433 MHz e diretividade conforme o previsto na listeratura. Um arranjo linear destas antenas pode ser projetado a seguir para que o plano orbital do satélite esteja dentro do ângulo de abertura a meia potência da antena quando este estiver visível. Entretanto, O conjunto de antenas desenvolvido apresenta baixo parâmetro S para apenas uma faixa de frequências muito estreita, assim limitando a banda de frequências em que podem operar. É necessário uma banda mais larga para que a construção destas antenas seja interessante comercialmente.

Os softwares desenvolvidos neste trabalho apresentam uma ferramente muito interessante e poderosa para desenvolver arranjos de antenas arbitrários. Seja o código utilizado para validação e avaliação de arranjos pré-definidos, ou para otimização de arranjos visando atingir uma distribuição de radiação arbitrária, os códigos podem ser utilizados para as mais diversas pesquisas, projetos e objetivos.

Os códigos desenvolvidos neste trabalho, embora rápidos, não levam em consideração a inteferência de materiais posicionados no campo próximo das antenas. O código apresenta grande utilidade em arranjos com muitas antenas e/ou com largas distâncias entre elas. Nestes casos, softwares de simulação numérica como o HFSS se tornam ineficazes pois a complexidade dos problemas cresce com o volume da região de controle, e este cresce exponencialmente conforme antenas são adicionadas ou as dimensões do volume aumentam. Enquanto isso, os códigos desenvolvidos apresentam um crescimento linear com o número de antenas adicionadas, e com o número de pontos de amostragem do campo elétrico, e a demanda computacional não cresce conforme a distância entre as antenas aumenta. Assim, os códigos desenvolvidos se mostram mais adequados para arranjos grandes, justamente onde outros softwares se mostram menos eficazes. Os códigos desenvolvidos então se tornam uma ferramenta complementar à softwares comerciais, não partilhando o mesmo nicho de aplicações, mas acrescentando às capacidades dos projetistas. Com os desenvolvimentos apresentados neste trabalho, foi possível cumprir os objetivos propostos.

Novos estudos devem ser desenvolvidos a fim de aumentar a eficiência dos códigos desenvolvidos, diminuindo o número de computações realizadas a cada cálculo do campo elétrico resultante dos arranjos por meio da reutilização de vetores e matrizes que não necessitam ser recalculados a cada iteração. Outra parte de grande interesse é a utilização de métodos de otimização mais adequados para os problemas apresentados neste trabalho. Deve-se encontrar desenvolvimentos analíticos que permitem escolher condições iniciais

próximas de um mínimo local menor das funções de cuso, assim como utilizar algoritmos de otimização adequados para um alto número de variáveis parametrizadas, que possam caracterizar o hipervolume da função de custo e encontrar os mínimos locais mais ótimos, proporcionando assim soluções melhores para as otimizações.

Os códigos desenvolvidos neste trabalho estão dispostos nos apêndices. Estes foram refratorados para melhorar a legibilidade e facilitar a implementação e utilização dos códigos por qualquer usuário. Entretanto, ainda é necessária a organização dos arquivos para que os códigos funcionem corretamente com a especificação dos caminhos relativos entre os arquivos dos códigos. É necessária ainda a adaptação do nome do caminho dos diretórios para o computador do usuário.

Referências

ASTRON. 2023. Disponível em: <https://www.astron.nl/telescopes/lofar/>. Citado na página 24.

BALANIS, C. C. *Antenna Theory Analysis and Design*. [S.l.: s.n.], 2005. Citado 5 vezes nas páginas 25, 40, 41, 42 e 43.

BHATTACHARYYA, A. K. *Phased array antennas: Floquet analysis, synthesis, BFNs and active array systems*. [S.l.]: John Wiley & Sons, 2006. v. 179. Citado na página 62.

BISNETO, E. Comparison of evolutionary algorithms for synthesis of linear array of antennas with minimal level of sidelobe. 2022. Citado na página 105.

BOYD, S. P.; VANDENBERGHE, L. *Convex optimization*. [S.l.]: Cambridge university press, 2004. Citado na página 105.

Britannica. 2023. Disponível em: <https://www.britannica.com/topic/Arecibo-Observatory>. Citado na página 24.

CHEN, C. Optimum element lenghts for yagi-uda arrays. *IEEE Transactions on Antennas and Propagation*, Jan 1975. Citado 2 vezes nas páginas 23 e 61.

CHEN, W.-K. *The Electrical Engineer Handbook*. [S.l.]: Elsevier Academic Press, 2005. Citado na página 62.

DREGELY, D. et al. 3d optical yagi–uda nanoantenna array. *Nature communications*, Nature Publishing Group UK London, v. 2, n. 1, p. 267, 2011. Citado 2 vezes nas páginas 61 e 62.

Event Horizon Telescope. 2023. Disponível em: <https://eventhorizontelescope.org>. Citado na página 25.

FURSE. *Dipole Antennas*. [S.l.: s.n.], 2007. Citado 4 vezes nas páginas 41, 42, 43 e 45.

GANZ, A.; GONG, Y.; LI, B. Performance study of low earth-orbit satellite systems. *IEEE Transactions on Communications*, v. 42, n. 234, p. 1866–1871, 1994. Citado na página 23.

GODARA, L. C. Applications of antenna arrays to mobile communications. i. performance improvement, feasibility, and system considerations. *Proceedings of the IEEE*, IEEE, v. 85, n. 7, p. 1031–1060, 1997. Citado na página 23.

GRAF, R. F. *Modern dictionary of electronics*. [S.l.]: Elsevier, 1999. Citado na página 40.

GUANELLA, G. New method of impedance matching in radio-frequency circuits. *Brown Boveri Review*, sep 1944. Citado na página 59.

HARIHARAN, P. *Basics of interferometry*. [S.l.]: Elsevier, 2010. Citado na página 62.

HHEINBOCKEL, J. *IntroductiontoTensor calculus and continuum Mechanics*. [S.l.: s.n.], 1996. Citado na página 29.

HILBERT, D. *The foundations of geometry*. [S.l.]: Prabhat Prakashan, 1950. Citado na página 29.

INGRAM, M. A. et al. Leo download capacity analysis for a network of adaptive array ground stations. In: *SERP 2005*. [S.l.: s.n.], 2005. Citado 2 vezes nas páginas 25 e 61.

JACKSON, J. D. *Classical Electrodynamics*. 1st. ed. [S.l.]: John Wiley & Sons, 1962. Citado na página 42.

LUDWIG, A. The definition of cross polarization. *IEEE Transactions on Antennas and Propagation*, v. 21, n. 1, p. 116–119, 1973. Citado na página 94.

MAILLOUX, R. J. *Phased array antenna handbook*. [S.l.]: Artech house, 2017. Citado na página 62.

MARAL, G. et al. Low earth orbit satellite systems for communications. *International Journal of satellite communications*, Wiley Online Library, v. 9, n. 4, p. 209–225, 1991. Citado na página 23.

MAXWELL, J. C. Li. on physical lines of force. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, Taylor & Francis, v. 21, n. 141, p. 338–348, 1861. Citado na página 34.

MAXWELL, J. C. *On Faraday's lines of force*. [S.l.]: DigiCat, 2023. Citado na página 34.

MOON, P.; SPENCER, D. E. *Field Theory Handbook*. [S.l.: s.n.], 1961. Citado 2 vezes nas páginas 32 e 45.

MOREIRA, A. et al. A tutorial on synthetic aperture radar. *IEEE Geoscience and remote sensing magazine*, IEEE, v. 1, n. 1, p. 6–43, 2013. Citado na página 25.

National Geographic. 2022. Disponível em: <https://www.nationalgeographicbrasil.com/espaco/2022/05/divulgada-primeira-imagem-do-buraco-negro-no-centro-de-nossa-galaxia>. Citado na página 25.

NOCEDAL, J.; WRIGHT, S. J. *Numerical optimization*. [S.l.]: Springer, 1999. Citado na página 105.

ORFANIDIS, S. J. *Electromagnetic Waves and Antennas*. [S.l.: s.n.], 1999. Citado 8 vezes nas páginas 43, 51, 52, 53, 54, 55, 56 e 58.

POZAR, D. M. *Microwave Engineering*. [S.l.: s.n.], 1998. Citado 3 vezes nas páginas 35, 36 e 52.

RADIO SOCIETY OF GREAT BRITAIN. *Radio communication handbook 5th ed.* [S.l.], 1976. Citado na página 59.

RE, P. D. H. et al. Circularly polarized retrodirective antenna array for wireless power transmission. *IEEE Transactions on Antennas and Propagation*, IEEE, v. 68, n. 4, p. 2743–2752, 2019. Citado na página 23.

RONDINEAU, S. et al. Ground stations of arrays to increase the leo download capacity. In: IEEE. *2006 European Microwave Conference*. [S.l.], 2006. p. 874–877. Citado 4 vezes nas páginas 24, 25, 26 e 61.

ROSHI, D. A. et al. The future of the arecibo observatory: The next generation arecibo telescope. *arXiv preprint arXiv:2103.01367*, 2021. Citado na página 24.

SANTANA, M. P. *Reconfigurable Metasurface Antenna Array Using Holographic Beamforming*. Dissertação (Mestrado) — Universidade de Brasília, 2021. Citado na página 62.

SARAO. 2022. Disponível em: <https://www.sarao.ac.za/science/meerkat/about-meerkat/>. Citado na página 24.

SKAO. 2023. Disponível em: <https://www.skao.int/en/about-us/skao>. Citado na página 24.

SOUSA, D. C. de. *Integration and Design of a Reconfigurable Reflectarray Antenna with an RF LoRa-based System for IoT Multi-protocol Applications*. Dissertação (Mestrado) — Universidade de Brasília, 2022. Citado na página 62.

SOUSA, W. F. Antena holográfica reconfigurável com otimização de apontamento de feixe pelo uso de redes neurais e otimização convexa. 2021. Citado 2 vezes nas páginas 105 e 125.

STUTZMAN, W. L. *Antenna Theory and Design*. [S.l.: s.n.], 1981. Citado 4 vezes nas páginas 35, 41, 60 e 67.

THIELE, G. Analysis of yagi-uda-type antennas. *IEEE Transactions on Antennas and Propagation*, IEEE, v. 17, n. 1, p. 24–31, 1969. Citado na página 60.

VIEZBIKE, P. Yagi antenna design. *NBS*, Dec 1968. Citado na página 61.

Apêndices

9.1 Octave

Apêndice 9.1 – readFile.m

```
function [Names,Values] = readFile(filename)
  FID = fopen(filename);
  if FID==-1
    Names = [];
    Values = [];
    disp("File␣not␣found");
    return
  end
  s = fgetl(FID);
  if s(1)==’"’
    s(1) = [];
    s(end) = [];
    s = strsplit(s,’\",\"’);
  else
    s = strsplit(s,’,’);
  end
  Names = s;
## template = strjoin(repmat({"%f"},1,length(Names)), ’,’);
  Values = csvread(FID);
  fclose(FID);
## [VAL, COUNT, ERRMSG] = fscanf(FID, template);
## Values = [];
## while (s=fgetl(FID))!=-1
## if isempty(s)
## continue;
## end
## s = strsplit(s,",");
## Values(end+1,:) = s = str2double(s);
## end
end
```

Apêndice 9.2 – save3DDirectivity.m

```
function save3DDirectivity(n, v, tgt)
  Phi = v(:,1);
  Theta = v(:,2);
  Ddb = v(:,3);

  delta_Phi = sum(Theta==Theta(1));
  delta_Theta = sum(Phi==Phi(1));

  Phi = reshape(Phi, delta_Phi, delta_Theta);
  Theta = reshape(Theta, delta_Phi, delta_Theta);
  Ddb = reshape(Ddb, delta_Phi, delta_Theta);

  D = 10.^(Ddb./10);

  X = D.*sind(Theta).*cosd(Phi);
  Y = D.*sind(Theta).*sind(Phi);
  Z = D.*cosd(Theta);

  figure('visible', 'off');
  hold on;
  colormap(jet(64));
  ##scatter3(X, Y, Z, [], v(:,3)-min(v(:,3)), 'filled');
  surf(X, Y, Z, Ddb, 'linestyle', 'none', 'facecolor', 'interp');
  xlabel('x');
  ylabel('y');
  zlabel('z');
  xticks([]);
  yticks([]);
  zticks([]);
  cb = colorbar;
  ylabel(cb, '[dB]');

  grid on;
  title(['Diretividade␣máxima␣=␣', num2str(max(max(Ddb)),2), '␣dB']);
  axis equal;
  view(-45,25);

  hold off;
  print(["Resultados",filesep,tgt,"-3DDirectivity.png"]);
end
```

Apêndice 9.3 – save3DPolarPlot.m

```
function save3DPolarPlot(filename)
  [n,v] = readFile(filename);
  if !isempty(n) && !isempty(v)
    phi = v(v(:,2)==v(1,2),1);
    theta = v(v(:,1)==v(1,1),2);

    [THETA, PHI] = meshgrid(theta, phi);
    mesh_shape = [length(phi) length(theta)];
    C = reshape(v(:,3), mesh_shape);
## R = reshape((v(:,3)-min(v(:,3)))/(max(v(:,3))-min(v(:,3))), mesh_shape);
    R = reshape(v(:,3), mesh_shape);
    R = abs(R);
    maxR = max(R);
    if max(R) > 0.00000001
      R /= max(R)
    endif


    figure('visible', 'off');
    hold on;

    cp = cosd(PHI);
    sp = sind(PHI);
    ct = cosd(THETA);
    st = sind(THETA);
    XX = R.*st.*cp;
    YY = R.*st.*sp;
    ZZ = R.*ct;
    h = surf(XX, YY, ZZ, C, 'linestyle', 'none', 'facecolor', 'interp');
## rotate(h, [0 1 0], 45);
## rotate(h, [0 0 1], 45);

    disp([filename, "\nPrad: ",num2str(sum(sum(C.*st))/prod(size(THETA))),"\n\n"]);

    colormap("jet");
    cb = colorbar;
    caxis([0 10000]);
    xlabel("x");
    ylabel("y");
    zlabel("z");
    xticks([]);
    yticks([]);
    zticks([]);
    grid on;
    axis equal;
## set(gcf, 'visible', 'on');
## set(gca, 'cameraposition', [1 1 0.4]);
## set(gca, 'cameratarget', [0 0 0]);
## set(gca, 'cameraupvector', [0 0 1]);
    view(-45, 30);
## ylabel(cb, '[mV]');
    hold off;

    in = 1;%input("Enter 1 to continue\nEnter 2 to cancel\n");
    if in==1
```

```
    print([filename,".png"]);
  end
 else
  disp("No variables to plot");
 end
end
```

Apêndice 9.4 – saveGraphs.m

```
function saveGraphs(tgt, phi_id, theta_id, D_dB_id)
  [n,v] = readFile(["Resultados",filesep,tgt,".csv"]);

  save3DDirectivity(n,v,tgt);

  phi = v(:,phi_id);
  theta = v(:,theta_id);
  D_dB = v(:,D_dB_id);

  if !isempty(n) && !isempty(v)
    figure('visible', 'off');
    hold on;
    grid on;
    idx = phi==0;
    line(theta(idx), D_dB(idx), 'linewidth', 2, 'color', 'red');
    xlabel("Theta␣[^\\circ]");
    ylabel("Diretividade␣[dB]");
    hold off;
    print(["Resultados",filesep,tgt,"-DirectivityTheta.png"]);
  end

  if !isempty(n) && !isempty(v)
    figure('visible', 'off');
    hold on;
    grid on;
    idx = theta==90;
    line(phi(idx), D_dB(idx), 'linewidth', 2, 'color', 'red');
    xlabel("Phi␣[^\\circ]");
    ylabel("Diretividade␣[dB]");
    hold off;
    print(["Resultados",filesep,tgt,"-DirectivityPhi.png"]);
  end

  [n,v] = readFile(["Resultados",filesep,"SweepPlot",tgt,".csv"]);
  if !isempty(n) && !isempty(v)
    figure('visible', 'off');
    hold on;
    grid on;
    line(v(:,1), v(:,2), 'linewidth', 2, 'color', 'red');
    xlabel(n(1));
    ylabel("S_{11}␣[dB]");
    hold off;
    print(["Resultados",filesep,tgt,"-SweepPlot.png"]);
  end
end
```

Apêndice 9.5 – scrpt.m

```
close all
for FID = fopen("all")
  fclose(FID)
end
##clear
clc

%DIRNAME = uigetdir("D:\\WS\\AnsysEM\\DipoloTeste\\Results\\", "Select a directory with .csv 3d
    ↪ polar plot exported from HFSS");
DIRNAME = [pwd, filesep, "Resultados", filesep, "Dipoles"];
save3DPolarPlot([DIRNAME,filesep,"rE␣Plot␣0-5.csv"]);
save3DPolarPlot([DIRNAME,filesep,"rE␣Plot␣1-0.csv"]);
save3DPolarPlot([DIRNAME,filesep,"rE␣Plot␣1-25.csv"]);
save3DPolarPlot([DIRNAME,filesep,"rE␣Plot␣1-5.csv"]);
##saveGraphs("LogPeriodica");

##DIRNAME = [pwd, filesep, "Resultados", filesep, "Yagis"];
##saveGraphs([DIRNAME,filesep,"Dipolo"], 1,2,3);
##saveGraphs([DIRNAME,filesep,"2EL"],1,2,3);
##saveGraphs([DIRNAME,filesep,"3EL"],1,2,3);
##saveGraphs([DIRNAME,filesep,"4EL"],1,2,3);

##[n,v] = readFile(["Resultados",filesep,"Dipole 0-5.csv"]);
##save3DDirectivity(n,v,"Dipole 0-5");
```

Apêndice 9.6 – show_interpolation.m

```
clear all; close all; clc;

N_theta = 9;
N_phi = 11;

fontsize = 10.0;

elevation = -40;
azimuth = 15;
##roll = 45;
##elevation = 0;
##azimuth = 0;
roll = 0;

theta = linspace(0, 180, N_theta);
phi = linspace(-180, 180, N_phi);

[THETA, PHI] = meshgrid(theta, phi);
N = N_theta*N_phi;
mesh_shape = [N_phi, N_theta];
vector_shape = [1, N];

THETA = THETA(:)';
PHI = PHI(:)';

ct = cosd(THETA);
st = sind(THETA);
cp = cosd(PHI);
sp = sind(PHI);

r = [st.*cp; st.*sp; ct];
t = [ct.*cp; ct.*sp; -st];
p = [-sp; cp; zeros(1, N)];

Re = roty(elevation);
Ra = rotz(azimuth);
Rr = rotx(roll);

R = Ra*Re*Rr;

R_LG = Ra'*Re'*Rr';
R_GL = R_LG';

rll = r;
tll = t;
pll = p;
rgg = r;
tgg = t;
pgg = p;

rlg = R_LG*rll;
tlg = R_LG*tll;
plg = R_LG*pll;

rgl = R_GL*rgg;
```

```
tgl = R_GL*tgg;
pgl = R_GL*pgg;

THETAll = THETA;
PHIll = PHI;
THETAgg = THETA;
PHIgg = PHI;

THETAlg = atan2d(sqrt(sum(rlg([1,2],:).^2,1)), rlg(3,:));
PHIlg = atan2d(plg(1,:), plg(2,:));

THETAgl = atan2d(sqrt(sum(rgl([1,2],:).^2,1)), rgl(3,:));
PHIgl = atan2d(pgl(1,:), pgl(2,:));

figure('visible', 'off');
hold on;

scatter(PHIgg, THETAgg, "b", 'marker', '*');
scatter(PHIlg, THETAlg, "r", 'marker', '*');
PHIgg = reshape(PHIgg, mesh_shape);
THETAgg = reshape(THETAgg, mesh_shape);
mesh(PHIgg, THETAgg, zeros(mesh_shape), 'facecolor', 'none', 'edgecolor', 'b');
PHIgg = reshape(PHIgg, vector_shape);
THETAgg = reshape(THETAgg, vector_shape);

grid on;
xlim([-190, 190]);
ylim([-10, 190]);
set(gca, 'fontsize', fontsize);
xlabel("\\phi␣[^{\\circ}]");
ylabel("\\theta␣[^{\\circ}]");
saveas(gcf, "badInterpolation", 'png');

figure('visible', 'off');
hold on;

scatter(PHIll, THETAll, "r", 'marker', '*');
scatter(PHIgl, THETAgl, "b", 'marker', '*');
PHIll = reshape(PHIll, mesh_shape);
THETAll = reshape(THETAll, mesh_shape);
mesh(PHIll, THETAll, zeros(mesh_shape), 'facecolor', 'none', 'edgecolor', 'r');
PHIll = reshape(PHIll, vector_shape);
THETAll = reshape(THETAll, vector_shape);

grid on;
xlim([-190, 190]);
ylim([-10, 190]);
set(gca, 'fontsize', fontsize);
xlabel("\\phi␣[^{\\circ}]");
ylabel("\\theta␣[^{\\circ}]");
saveas(gcf, "goodInterpolation", 'png');
```

Apêndice 9.7 – CircularPolarization.m

```
clear all;
close all;
clc;

f = 1;
w = 2*pi*f;
t = linspace(0, 1, 51);

hat_rhcp = [1; -1j]/sqrt(2);
hat_lhcp = [1; 1j]/sqrt(2);

F_theta = 1;
F_phi = 1j;
t_exp = exp(1j*w*t);
vec_F = [F_theta; F_phi];

F_rhcp = sum(vec_F.*conj(hat_rhcp), 1);
F_lhcp = sum(vec_F.*conj(hat_lhcp), 1);

vec_F2 = F_rhcp*hat_rhcp + F_lhcp*hat_lhcp;

F_theta2 = F_rhcp*hat_rhcp(1) + F_lhcp*hat_lhcp(1);
F_phi2 = F_rhcp*hat_rhcp(2) + F_lhcp*hat_lhcp(2);

function error = error(f1, f2)
  error = abs(f1-f2);
end

disp(["RHCP: ", num2str(F_rhcp)]);
disp(["LHCP: ", num2str(F_lhcp)]);

disp(["Error theta: ", num2str(error(F_theta, F_theta2))]);
disp(["Error phi: ", num2str(error(F_phi, F_phi2))]);

##figure;
##hold on;
##
##plot(t, real(F_theta.*t_exp), 'displayname', "\\theta");
##plot(t, real(F_phi.*t_exp), 'displayname', "\\phi");
##
##legend;

figure;
hold on;

h1 = quiver(0,0,real(F_theta(1)), real(F_phi(1)));
h2 = line(real(F_theta(1)), real(F_phi(1)), 'marker', '*');

axis equal;
grid on;
xlim([-1,1]);
ylim([-1,1]);
line(cos(2*pi*t), sin(2*pi*t), 'color', 'k');

flag = true;
```

```
while flag
  flag = false;
  for i = 1:length(t)
    vec_F = F_rhcp*hat_rhcp + F_lhcp*hat_lhcp;
    u = vec_F(1).*t_exp(i);
    v = vec_F(2).*t_exp(i);
    set(h1, 'udata', real(u), 'vdata', real(v));
    set(h2, 'xdata', real(u), 'ydata', real(v));
    pause(0.01);
  endfor
end
```

Apêndice 9.8 – create_dlp_visualization.m

```
##close all; clear all; clc;

N = 2;

c0 = 299792458;
f = 433e6;
T = 1/f;
lamb = c0/f;
A = 0.4;

t = 0.5*T;
Dt = 0.125*T;

z_contour = linspace(0, N, 1001);
x_contour = A*cos(2*pi*(z_contour - f*t));
y_contour = A*cos(2*pi*(z_contour - f*t));

z_vector = linspace(0, N, 55);
x_vector = A*cos(2*pi*(z_vector - f*t));
y_vector = A*cos(2*pi*(z_vector - f*t));

figure('visible', show_imgs);
hold on;

xlabel('x', 'fontsize', fontsize);
ylabel('y', 'fontsize', fontsize);
zlabel('z', 'fontsize', fontsize);

axis equal;
xticks([]);
yticks([]);
zticks([]);
grid on;
view(45, 45);

hc = line(x_contour, y_contour, z_contour, 'linewidth', 2.0);
hv = quiver3(
  zeros(size(z_vector)), zeros(size(z_vector)), z_vector,
  x_vector, y_vector, zeros(size(z_vector)),
  0, 'k', 'linewidth', 1.5, "filled");
line([0,0],[0,0],[0,2], 'color', 'b', 'linewidth', 1.5);

saveas(gcf, "LinearPolarizationDiagonal", 'png');

return;

t = t + Dt;

z_contour = linspace(0, N, 1001);
x_contour = Ax*cos(2*pi*(z_contour - f*t));
y_contour = Ay*sin(-2*pi*(z_contour - f*t));

z_vector = linspace(0, N, 55);
x_vector = Ax*cos(2*pi*(z_vector - f*t));
y_vector = Ay*sin(-2*pi*(z_vector - f*t));
```

```
figure;
hold on;

xlabel('x');
ylabel('y');
zlabel('z');

axis equal;
##axis('visible', 'off');
grid on;
view(45, 45);

hc = line(x_contour, y_contour, z_contour, 'linewidth', 2.0);
hv = quiver3(
  zeros(size(z_vector)), zeros(size(z_vector)), z_vector,
  x_vector, y_vector, zeros(size(z_vector)),
  0, 'k', 'linewidth', 1.5, "filled");
line([0,0],[0,0],[0,2], 'color', 'b', 'linewidth', 1.5);
```

Apêndice 9.9 – create_ecp_visualization.m

```
##close all; clear all; clc;

N = 2;

c0 = 299792458;
f = 433e6;
T = 1/f;
lamb = c0/f;
Ax = 0.4;
Ay = 0.6;

t = 0.5*T;
Dt = 0.125*T;

z_contour = linspace(0, N, 1001);
x_contour = Ax*cos(2*pi*(z_contour - f*t));
y_contour = Ay*sin(2*pi*(z_contour - f*t));

z_vector = linspace(0, N, 55);
x_vector = Ax*cos(2*pi*(z_vector - f*t));
y_vector = Ay*sin(2*pi*(z_vector - f*t));

figure('visible', show_imgs);
hold on;

xlabel('x', 'fontsize', fontsize);
ylabel('y', 'fontsize', fontsize);
zlabel('z', 'fontsize', fontsize);

axis equal;
xticks([]);
yticks([]);
zticks([]);
grid on;
view(135, 45);

hc = line(x_contour, y_contour, z_contour, 'linewidth', 2.0);
hv = quiver3(
  zeros(size(z_vector)), zeros(size(z_vector)), z_vector,
  x_vector, y_vector, zeros(size(z_vector)),
  0, 'k', 'linewidth', 1.5, "filled");
line([0,0],[0,0],[0,2], 'color', 'b', 'linewidth', 1.5);

saveas(gcf, "EllipticalPolarization", 'png');

return;

t = t + Dt;

z_contour = linspace(0, N, 1001);
x_contour = Ax*cos(2*pi*(z_contour - f*t));
y_contour = Ay*sin(2*pi*(z_contour - f*t));

z_vector = linspace(0, N, 55);
x_vector = Ax*cos(2*pi*(z_vector - f*t));
```

```
y_vector = Ay*sin(2*pi*(z_vector - f*t));

saveas(gcf, "CircularPolarizationElliptical", 'png');

return;

figure;
hold on;

xlabel('x');
ylabel('y');
zlabel('z');

axis equal;
xticks([]);
yticks([]);
zticks([]);
##axis('visible', 'off');
grid on;
view(135, 45);

hc = line(x_contour, y_contour, z_contour, 'linewidth', 2.0);
hv = quiver3(
  zeros(size(z_vector)), zeros(size(z_vector)), z_vector,
  x_vector, y_vector, zeros(size(z_vector)),
  0, 'k', 'linewidth', 1.5, "filled");
line([0,0],[0,0],[0,2], 'color', 'b', 'linewidth', 1.5);
```

Apêndice 9.10 – create_hlp_visualization.m

```
##close all; clear all; clc;

N = 2;

c0 = 299792458;
f = 433e6;
T = 1/f;
lamb = c0/f;
A = 0.4;
Ax = A*cos(pi/180*30);
Ay = A*sin(pi/180*30);

t = 0.5*T;
Dt = 0.125*T;

z_contour = linspace(0, N, 1001);
x_contour = A*cos(2*pi*(z_contour - f*t));
y_contour = zeros(size(z_contour));

z_vector = linspace(0, N, 55);
x_vector = A*cos(2*pi*(z_vector - f*t));
y_vector = zeros(size(z_vector));

figure('visible', show_imgs);
hold on;

xlabel('x', 'fontsize', fontsize);
ylabel('y', 'fontsize', fontsize);
zlabel('z', 'fontsize', fontsize);

axis equal;
xticks([]);
yticks([]);
zticks([]);
grid on;
view(45, 45);

hc = line(x_contour, y_contour, z_contour, 'linewidth', 2.0);
hv = quiver3(
  zeros(size(z_vector)), zeros(size(z_vector)), z_vector,
  x_vector, y_vector, zeros(size(z_vector)),
  0, 'k', 'linewidth', 1.5, "filled");
line([0,0],[0,0],[0,2], 'color', 'b', 'linewidth', 1.5);

saveas(gcf, "LinearPolarization1", 'png');

return;

t = t + Dt;

z_contour = linspace(0, N, 1001);
x_contour = Ax*cos(2*pi*(z_contour - f*t));
y_contour = Ay*sin(-2*pi*(z_contour - f*t));

z_vector = linspace(0, N, 55);
```

```
x_vector = Ax*cos(2*pi*(z_vector - f*t));
y_vector = Ay*sin(-2*pi*(z_vector - f*t));

figure;
hold on;

xlabel('x');
ylabel('y');
zlabel('z');

axis equal;
set(gcf, 'fontsize', 20);
##axis('visible', 'off');
grid on;
view(45, 45);

hc = line(x_contour, y_contour, z_contour, 'linewidth', 2.0);
hv = quiver3(
  zeros(size(z_vector)), zeros(size(z_vector)), z_vector,
  x_vector, y_vector, zeros(size(z_vector)),
  0, 'k', 'linewidth', 1.5, "filled");
line([0,0],[0,0],[0,2], 'color', 'b', 'linewidth', 1.5);
```

Apêndice 9.11 – create_lhcp_visualization.m

```
##close all; clear all; clc;

N = 2;

c0 = 299792458;
f = 433e6;
T = 1/f;
lamb = c0/f;
Ax = 0.4;
Ay = Ax;

t = 0.5*T;
Dt = 0.125*T;

z_contour = linspace(0, N, 1001);
x_contour = Ax*cos(2*pi*(z_contour - f*t));
y_contour = Ay*sin(2*pi*(z_contour - f*t));

z_vector = linspace(0, N, 55);
x_vector = Ax*cos(2*pi*(z_vector - f*t));
y_vector = Ay*sin(2*pi*(z_vector - f*t));

figure('visible', show_imgs);
hold on;

xlabel('x', 'fontsize', fontsize);
ylabel('y', 'fontsize', fontsize);
zlabel('z', 'fontsize', fontsize);

axis equal;
xticks([]);
yticks([]);
zticks([]);
grid on;
view(135, 45);

hc = line(x_contour, y_contour, z_contour, 'linewidth', 2.0);
hv = quiver3(
  zeros(size(z_vector)), zeros(size(z_vector)), z_vector,
  x_vector, y_vector, zeros(size(z_vector)),
  0, 'k', 'linewidth', 1.5, "filled");
line([0,0],[0,0],[0,2], 'color', 'b', 'linewidth', 1.5);

saveas(gcf, "CircularPolarizationLHCP", 'png');

return;

t = t + Dt;

z_contour = linspace(0, N, 1001);
x_contour = Ax*cos(2*pi*(z_contour - f*t));
y_contour = Ay*sin(2*pi*(z_contour - f*t));

z_vector = linspace(0, N, 55);
x_vector = Ax*cos(2*pi*(z_vector - f*t));
```

```
y_vector = Ay*sin(2*pi*(z_vector - f*t));

figure;
hold on;

xlabel('x');
ylabel('y');
zlabel('z');

axis equal;
##axis('visible', 'off');
grid on;
view(135, 45);

hc = line(x_contour, y_contour, z_contour, 'linewidth', 2.0);
hv = quiver3(
  zeros(size(z_vector)), zeros(size(z_vector)), z_vector,
  x_vector, y_vector, zeros(size(z_vector)),
  0, 'k', 'linewidth', 1.5, "filled");
line([0,0],[0,0],[0,2], 'color', 'b', 'linewidth', 1.5);
```

Apêndice 9.12 – create_rhcp_visualization.m

```
##close all; clear all; clc;

N = 2;

c0 = 299792458;
f = 433e6;
T = 1/f;
lamb = c0/f;
Ax = 0.4;
Ay = Ax;

t = 0.5*T;
Dt = 0.125*T;

z_contour = linspace(0, N, 1001);
x_contour = Ax*cos(2*pi*(z_contour - f*t));
y_contour = Ay*sin(-2*pi*(z_contour - f*t));

z_vector = linspace(0, N, 55);
x_vector = Ax*cos(2*pi*(z_vector - f*t));
y_vector = Ay*sin(-2*pi*(z_vector - f*t));

figure('visible', show_imgs);
hold on;

xlabel('x', 'fontsize', fontsize);
ylabel('y', 'fontsize', fontsize);
zlabel('z', 'fontsize', fontsize);

axis equal;
xticks([]);
yticks([]);
zticks([]);
grid on;
view(45, 45);

hc = line(x_contour, y_contour, z_contour, 'linewidth', 2.0);
hv = quiver3(
  zeros(size(z_vector)), zeros(size(z_vector)), z_vector,
  x_vector, y_vector, zeros(size(z_vector)),
  0, 'k', 'linewidth', 1.5, "filled");
line([0,0],[0,0],[0,2], 'color', 'b', 'linewidth', 1.5);

saveas(gcf, "CircularPolarizationRHCP_1", 'png');

t = t + Dt;

z_contour = linspace(0, N, 1001);
x_contour = Ax*cos(2*pi*(z_contour - f*t));
y_contour = Ay*sin(-2*pi*(z_contour - f*t));

z_vector = linspace(0, N, 55);
x_vector = Ax*cos(2*pi*(z_vector - f*t));
y_vector = Ay*sin(-2*pi*(z_vector - f*t));
```

```
figure('visible', show_imgs);
hold on;

xlabel('x', 'fontsize', fontsize);
ylabel('y', 'fontsize', fontsize);
zlabel('z', 'fontsize', fontsize);

axis equal;
xticks([]);
yticks([]);
zticks([]);
##axis('visible', 'off');
grid on;
view(45, 45);

hc = line(x_contour, y_contour, z_contour, 'linewidth', 2.0);
hv = quiver3(
  zeros(size(z_vector)), zeros(size(z_vector)), z_vector,
  x_vector, y_vector, zeros(size(z_vector)),
  0, 'k', 'linewidth', 1.5, "filled");
line([0,0],[0,0],[0,2], 'color', 'b', 'linewidth', 1.5);

saveas(gcf, "CircularPolarizationRHCP_2", 'png');
```

Apêndice 9.13 – create_rhcp_visualization_gif.m

```
close all; clear all; clc;

N = 2;

c0 = 299792458;
f = 433e6;
T = 1/f;
lamb = c0/f;
Ax = 0.4;
Ay = Ax;

gif_duration = 2;
gif_fps = 15;
t = linspace(0, T, round(gif_duration*gif_fps) + 1);

z_contour = linspace(0, N, 1001);
x_contour = Ax*cos(2*pi*(z_contour - f*t(1)));
y_contour = Ay*sin(-2*pi*(z_contour - f*t(1)));

z_vector = linspace(0, N, 55);
x_vector = Ax*cos(2*pi*(z_vector - f*t(1)));
y_vector = Ay*sin(-2*pi*(z_vector - f*t(1)));

figure;
hold on;

xlabel('x');
ylabel('y');
zlabel('z');

axis equal;

hc = line(x_contour, y_contour, z_contour, 'linewidth', 2.0);
hv = quiver3(
  zeros(size(z_vector)), zeros(size(z_vector)), z_vector,
  x_vector, y_vector, zeros(size(z_vector)),
  0, 'k', 'linewidth', 1.5, "filled");
k_vector = quiver3(0,0,0,0,0,1.6*N, 'b', 'linewidth', 1.5);

h_fps = title("fps: 0");

tic;
gif_time = 0;
gif_next_frame = gif_time + 1/gif_fps;
gif_frames = 0;
while true
  for i = 2:length(t)
    t_i = t(i);

    x_contour = Ax*cos(2*pi*(z_contour - f*t_i));
    y_contour = Ay*sin(-2*pi*(z_contour - f*t_i));

    x_vector = Ax*cos(2*pi*(z_vector - f*t_i));
    y_vector = Ay*sin(-2*pi*(z_vector - f*t_i));
```

```
    set(hc,
      'xdata', x_contour,
      'ydata', y_contour);
    set(hv,
      'udata', x_vector,
      'vdata', y_vector);

    gif_frames++;
    current_fps = gif_frames/toc;
    gif_delay = gif_next_frame - toc;
    gif_time = gif_next_frame;
    gif_next_frame = gif_time + 1/gif_fps;

    set(h_fps, 'string', ['fps: ', num2str(round(current_fps))]);
    pause(gif_delay);
  end
end
```

Apêndice 9.14 – create_rhcp_visualization_gif_file.m

```
close all; clear all; clc;

N = 2;

c0 = 299792458;
f = 433e6;
T = 1/f;
lamb = c0/f;
Ax = 0.4;
Ay = Ax;

gif_duration = 12;
gif_fps = 15;
gif_frames = 2*gif_fps;
t = linspace(0, 4*T, round(gif_duration*gif_fps) + 1);

z_contour = linspace(0, N, 1001);
x_contour = Ax*cos(2*pi*(z_contour - f*t(1)));
y_contour = Ay*sin(-2*pi*(z_contour - f*t(1)));

z_vector = linspace(0, N, 55);
x_vector = Ax*cos(2*pi*(z_vector - f*t(1)));
y_vector = Ay*sin(-2*pi*(z_vector - f*t(1)));

figure;
hold on;

xlabel('x');
ylabel('y');
zlabel('z');

axis equal;
grid on;
view(135, 45);

hc = line(x_contour, y_contour, z_contour, 'linewidth', 2.0);
hv = quiver3(
  zeros(size(z_vector)), zeros(size(z_vector)), z_vector,
  x_vector, y_vector, zeros(size(z_vector)),
  0, 'k', 'linewidth', 1.5, "filled");
k_vector = quiver3(0,0,0,0,0,1.6*N, 'b', 'linewidth', 1.5);

h_fps = title(["fps:␣", num2str(gif_fps)]);

frame = getframe(gcf);
img_data = zeros([size(frame.cdata),gif_frames]);

for i = 2:length(t)
  t_i = t(i);

  x_contour = Ax*cos(2*pi*(z_contour - f*t_i));
  y_contour = Ay*sin(-2*pi*(z_contour - f*t_i));

  x_vector = Ax*cos(2*pi*(z_vector - f*t_i));
  y_vector = Ay*sin(-2*pi*(z_vector - f*t_i));
```

```
  set(hc,
    'xdata', x_contour,
    'ydata', y_contour);
  set(hv,
    'udata', x_vector,
    'vdata', y_vector);

  frame = getframe(gcf);
  img_data(:,:,:,i) = frame.cdata;
end

imwrite(img_data, "ani.gif", 'delaytime', 1/gif_fps);
```

Apêndice 9.15 – create_vlp_visualization.m

```
##close all; clear all; clc;

N = 2;

c0 = 299792458;
f = 433e6;
T = 1/f;
lamb = c0/f;
A = 0.4;

t = 0.5*T;
Dt = 0.125*T;

z_contour = linspace(0, N, 1001);
x_contour = zeros(size(z_contour));
y_contour = A*cos(2*pi*(z_contour - f*t));

z_vector = linspace(0, N, 55);
x_vector = zeros(size(z_vector));
y_vector = A*cos(2*pi*(z_vector - f*t));

figure('visible', show_imgs);
hold on;

xlabel('x', 'fontsize', fontsize);
ylabel('y', 'fontsize', fontsize);
zlabel('z', 'fontsize', fontsize);

axis equal;
xticks([]);
yticks([]);
zticks([]);
grid on;
view(45, 45);

hc = line(x_contour, y_contour, z_contour, 'linewidth', 2.0);
hv = quiver3(
  zeros(size(z_vector)), zeros(size(z_vector)), z_vector,
  x_vector, y_vector, zeros(size(z_vector)),
  0, 'k', 'linewidth', 1.5, "filled");
line([0,0],[0,0],[0,2], 'color', 'b', 'linewidth', 1.5);

saveas(gcf, "LinearPolarization2", 'png');

return;

t = t + Dt;

z_contour = linspace(0, N, 1001);
x_contour = Ax*cos(2*pi*(z_contour - f*t));
y_contour = Ay*sin(-2*pi*(z_contour - f*t));

z_vector = linspace(0, N, 55);
x_vector = Ax*cos(2*pi*(z_vector - f*t));
y_vector = Ay*sin(-2*pi*(z_vector - f*t));
```

```
figure;
hold on;

xlabel('x');
ylabel('y');
zlabel('z');

axis equal;
##axis('visible', 'off');
grid on;
view(45, 45);

hc = line(x_contour, y_contour, z_contour, 'linewidth', 2.0);
hv = quiver3(
  zeros(size(z_vector)), zeros(size(z_vector)), z_vector,
  x_vector, y_vector, zeros(size(z_vector)),
  0, 'k', 'linewidth', 1.5, "filled");
line([0,0],[0,0],[0,2], 'color', 'b', 'linewidth', 1.5);
```

Apêndice 9.16 – generateTheoryImages.m

```
clear all; close all; clc;

fontsize = 20;
show_imgs = 'off';

create_lhcp_visualization;
create_rhcp_visualization;
create_ecp_visualization;
create_hlp_visualization;
create_vlp_visualization;
create_dlp_visualization;
show_rotations;

return;

[img, map, alpha] = imread("Johnson␣1-2␣(blank).png");
imshow(img);
axis on;
text(162, 108, "Posição␣da\nantena",
  'fontsize', fontsize,
  'verticalalignment', 'middle',
  'horizontalalignment', 'right',
  'interpreter', 'latex');
text(360, 297, "\\phi",
  'fontsize', fontsize,
  'verticalalignment', 'top',
  'horizontalalignment', 'right');
text(400, 180, "\\theta",
  'fontsize', fontsize,
  'verticalalignment', 'bottom',
  'horizontalalignment', 'center',
  'interpreter', 'latex');
text(450, 190, "\\tau",
  'fontsize', fontsize,
  'verticalalignment', 'top',
  'horizontalalignment', 'right',
  'interpreter', 'latex');
text(440, 215, "\\vec{u}_\\theta",
  'fontsize', fontsize,
  'verticalalignment', 'top',
  'horizontalalignment', 'right',
  'interpreter', 'latex');
text(500, 145, "\\vec{u}_\\phi",
  'fontsize', fontsize,
  'verticalalignment', 'top',
  'horizontalalignment', 'right',
  'interpreter', 'latex');
```

Apêndice 9.17 – show_azimuth.m

```
newfig;

x3 = ra*x2;
y3 = ra*y2;
z3 = ra*z2;

quiver3(0,0,0,x3(1),x3(2),x3(3), 'g', 'linewidth', 2.0);
quiver3(0,0,0,y3(1),y3(2),y3(3), 'g', 'linewidth', 2.0);
quiver3(0,0,0,z3(1),z3(2),z3(3), 'g', 'linewidth', 2.0);
text(x3(1),x3(2),x3(3), "x\'", 'fontsize', fontsize);
text(y3(1),y3(2),y3(3), "y\'", 'fontsize', fontsize);
text(z3(1),z3(2),z3(3), "z\'", 'fontsize', fontsize);

angles = pi/180*315*t;
aa = G*[-A*cos(angles); -A*sin(angles); 0.5*ones(size(t))];
arrow = G*0.01*sqrt(2)/2*[-1; -1; 0];
line(aa(1,:), aa(2,:), aa(3,:), 'color', 'black', 'linewidth', 1.0);
quiver3(aa(1,end), aa(2,end), aa(3,end), arrow(1), arrow(2), arrow(3),
'color', 'black', 'linewidth', 1.0, 'maxheadsize', 10);

beta = G*[-0.3,0,0.5]';
text(beta(1),beta(2),beta(3), '\beta', 'fontsize', fontsize);

saveas(gcf, "RefSysAzimuthOctave", 'png');
```

Apêndice 9.18 – show_elevation.m

```
newfig;

x2 = re*x1;
y2 = re*y1;
z2 = re*z1;

quiver3(0,0,0,x2(1),x2(2),x2(3), 'g', 'linewidth', 2.0);
quiver3(0,0,0,y2(1),y2(2),y2(3), 'g', 'linewidth', 2.0);
quiver3(0,0,0,z2(1),z2(2),z2(3), 'g', 'linewidth', 2.0);
text(x2(1),x2(2),x2(3), "x\'", 'fontsize', fontsize);
text(y2(1),y2(2),y2(3), "y\'", 'fontsize', fontsize);
text(z2(1),z2(2),z2(3), "z\'", 'fontsize', fontsize);

angles = pi/180*315*t;
ae = G*[-A*cos(angles); 0.5*ones(size(t)); -A*sin(angles)];
arrow = G*0.01*sqrt(2)/2*[-1; 0; -1];
line(ae(1,:), ae(2,:), ae(3,:), 'color', 'black', 'linewidth', 1.0);
quiver3(ae(1,end), ae(2,end), ar(3,end), arrow(1), arrow(2), arrow(3),
'color', 'black', 'linewidth', 1.0, 'maxheadsize', 10);

alpha = G*[0,0.5,0.3]';
text(alpha(1),alpha(2),alpha(3), '\alpha', 'fontsize', fontsize);

##view(19,40);

saveas(gcf, "RefSysElevationOctave", 'png');
```

Apêndice 9.19 – show_refsys.m

```
clear all; close all; clc;

N_theta = 5;
N_phi = 9;

fontsize = 20.0;

elevation = -40;
azimuth = 15;
##roll = 45;
##elevation = 0;
##azimuth = 0;
roll = 0;

theta = linspace(40, 140, N_theta);
phi = linspace(-180, 180, N_phi);

[THETA, PHI] = meshgrid(theta, phi);
N = N_theta*N_phi;
mesh_shape = [N_phi, N_theta];
vector_shape = [1, N];

THETA = THETA(:)';
PHI = PHI(:)';

ct = cosd(THETA);
st = sind(THETA);
cp = cosd(PHI);
sp = sind(PHI);

r = [st.*cp; st.*sp; ct];
t = [ct.*cp; ct.*sp; -st];
p = [-sp; cp; zeros(1, N)];

figure;
hold on;

plot3(r(1,:), r(2,:), r(3,:), 'marker', '*', 'linestyle', 'none', 'color', 'b');
quiver3(r(1,:),r(2,:),r(3,:),t(1,:), t(2,:), t(3,:), 'linewidth', 2.0, 'color', 'g');
quiver3(r(1,:),r(2,:),r(3,:),p(1,:), p(2,:), p(3,:), 'linewidth', 2.0, 'color', 'r');

grid on;
axis equal;
set(gca, 'fontsize', fontsize);
xlabel('x');
ylabel('y');
zlabel('z');
xticklabels([]);
yticklabels([]);
zticklabels([]);
view(45,30);

Re = roty(elevation);
Ra = rotz(azimuth);
Rr = rotx(roll);
```

```
R = Ra*Re*Rr;

x = [1; 0; 0];
xl = R*x;
y = [0; 1; 0];
yl = R*y;
z = [0; 0; 1];
zl = R*z;

figure;
hold on;

quiver3(0,0,0,xl(1), xl(2), xl(3), 'linewidth', 2.0, 'color', 'b');
quiver3(0,0,0,yl(1), yl(2), yl(3), 'linewidth', 2.0, 'color', 'g');
quiver3(0,0,0,zl(1), zl(2), zl(3), 'linewidth', 2.0, 'color', 'r');

grid on;
axis equal;
set(gca, 'fontsize', fontsize);
xlabel('x');
ylabel('y');
zlabel('z');
xticklabels([]);
yticklabels([]);
zticklabels([]);
view(45,30);

R_LG = Ra'*Re'*Rr';
R_GL = R_LG';

rll = r;
tll = t;
pll = p;
rgg = r;
tgg = t;
pgg = p;

rlg = R_LG*rll;
tlg = R_LG*tll;
plg = R_LG*pll;

rgl = R_GL*rgg;
tgl = R_GL*tgg;
pgl = R_GL*pgg;

THETAll = THETA;
PHIll = PHI;
THETAgg = THETA;
PHIgg = PHI;

THETAlg = atan2d(sqrt(sum(rlg([1,2],:).^2,1)), rlg(3,:));
PHIlg = atan2d(plg(1,:), plg(2,:));

THETAgl = atan2d(sqrt(sum(rgl([1,2],:).^2,1)), rgl(3,:));
PHIgl = atan2d(pgl(1,:), pgl(2,:));
```

```
figure;
hold on;

plot3(rlg(1,:), rlg(2,:), rlg(3,:), 'marker', '*', 'linestyle', 'none', 'color', 'b');
quiver3(rlg(1,:),rlg(2,:),rlg(3,:),tlg(1,:), tlg(2,:), tlg(3,:), 'linewidth', 2.0, 'color', 'g')
    ↪ ;
quiver3(rlg(1,:),rlg(2,:),rlg(3,:),plg(1,:), plg(2,:), plg(3,:), 'linewidth', 2.0, 'color', 'r')
    ↪ ;

grid on;
axis equal;
set(gca, 'fontsize', fontsize);
xlabel('x');
ylabel('y');
zlabel('z');
xticklabels([]);
yticklabels([]);
zticklabels([]);
view(45,30);
```

Apêndice 9.20 – show_refsys2.m

```
close all;

function putgrid()
  l = 1.0;
  lw = 0.05;
  c = 0.85;
  lc = [c, c, c];
  G = 2.0;
  for i = -1:0.25:1
    line(G*[-l l],i*G*[-l -l],G*[-l -l], 'color', lc, 'linewidth', lw);
    line(i*G*[-l -l],G*[l l],G*[-l l], 'color', lc, 'linewidth', lw);
    line(G*[-l l],G*[l l],i*G*[-l -l], 'color', lc, 'linewidth', lw);
    line(i*G*[-l -l],G*[-l l],G*[-l -l], 'color', lc, 'linewidth', lw);
    line(G*[-l -l],G*[-l l],i*G*[-l -l], 'color', lc, 'linewidth', lw);
    line(G*[-l -l],i*G*[-l -l],G*[-l l], 'color', lc, 'linewidth', lw);
  end
end

function newfig()
  show_imgs = true;
  fontsize = 20;

  figure('visible', show_imgs);
  hold on;
  xticklabels([]);
  yticklabels([]);
  zticklabels([]);
  axis equal;
  axis off;
  grid on;
  view(55,30);

## xlabel('x', 'fontsize', fontsize);
## ylabel('y', 'fontsize', fontsize);
## zlabel('z', 'fontsize', fontsize);
## l = 1.0;
## xlim([-l, l]);
## ylim([-l, l]);
## zlim([-l, l]);

  putgrid();

  G = 2.0;
  x = G*[1, 0, 0]';
  y = G*[0, 1, 0]';
  z = G*[0.0001, 0, 1]';

  quiver3(0,0,0,x(1),x(2),x(3), 'blue', 'linewidth', 2.0);
  quiver3(0,0,0,y(1),y(2),y(3), 'blue', 'linewidth', 2.0);
  quiver3(0,0,0,z(1),z(2),z(3), 'blue', 'linewidth', 2.0);
  text(x(1),x(2)+0.2,x(3),'x', 'fontsize', fontsize);
  text(y(1),y(2),y(3),'y', 'fontsize', fontsize);
  text(z(1),z(2),z(3),'z', 'fontsize', fontsize);
end
```

```
G = 2.0;

x = G*[1, 0, 0]';
y = G*[0, 1, 0]';
z = G*[0.0001, 0, 1]';

elevation = -30;
azimuth = -130;
roll = 60;
re = roty(elevation);
ra = rotz(azimuth);
rr = rotx(roll);
R = ra*re*rr;

t = linspace(0,1,100);
A = 0.2;
B = 1.4;

show_roll;
show_elevation;
show_azimuth;
```

Apêndice 9.21 – show_roll.m

```
newfig;

x1 = rr*x;
y1 = rr*y;
z1 = rr*z;

quiver3(0,0,0,x1(1),x1(2),x1(3), 'g', 'linewidth', 2.0);
quiver3(0,0,0,y1(1),y1(2),y1(3), 'g', 'linewidth', 2.0);
quiver3(0,0,0,z1(1),z1(2),z1(3), 'g', 'linewidth', 2.0);
text(x1(1),x1(2),x1(3), "x\'", 'fontsize', fontsize);
text(y1(1),y1(2),y1(3), "y\'", 'fontsize', fontsize);
text(z1(1),z1(2),z1(3), "z\'", 'fontsize', fontsize);

angles = pi/180*315*t;
ar = G*[0.5*ones(size(t)); -A*cos(angles); -A*sin(angles)];
arrow = G*0.01*sqrt(2)/2*[0, -1, -1];
line(ar(1,:), ar(2,:), ar(3,:), 'color', 'black', 'linewidth', 1.0);
quiver3(ar(1,end), ar(2,end), ar(3,end), arrow(1), arrow(2), arrow(3),
'color', 'black', 'linewidth', 1.0, 'maxheadsize', 10);

gamma = G*[0.5,-0.5,0]';
text(gamma(1),gamma(2),gamma(3), '\gamma', 'fontsize', fontsize);

saveas(gcf, "RefSysRollOctave", 'png');
```

Apêndice 9.22 – show_rotations.m

```
close all;

show_imgs = true;
fontsize = 20;

figure('visible', show_imgs);
view(15,30);
hold on;
xticks([]);
yticks([]);
zticks([]);
axis equal;
axis off;

xlabel('x', 'fontsize', fontsize);
ylabel('y', 'fontsize', fontsize);
zlabel('z', 'fontsize', fontsize);

x = [1, 0, 0]';
y = [0, 1, 0]';
z = [0.0001, 0, 1]';

x1 = x;
y1 = y;
z1 = z;

elevation = -30;
azimuth = -130;
roll = 60;
re = roty(elevation);
ra = rotz(azimuth);
rr = rotx(roll);
R = ra*re*rr;

x1 = R*x;
y1 = R*y;
z1 = R*z;

quiver3(0,0,0,x(1),x(2),x(3), 'b', 'linewidth', 2.0);
quiver3(0,0,0,y(1),y(2),y(3), 'b', 'linewidth', 2.0);
quiver3(0,0,0,z(1),z(2),z(3), 'b', 'linewidth', 2.0);
quiver3(0,0,0,x1(1),x1(2),x1(3), 'g', 'linewidth', 2.0);
quiver3(0,0,0,y1(1),y1(2),y1(3), 'g', 'linewidth', 2.0);
quiver3(0,0,0,z1(1),z1(2),z1(3), 'g', 'linewidth', 2.0);
B = 1.1;
x = B*x;
y = B*y;
z = B*z;
x1 = B*x1;
y1 = B*y1;
z1 = B*z1;
text(x(1),x(2),x(3), 'x', 'fontsize', fontsize);
text(y(1),y(2),y(3), 'y', 'fontsize', fontsize);
text(z(1),z(2),z(3), 'z', 'fontsize', fontsize);
text(1.1*x1(1),x1(2),x1(3), "x\'", 'fontsize', fontsize);
```

```
text(y1(1),y1(2),y1(3), "y\'", 'fontsize', fontsize);
text(z1(1),z1(2),z1(3), "z\'", 'fontsize', fontsize);

t = linspace(0,1,100);
A = 0.5;

angles = pi/180*azimuth*t;
aa = [cos(angles); sin(angles); zeros(size(t))];
line(A*aa(1,:), A*aa(2,:), A*aa(3,:), 'color', 'black', 'linewidth', 1.0);
line([0, x1(1)], [0, x1(2)], [0, 0],
'linestyle', '--', 'color', 'black', 'linewidth', 1.0);
line([x1(1), x1(1)], [x1(2), x1(2)], [0, x1(3)],
'linestyle', '--', 'color', 'black', 'linewidth', 1.0);

angles = pi/180*elevation*t;
ae = A*ra*[cos(angles); zeros(size(t)); sin(-angles)];
line(ae(1,:), ae(2,:), ae(3,:), 'color', 'black', 'linewidth', 1.0);

angles = pi/180*roll*t;
ar = A*ra*re*[zeros(size(t)); cos(angles); sin(angles)];
line(ar(1,:), ar(2,:), ar(3,:), 'color', 'black', 'linewidth', 1.0);

B = 1.3;
beta = A*B*[cosd(azimuth/2),sind(azimuth/2),0]';
text(beta(1),beta(2),beta(3), '\beta', 'fontsize', fontsize);
alpha = A*B*ra*[cosd(elevation/2),0,-sind(elevation/2)]';
text(alpha(1),alpha(2),alpha(3), '\alpha', 'fontsize', fontsize);
gamma = A*B*ra*re*[0,cosd(roll/2),sind(roll/2)]';
text(gamma(1),gamma(2),gamma(3), '\gamma', 'fontsize', fontsize);

##saveas(gcf, "orientationOctave", 'png');
```

Apêndice 9.23 – create_dipole_model_visualization.m

```
close all; clear all; clc;

figure;
hold on;

grid off;
axis equal;
xlabel('x');
ylabel('y');
zlabel('z');
view(45,45);

t = linspace(0, 1, 51);

##quiver3(0,0,0,1,0,0,'k');
##quiver3(0,0,0,0,1,0,'k');
##quiver3(0,0,0,0,0,1,'k');

line([0,0],[0,0],[0.1,0.4], 'color', 'b', 'linewidth', 3.0);
line([0,0],[0,0],[-0.1,-0.4], 'color', 'b', 'linewidth', 3.0);

quiver3(-0.1, 0, 0.2, 0.0001, 0, 0.2, 'color', 'k', 'linewidth', 1.0);
quiver3(-0.1, 0, 0.2, 0.0001, 0, -0.2, 'color', 'k', 'linewidth', 1.0);
line([-0.05, -0.15],[0,0],[0.4,0.4], 'color', 'k', 'linewidth', 1.0);
line([-0.05, -0.15],[0,0],[0,0], 'color', 'k', 'linewidth', 1.0);

text(-0.15, 0, 0.2, 'h');
```

Apêndice 9.24 – currentDistribution.m

```
c = 299792458;
f = 433e6;
eta = 120*pi;
w = 2*pi*f;
lambda = c/f;
k = w/c;

dipole_length = 0.5;
L = dipole_length*lambda;

z = linspace(-L/2,L/2,200);

I = cos(k*z);

##I = zeros(1,length(z));
##for i = 1:length(I)
## if z >= 0
## I(i) = sin(k.*(L/2 - z(i)));
## else
## I(i) = sin(k.*(L/2 + z(i)));
## end
##end

plot(z, I);
```

Apêndice 9.25 – saveIdealDipole.m

```
function saveIdealDipole(dipole_length, radiatedPower)
  c = 299792458;
  f = 433e6;
  eta = 120*pi;
  w = 2*pi*f;
  lambda = c/f;
  k = w/c;

  fontsize = 20.0;

  %P_in = 1;
  %C_in = 2.435;
  %I_0 = sqrt(P_in/C_in/eta*8*pi);

  L = dipole_length*lambda;
  theta = linspace(0, pi, 90);
  phi = linspace(-pi, pi, 30);
  [THETA, PHI] = meshgrid(theta, phi);

  A = eta;
  B = 2*pi;
  C = cos(k*L/2*cos(THETA)) - cos(k*L/2);
  E = sin(THETA);
  rE = A ./ B .* C ./ E;
  rE(THETA==0) = 0;
  rE = abs(rE);

  W_ar = 1/2/eta*rE.*rE;
  C = cos(k*L/2*cos(theta)) - cos(k*L/2);
  D = C.*C ./ sin(theta) * pi/(length(theta)-1);
  D(theta==0) = 0;
  P_in = eta/4/pi*sum(D)/radiatedPower;
  I_0 = 1/sqrt(P_in);
  rE *= I_0;

  rE *= 1000;

  st = sin(THETA);
  ct = cos(THETA);
  sp = sin(PHI);
  cp = cos(PHI);

## min_rE = min(min(rE));
  max_rE = max(max(rE));
  R = rE/max_rE;
  XX = R.*st.*cp;
  YY = R.*st.*sp;
  ZZ = R.*ct;

  disp([num2str(dipole_length), "␣Prad:␣",
  num2str(sum(sum(rE.*st))/prod(size(THETA)))])

  figure('visible', 'on');
  hold on;
```

```
  h = surf(XX, YY, ZZ, rE, 'linestyle', 'none', 'facecolor', 'interp');
  rotate(h, [0 1 0], 90);
  rotate(h, [0 0 1], 90);

  xlabel('x');
  ylabel('y');
  zlabel('z');
  xticks([]);
  yticks([]);
  zticks([]);
  grid on;
  colormap jet;
  cb = colorbar;
  caxis([0 10000]);
  ylabel(cb, '[mV]');
  axis equal;
## set(gca, 'cameraposition', [1 1 0.4]);
## set(gca, 'cameratarget', [0 0 0]);
## set(gca, 'cameraupvector', [0 0 1]);
  view(-45, 30);
  set(gca, 'fontsize', fontsize);

  dipole_length = strjoin(strsplit(num2str(dipole_length), '.'), '-');
## in = input("Enter 1 to continue\nEnter 2 to cancel\n");
## if in==1
    saveas(gcf, ['rE␣', dipole_length, '.png']);
## end
end
```

Apêndice 9.26 – saveIdealDipoleDirectivity.m

```
function saveIdealDipoleDirectivity()
  c = 299792458;
  f = 433e6;
  eta = 120*pi;
  w = 2*pi*f;
  lambda = c/f;
  k = w/c;

  %P_in = 1;
  %C_in = 2.435;
  %I_0 = sqrt(P_in/C_in/eta*8*pi);

  dipole_length = 0.5;
  L = dipole_length*lambda;
  theta = linspace(1e-3, pi-1e-3, 90);
  phi = linspace(-pi, pi, 90);
  [THETA, PHI] = meshgrid(theta, phi);

  A = eta/2/pi;
  C = cos(k*L/2*cos(THETA)) - cos(k*L/2);
  E = sin(THETA);
  rE = A .* C ./ E;
  rE(THETA==0) = 0;
  rE = abs(rE);

## W_ar = 1/2/eta*rE.*rE;
## C = cos(k*L/2*cos(theta)) - cos(k*L/2);
## D = C.*C ./ sin(theta) * pi/(length(theta)-1);
## D(theta==0) = 0;
## P_in = eta/4/pi*sum(D)/radiatedPower;
## I_0 = 1/sqrt(P_in);
## rE *= 1000*I_0;

## min_rE = min(min(rE));
## max_rE = max(max(rE));
  C = cos(k*L/2*cos(theta)) - cos(k*L/2);

  U = rE.*rE/2/eta;
## P_rad = sum(sum(U.*sin(THETA)))*pi/length(theta)*2*pi/length(phi);
  P_rad = eta/4/pi*sum(C.*C./sin(theta))*pi/length(theta)
  U_iso = P_rad/4/pi;
  D = U/U_iso;
  D_dB = 10*log10(D);

  XX = D.*sin(THETA).*cos(PHI);
  YY = D.*sin(THETA).*sin(PHI);
  ZZ = D.*cos(THETA);

  figure('visible', 'on');
  hold on;

  h = surf(XX, YY, ZZ, D_dB, 'linestyle', 'none', 'facecolor', 'interp');
  rotate(h, [0 1 0], 90);
  rotate(h, [0 0 1], 90);
```

```
  xlabel('x');
  ylabel('y');
  zlabel('z');
  xticks([]);
  yticks([]);
  zticks([]);
  grid on;
  colormap jet;
  cb = colorbar;
  caxis([-30 0]);
  ylabel(cb, '[dB]');
  axis equal;
  view(-45,25);
  title(['Diretividade␣máxima␣=␣', num2str(max(max(D_dB)),2), '␣dB']);

  dipole_length = strjoin(strsplit(num2str(dipole_length), '.'), '-');
## in = input("Enter 1 to continue\nEnter 2 to cancel\n");
## if in==1
    saveas(gcf, ['rE␣', dipole_length, '.png']);
## end
end
```

Apêndice 9.27 – script.m

```
close all
for FID = fopen("all")
  fclose(FID);
end
clear;
clc;

saveIdealDipole(0.5, 0.82032);
saveIdealDipole(1.0, 0.14117);
saveIdealDipole(1.25, 0.18348);
saveIdealDipole(1.5, 0.18348);

##saveIdealDipoleDirectivity(0.5);
```

Apêndice 9.28 – calculateArrayFactor.m

```
antennaArray.Af /= length(antennaArray.antennas);
antennaArray.Af = abs(antennaArray.Af);
```

Apêndice 9.29 – createGraphics.m

```
function createGraphics(array, antv)
  global f;
  global c;
  global lambda;
  global k;

  cd graphics

  C = array.E;

  trisurf_invertedsphere( ...
    array.PHI, ...
    array.THETA, ...
    zeros(1,length(array.PHI)), ...
    C, ...
    'array_magE');
  title("Electric field magnitude");

  cd ..
end
```

Apêndice 9.30 – displayResults.m

```
function displayResults(varargin)
  p = inputParser();
  p.FunctionName = "displayResults";
  p.addRequired("antenna");
  p.addRequired("field_name");
  p.addRequired("plot_type");
  p.addParameter("showIm", false);
  p.addParameter("cmap", "jet");
  p.addParameter("title", "");
  p.addParameter("showImages", 'on');
  p.addParameter("printImages", true);
  p.addParameter("savedir", pwd);
  p.addParameter("overwriteImages", true);
  p.addParameter("close_after", false);

  p.parse(varargin{:});
  args_in = p.Results;

  args_in.antenna = toMeshShape(args_in.antenna);

  ##images_dir = 'C:\Users\16004741 2\OneDrive - unb.br\LoraAEB\Octave\Validation\Images';
## images_dir = '/media/vitinho/DADOS/TCC/Octave/Validation/Images';
  images_dir = args_in.savedir;

  importDataGraphics;

  color_range = "auto";
  if strcmp(args_in.field_name, 'E') || strcmp(args_in.field_name, 'magE')
    field = args_in.antenna.E;
    color_map = "jet";
  elseif strcmp(args_in.field_name, 'E db') || strcmp(args_in.field_name, 'magE db')
    field = args_in.antenna.E/max(max(args_in.antenna.E));
    field = 20*log10(field);
    color_range = [-30 0];
    color_map = "jet";
  elseif strcmp(args_in.field_name, 'E normalized') || strcmp(args_in.field_name, 'magE
      ↪ normalized')
    field = args_in.antenna.E/max(max(args_in.antenna.E));
    color_range = [0 1];
    color_map = "jet";
  elseif strcmp(args_in.field_name, 'Etheta') || strcmp(args_in.field_name, 'magEtheta')
    field = abs(args_in.antenna.Etheta);
    color_map = "jet";
  elseif strcmp(args_in.field_name, 'Etheta db') || strcmp(args_in.field_name, 'magEtheta db')
    field = abs(args_in.antenna.Etheta);
    field /= max(max(field));
    field = 20*log10(field);
    color_range = [-30 0];
    color_map = "jet";
  elseif strcmp(args_in.field_name, 'Etheta normalized') || strcmp(args_in.field_name, '
      ↪ magEtheta normalized')
    field = abs(args_in.antenna.Etheta);
    field /= max(max(field));
    color_range = [0 1];
    color_map = "jet";
```

```
  elseif strcmp(args_in.field_name, 'angEtheta')
    field = angle(args_in.antenna.Etheta);
    color_map = "hsv";
  elseif strcmp(args_in.field_name, 'angEtheta␣normalized')
    field = angle(args_in.antenna.Etheta*exp(-1j*mean(mean(angle(args_in.antenna.Etheta)))));
    color_map = "hsv";
  elseif strcmp(args_in.field_name, 'Ephi') || strcmp(args_in.field_name, 'magEphi')
    field = abs(args_in.antenna.Ephi);
    color_map = "jet";
  elseif strcmp(args_in.field_name, 'Ephi␣db') || strcmp(args_in.field_name, 'magEphi␣db')
    field = abs(args_in.antenna.Ephi);
    field /= max(max(field));
    field = 20*log10(field);
    color_range = [-30 0];
    color_map = "jet";
  elseif strcmp(args_in.field_name, 'Ephi␣normalized') || strcmp(args_in.field_name, 'magEphi␣
      ↪ normalized')
    field = abs(args_in.antenna.Ephi);
    field /= max(max(field));
    color_range = [0 1];
    color_map = "jet";
  elseif strcmp(args_in.field_name, 'angEphi')
    field = angle(args_in.antenna.Ephi);
    color_map = "hsv";
  elseif strcmp(args_in.field_name, 'angEphi␣normalized')
    field = angle(args_in.antenna.Ephi*exp(-1j*mean(mean(angle(args_in.antenna.Ephi)))));
    color_map = "hsv";
  else
    error(["Unknow␣field␣", args_in.field_name])
  endif

  if strcmp(args_in.plot_type, 'inverted␣sphere')
    invertedSphere_rE(args_in.antenna.THETA, args_in.antenna.PHI, field,
    "title", [args_in.antenna.Name, '␣', args_in.field_name],
    "showIm", args_in.showIm,
    "cmap", color_map,
    "color_range", color_range
    );
  elseif strcmp(args_in.plot_type, 'radiation␣diagram')
    radiationDiagram(args_in.antenna, 0, 0, false);
  elseif strcmp(args_in.plot_type, 'polar␣directivity')
    polar_Directivity(args_in.antenna, false);
  elseif strcmp(args_in.plot_type, 'quiver3d')
    quiver3d(args_in.antenna, args_in.antenna.Ephi.*args_in.antenna.phi_hat + args_in.antenna.
        ↪ Etheta.*args_in.antenna.theta_hat, false);
  elseifstrcmp(args_in.plot_type, 'polar3d')
    polar3d(args_in.antenna, 'E', false);
  endif

  HFIGS = findall('type', 'figure');
  if strcmp(args_in.showImages, 'on')
    set(HFIGS, 'visible', 'on');
  end
  if args_in.printImages
    if strcmp(args_in.showImages, 'on')
## input("Press enter to continue.");
    end
```

```
    if !exist(images_dir)
      mkdir(images_dir);
    endif
    for i = 1:length(HFIGS)
      hfig = HFIGS(i);
## images_dir
      filename = [images_dir, filesep, get(hfig, 'filename'), '.png'];
      if or(!isfile([filename, '.png']), args_in.overwriteImages)
        print(hfig, filename, '-dpng');
      end
## if or(!isfile([filename, '.fig']), overwriteImages)
## savefig(hfig, filename);
## end
    end
  end

  if args_in.close_after
    close all;
  endif
end
```

Apêndice 9.31 – emptyAntenna.m

```
function antenna = emptyAntenna(phi_samplei, phi_samplef, Nphi,
    theta_samplei, theta_samplef, Ntheta)

  Nsamples = Nphi*Ntheta;
  PHI_samplings = linspace(phi_samplei,phi_samplef,Nphi);
  THETA_samplings = linspace(theta_samplei,theta_samplef,Ntheta);
  [THETA, PHI] = meshgrid(THETA_samplings, PHI_samplings);
  THETA = THETA(:)';
  PHI = PHI(:)';

  k_hat = zeros(3,Nsamples);
  phi_hat = zeros(3,Nsamples);
  theta_hat = zeros(3,Nsamples);

  cp = cosd(PHI);
  sp = sind(PHI);
  ct = cosd(THETA);
  st = sind(THETA);

  k_hat(1,:) = st.*cp;
  k_hat(2,:) = st.*sp;
  k_hat(3,:) = ct;
  phi_hat(1,:) = -sp;
  phi_hat(2,:) = cp;
  theta_hat(1,:) = ct.*cp;
  theta_hat(2,:) = ct.*sp;
  theta_hat(3,:) = -st;

  antenna = struct(
    'header', [],
    'Name', 'Empty antenna',
    'PHI_samplings', PHI_samplings,
    'THETA_samplings', THETA_samplings,
    'PHI', PHI,
    'THETA', THETA,
    'k_hat', k_hat,
    'phi_hat', phi_hat,
    'theta_hat', theta_hat,
    'Ephi', zeros(1, Nsamples),
    'Etheta', zeros(1, Nsamples),
    'E', zeros(1, Nsamples),
    'magI', 1,
    'phaseI', 0,
    'position', [0 0 0],
    'alpha', 0,
    'beta', 0,
    'referenceSystem', 'local',
    'data_shape', 'vector',
    'N_samples', Nsamples,
    'N_phi', Nphi,
    'N_theta', Ntheta);
end
```

Apêndice 9.32 – evaluateArray.m

```
function array = evaluateArray(array)
  global f;
  global c;
  global lambda;
  global k;

  cp = cosd(array.PHI);
  sp = sind(array.PHI);
  ct = cosd(array.THETA);
  st = sind(array.THETA);
  v = repmat(cat(1,st.*cp,st.*sp,ct),1,1,array.N_antennas);
  pd = repmat(cat(1,-sp,cp,zeros(1,array.N_samples)),1,1,array.N_antennas);
  td = repmat(cat(1,ct.*cp,ct.*sp,-st),1,1,array.N_antennas);

  ct = cosd(cat(3,array.antennas.alpha));
  st = -sind(cat(3,array.antennas.alpha));
  cp = cosd(cat(3,array.antennas.beta));
  sp = -sind(cat(3,array.antennas.beta));
  Rt = zeros(3,3,array.N_antennas);
  Rt(1,1,:) = ct; Rt(1,2,:) = 0; Rt(1,3,:) = st;
  Rt(2,1,:) = 0; Rt(2,2,:) = 1; Rt(2,3,:) = 0;
  Rt(3,1,:) = -st; Rt(3,2,:) = 0; Rt(3,3,:) = ct;
  Rp = zeros(3,3,array.N_antennas);
  Rp(1,1,:) = cp; Rp(1,2,:) = -sp; Rp(1,3,:) = 0;
  Rp(2,1,:) = sp; Rp(2,2,:) = cp; Rp(2,3,:) = 0;
  Rp(3,1,:) = 0; Rp(3,2,:) = 0; Rp(3,3,:) = 1;
  function C = pagemtimes(A,B)
    B = permute(B, [2,1,3]);
    C(3,:,:) = sum(A(3,:,:).*B,2);
    C(2,:,:) = sum(A(2,:,:).*B,2);
    C(1,:,:) = sum(A(1,:,:).*B,2);
  end
  R = pagemtimes(Rt, Rp);

  vi = pagemtimes(R,v);
  R_t = permute(R, [2,1,3]);

  pi = atan2d(vi(2,:,:),vi(1,:,:));
  ti = atan2d(abs(vi(1,:,:) + 1j*vi(2,:,:)),vi(3,:,:));
  ct = cosd(ti);
  st = sind(ti);
  cp = cosd(pi);
  sp = sind(pi);
  pdi = pagemtimes(R_t, cat(1,-sp,cp,zeros(1,array.N_samples,array.N_antennas)));
  tdi = pagemtimes(R_t, cat(1,ct.*cp,ct.*sp,-st));
  Epi = zeros(1,array.N_samples,array.N_antennas);
  Eti = zeros(1,array.N_samples,array.N_antennas);

  for i = 1:array.N_antennas
    ant_i = array.antennas(i);

    t = ant_i.THETA_samplings;
    p = ant_i.PHI_samplings;

    Et = reshape(ant_i.Etheta, ant_i.N_phi, ant_i.N_theta);
```

```
    Ep = reshape(ant_i.Ephi, ant_i.N_phi, ant_i.N_theta);

    Epi(:,:,i) = interp2(t, p, Ep, ti(:,:,i), pi(:,:,i));
    Eti(:,:,i) = interp2(t, p, Et, ti(:,:,i), pi(:,:,i));
  end

  av = cat(3, array.antennas.position);
  ai = cat(3, array.antennas.magI);
  api = cat(3, array.antennas.phaseI);

  Af = exp(-1j*k*sum(v.*av,1)).*ai.*exp(1j*deg2rad(api));
## disp(["k: ", num2str(k)]);
  Epi .*= Af;
  Eti .*= Af;

  array.Ephi = sum(dot(pd,Epi.*pdi+Eti.*tdi,1),3);
  array.Etheta = sum(dot(td,Epi.*pdi+Eti.*tdi,1),3);
## array.Etheta = sum(Epi.*dot(td,pdi,1) + Eti.*dot(td,tdi,1),3);

  #Normalize electric fields
  a = array.Ephi.*conj(array.Ephi);
  b = array.Etheta.*conj(array.Etheta);
  array.E = sqrt(a + b);
  max_magE = max(array.E);

  array.E /= max_magE;
  array.Ephi /= max_magE;
  array.Etheta /= max_magE;
end
```

Apêndice 9.33 – GenerateImages.m

```
clear all; close all; clc;

%Constantes
global f = 4.33e8; % <= Frequência de operação [Hz]
global c = 299792458; % <= Velocidade da luz [m/s]
global lambda = c/f; % <= Comprimento de onda [m]
global k = 2*pi/lambda; % <= Número de onda [rad/m]

##base_antenna = readAntenna('1Y-4EL.csv'); # Not good, creates several "phanton" passive
    ↪ elements
base_antenna = readAntenna('antenna-Yagi-4Elements.csv');

%Problem 1
  x0 = 0;y0 = 0;
  Nx = 1;Ny = 1;
  dx = 0;dy = 0;
  default_phi_orientation = 0;
  default_theta_orientation = 0;
  theta_samplei = 0;
  theta_samplef = 180;
  phi_samplei = -180;
  phi_samplef = 180;
  Ntheta = 91;
  Nphi = 181;
array_1 = gridArrayConstructor(base_antenna,
  default_phi_orientation, default_theta_orientation,
  x0, Nx, dx, y0, Ny, dy,
  theta_samplei, theta_samplef, Ntheta,
  phi_samplei, phi_samplef, Nphi);
array_1.Name = 'Octave␣1';

##Problem 2
  x0 = 0;y0 = 0;
  Nx = 1;Ny = 2;
  dx = 0;dy = 0;
  default_phi_orientation = 0;
  default_theta_orientation = 0;
  theta_samplei = 0;
  theta_samplef = 180;
  phi_samplei = -180;
  phi_samplef = 180;
  Ntheta = 91;
  Nphi = 181;
array_2 = gridArrayConstructor(base_antenna,
  default_phi_orientation, default_theta_orientation,
  x0, Nx, dx, y0, Ny, dy,
  theta_samplei, theta_samplef, Ntheta,
  phi_samplei, phi_samplef, Nphi);
array_2.antennas(1).magI = 1;
array_2.antennas(2).position = lambda*[0 1.5 0]';
array_2.antennas(2).beta = 45;
array_2.antennas(2).alpha = 0;
array_2.antennas(2).magI = 1;
array_2.Name = 'Octave␣2';
```

```
%Problem 3
  x0 = 0;y0 = 0;
  Nx = 1;Ny = 3;
  dx = 0;dy = 0;
  default_phi_orientation = 0;
  default_theta_orientation = 0;
  theta_samplei = 0;
  theta_samplef = 180;
  phi_samplei = -180;
  phi_samplef = 180;
  Ntheta = 91;
  Nphi = 181;
array_3 = gridArrayConstructor(base_antenna,
  default_phi_orientation, default_theta_orientation,
  x0, Nx, dx, y0, Ny, dy,
  theta_samplei, theta_samplef, Ntheta,
  phi_samplei, phi_samplef, Nphi);
array_3.antennas(2).position = lambda*[0 1.5 0]';
array_3.antennas(2).beta = 45;
array_3.antennas(2).alpha = 0;
array_3.antennas(3).position = lambda*[0 -1.5 0]';
array_3.antennas(3).beta = -45;
array_3.antennas(3).alpha = -45;
array_3.Name = 'Octave␣3';

%Problem 4
  x0 = 0;y0 = 0;
  Nx = 1;Ny = 4;
  dx = 0;dy = 0;
  default_phi_orientation = 0;
  default_theta_orientation = 0;
  theta_samplei = 0;
  theta_samplef = 180;
  phi_samplei = -180;
  phi_samplef = 180;
  Ntheta = 91;
  Nphi = 181;
array_4 = gridArrayConstructor(base_antenna,
  default_phi_orientation, default_theta_orientation,
  x0, Nx, dx, y0, Ny, dy,
  theta_samplei, theta_samplef, Ntheta,
  phi_samplei, phi_samplef, Nphi);
array_4.antennas(2).position = lambda*[0 1.5 0]';
array_4.antennas(2).beta = 45;
array_4.antennas(2).alpha = 0;
array_4.antennas(3).position = lambda*[0 -1.5 0]';
array_4.antennas(3).beta = -45;
array_4.antennas(3).alpha = -45;
array_4.antennas(4).position = lambda*[-1.5 1 0]';
array_4.antennas(4).beta = -45;
array_4.antennas(4).alpha = 135;
array_4.Name = 'Octave␣4';

%Problem 5
  x0 = 0;y0 = 0;
  Nx = 1;Ny = 5;
  dx = 0;dy = 0;
```

```
  default_phi_orientation = 0;
  default_theta_orientation = 0;
  theta_samplei = 0;
  theta_samplef = 180;
  phi_samplei = -180;
  phi_samplef = 180;
  Ntheta = 91;
  Nphi = 181;
array_5 = gridArrayConstructor(base_antenna,
  default_phi_orientation, default_theta_orientation,
  x0, Nx, dx, y0, Ny, dy,
  theta_samplei, theta_samplef, Ntheta,
  phi_samplei, phi_samplef, Nphi);
array_5.antennas(2).position = lambda*[0 1.5 0]';
array_5.antennas(2).beta = 45;
array_5.antennas(2).alpha = 0;
array_5.antennas(3).position = lambda*[0 -1.5 0]';
array_5.antennas(3).beta = -45;
array_5.antennas(3).alpha = -45;
array_5.antennas(4).position = lambda*[0.34, -3.14, 1.423]';
array_5.antennas(4).beta = 135;
array_5.antennas(4).alpha = -45;
array_5.antennas(5).position = lambda*[0.83, 1.19, -0.72]';
array_5.antennas(5).beta = 14;
array_5.antennas(5).alpha = 72;
array_5.Name = 'Octave␣5';

antv_1 = readAntenna("1Y-4EL.csv");
antv_1.Name = 'HFSS␣1';
antv_2 = readAntenna("2Y-4EL.csv");
antv_2.Name = 'HFSS␣2';
antv_3 = readAntenna("3Y-4EL.csv");
antv_3.Name = 'HFSS␣3';
antv_4 = readAntenna("4Y-4EL.csv");
antv_4.Name = 'HFSS␣4';
antv_5 = readAntenna("5Y-4EL.csv");
antv_5.Name = 'HFSS␣5';

tic;
array_1 = evaluateArray(array_1);
array_1_time = toc;
tic;
array_2 = evaluateArray(array_2);
array_2_time = toc;
tic;
array_3 = evaluateArray(array_3);
array_3_time = toc;
tic;
array_4 = evaluateArray(array_4);
array_4_time = toc;
tic;
array_5 = evaluateArray(array_5);
array_5_time = toc;

plot_type = 'inverted␣sphere'
s = {
  'magEtheta',
```

```
  'angEtheta',
  'magEphi',
  'angEphi',
  'magE'
  };
for i = 1:length(s)
  field_name = cell2mat(s(i))
  displayResults(array_1, field_name, plot_type,'showImages','off', 'printImages', true, '
      ↪ savedir', '/media/vitinho/DADOS/TCC/Octave/Validation/Images/Array_1/', 'close_after',
      ↪ true);
  displayResults(antv_1, field_name, plot_type,'showImages','off', 'printImages', true, 'savedir
      ↪ ', '/media/vitinho/DADOS/TCC/Octave/Validation/Images/Array_1/', 'close_after', true);
  displayResults(array_2, field_name, plot_type,'showImages','off', 'printImages', true, '
      ↪ savedir', '/media/vitinho/DADOS/TCC/Octave/Validation/Images/Array_2/', 'close_after',
      ↪ true);
  displayResults(antv_2, field_name, plot_type,'showImages','off', 'printImages', true, 'savedir
      ↪ ', '/media/vitinho/DADOS/TCC/Octave/Validation/Images/Array_2/', 'close_after', true);
  displayResults(array_3, field_name, plot_type,'showImages','off', 'printImages', true, '
      ↪ savedir', '/media/vitinho/DADOS/TCC/Octave/Validation/Images/Array_3/', 'close_after',
      ↪ true);
  displayResults(antv_3, field_name, plot_type,'showImages','off', 'printImages', true, 'savedir
      ↪ ', '/media/vitinho/DADOS/TCC/Octave/Validation/Images/Array_3/', 'close_after', true);
  displayResults(array_4, field_name, plot_type,'showImages','off', 'printImages', true, '
      ↪ savedir', '/media/vitinho/DADOS/TCC/Octave/Validation/Images/Array_4/', 'close_after',
      ↪ true);
  displayResults(antv_4, field_name, plot_type,'showImages','off', 'printImages', true, 'savedir
      ↪ ', '/media/vitinho/DADOS/TCC/Octave/Validation/Images/Array_4/', 'close_after', true);
  displayResults(array_5, field_name, plot_type,'showImages','off', 'printImages', true, '
      ↪ savedir', '/media/vitinho/DADOS/TCC/Octave/Validation/Images/Array_5/', 'close_after',
      ↪ true);
  displayResults(antv_5, field_name, plot_type,'showImages','off', 'printImages', true, 'savedir
      ↪ ', '/media/vitinho/DADOS/TCC/Octave/Validation/Images/Array_5/', 'close_after', true);
end

disp(['Time␣elapsed␣for␣calculating␣',array_1.Name,':␣',num2str(array_1_time),'␣seconds']);
disp(['Time␣elapsed␣for␣calculating␣',array_2.Name,':␣',num2str(array_2_time),'␣seconds']);
disp(['Time␣elapsed␣for␣calculating␣',array_3.Name,':␣',num2str(array_3_time),'␣seconds']);
disp(['Time␣elapsed␣for␣calculating␣',array_4.Name,':␣',num2str(array_4_time),'␣seconds']);
disp(['Time␣elapsed␣for␣calculating␣',array_5.Name,':␣',num2str(array_5_time),'␣seconds']);
```

Apêndice 9.34 – gridArrayConstructor.m

```
function array = gridArrayConstructor(baseAntenna,
    default_beta, default_alpha,
    x0, Nx, dx, y0, Ny, dy,
    theta_samplei, theta_samplef, Ntheta,
    phi_samplei, phi_samplef, Nphi)
  disp(['Constructing␣array␣with␣',num2str(Nx*Ny), '␣antennas']);

  global f;
  global c;
  global lambda;
  global k;

  baseAntenna.beta = default_beta;
  baseAntenna.alpha = default_alpha;
  baseAntenna.magI = 1;
  baseAntenna.phaseI = 0;

  antennas = repmat(baseAntenna, 1, Nx*Ny);

  for i = 1:Nx
    for j = 1:Ny
      antennas((i-1)*Ny + j).position = [(x0+(i-1)*dx)*lambda,(y0+(j-1)*dy)*lambda,0]';
    end
  end

  Nsamples = Nphi*Ntheta;

  PHI_samplings = linspace(phi_samplei,phi_samplef,Nphi);
  THETA_samplings = linspace(theta_samplei,theta_samplef,Ntheta);
  [THETA, PHI] = meshgrid(THETA_samplings, PHI_samplings);
  THETA = THETA(:)';
  PHI = PHI(:)';

  cp = cosd(PHI);
  sp = sind(PHI);
  ct = cosd(THETA);
  st = sind(THETA);

  k_hat = zeros(3,Nsamples);
  phi_hat = zeros(3,Nsamples);
  theta_hat = zeros(3,Nsamples);

  k_hat(1,:) = st.*cp;
  k_hat(2,:) = st.*sp;
  k_hat(3,:) = ct;
  phi_hat(1,:) = -sp;
  phi_hat(2,:) = cp;
  theta_hat(1,:) = ct.*cp;
  theta_hat(2,:) = ct.*sp;
  theta_hat(3,:) = -st;

  array = struct(
    'Name', 'Planar␣Antenna␣Array',
    'antennas', antennas,
    'THETA_samplings', THETA_samplings,
```

```
    'PHI_samplings', PHI_samplings,
    'THETA', THETA,
    'PHI', PHI,
    'k_hat', k_hat,
    'phi_hat', phi_hat,
    'theta_hat', theta_hat,
    'Ephi', zeros(1,Nsamples),
    'Etheta', zeros(1,Nsamples),
    'E', zeros(1,Nsamples),
    'E_db', zeros(1,Nsamples),
    'data_shape', 'vector',
    'N_antennas', Nx*Ny,
    'N_samples', Nsamples,
    'N_phi', Nphi,
    'N_theta', Ntheta);
end
```

Apêndice 9.35 – idealDipoleAntenna.m

```
function antenna = idealDipoleAntenna(L,
    phi_samplei, phi_samplef, Nphi,
    theta_samplei, theta_samplef, Ntheta)
  global lambda;
  global k;

  antenna = emptyAntenna( ...
    phi_samplei, phi_samplef, Nphi, ...
    theta_samplei, theta_samplef, Ntheta);

  antenna.Ephi = zeros(1,antenna.N_samples);
  antenna.Etheta = zeros(1,antenna.N_samples);

  A = k*L*lambda/2;
  st = sind(antenna.THETA);
  ct = cosd(antenna.THETA);

  ids = st != 0;
  antenna.Ephi(ids) = (cos(A*ct(ids)) - cos(A))./st(ids);

  #Normalize electric fields
  a = antenna.Ephi.*conj(antenna.Ephi);
  b = antenna.Etheta.*conj(antenna.Etheta);
  antenna.E = sqrt(a + b);
  max_magE = max(max(antenna.E));

  antenna.E /= max_magE;
  antenna.Ephi /= max_magE;
  antenna.Etheta /= max_magE;
end
```

Apêndice 9.36 – importDataGraphics.m

```
1;
##disp('Importing data graphical functions');

function quiver3d(ant, field, showIm)
  figure('visible', showIm, 'filename', 'polar3d rE');
  hold on;

## ant = toVectorShape(ant);
  quiver3(ant.k_hat(1,:), ant.k_hat(2,:), ant.k_hat(3,:), field(1,:), field(2,:), field(3,:));

  axis equal;
  xlabel('x');
  ylabel('y');
  zlabel('z');
end

function polar3d(ant, field_name, showIm)
  figure('visible', showIm, 'filename', ['polar3d rE', field_name]);
  hold on;

  ant = toMeshShape(ant);
  field = getfield(ant, field_name);
  cp = cosd(ant.PHI);
  sp = sind(ant.PHI);
  ct = cosd(ant.THETA);
  st = sind(ant.THETA);
  XX = st.*cp;
  YY = st.*sp;
  ZZ = ct;
  surf(abs(field).*XX, ...
    abs(field).*YY, ...
    abs(field).*ZZ, ...
    abs(field), ...
    'linestyle', 'none', ...
    'facecolor', 'interp');

## title('polar3d field');
  axis equal;
  xlabel('x');
  ylabel('y');
  zlabel('z');
  view(45,30);
  set(gcf, 'filename', [ant.Name,' polar 3d ', field_name])
end

function invertedSphere_rE(varargin)
  p = inputParser();
  p.FunctionName = "invertedSphere_rE";
  p.addRequired("mesh_theta");
  p.addRequired("mesh_phi");
  p.addRequired("mesh_field");
  p.addParameter("showIm", false);
  p.addParameter("cmap", "jet");
  p.addParameter("title", "");
  p.addParameter("color_range", "auto");
```

```
p.parse(varargin{:});
args_in = p.Results;

figure('visible', args_in.showIm, 'filename', ['inverted␣sphere␣rE', args_in.title]);
hold on;

surf(args_in.mesh_theta.*cosd(args_in.mesh_phi), ...
  args_in.mesh_theta.*sind(args_in.mesh_phi), ...
  zeros(size(args_in.mesh_field)), ...
  args_in.mesh_field, ...
  'linestyle', 'none', ...
  'facecolor', 'interp');

R_ticks = [45, 90, 135, 180];
Theta_ticks = [0, 45, 90, 135, 180, 225, 270, 315];
grid_handles = [];
thick_handles = [];
for i = 1:length(R_ticks)
  R = R_ticks(i);
  grid_handles(end+1) = polar(linspace(0,2*pi,361), R*ones(1,361));
  thick_handles(end+1) = text(R*cosd(67.5), R*sind(67.5), [num2str(R_ticks(i)),"^\\circ"], ...
    'horizontalalignment', 'center', ...
    'verticalalignment', 'bottom');
end
for i = 1:length(Theta_ticks)
  grid_handles(end+1) = polar(deg2rad(Theta_ticks(i))*[1 1], [R_ticks(1) R_ticks(end)]);
  if -22.5 < Theta_ticks(i) && Theta_ticks(i) < 22.5
    v_alignment = 'middle';
    h_alignment = 'left';
  elseif 22.5 <= Theta_ticks(i) && Theta_ticks(i) < 67.5
    v_alignment = 'bottom';
    h_alignment = 'left';
  elseif 67.5 <= Theta_ticks(i) && Theta_ticks(i) < 112.5
    v_alignment = 'bottom';
    h_alignment = 'center';
  elseif 112.5 <= Theta_ticks(i) && Theta_ticks(i) < 157.5
    v_alignment = 'bottom';
    h_alignment = 'right';
  elseif 157.5 <= Theta_ticks(i) && Theta_ticks(i) < 202.5
    v_alignment = 'middle';
    h_alignment = 'right';
  elseif 202.5 <= Theta_ticks(i) && Theta_ticks(i) < 247.5
    v_alignment = 'top';
    h_alignment = 'right';
  elseif 247.5 <= Theta_ticks(i) && Theta_ticks(i) < 292.5
    v_alignment = 'top';
    h_alignment = 'center';
  elseif 292.5 <= Theta_ticks(i) && Theta_ticks(i) < 337.5
    v_alignment = 'top';
    h_alignment = 'left';
  else
    v_alignment = 'middle';
    h_alignment = 'left';
  end
  R = R_ticks(end);
  thick_handles(end+1) = text(R*cosd(Theta_ticks(i)), R*sind(Theta_ticks(i)), [num2str(
```

```
        ↪ Theta_ticks(i)),"^\\circ"], 'verticalalignment', v_alignment, 'horizontalalignment',
        ↪ h_alignment);
  end
  set(grid_handles, 'color', [1 1 1], 'linewidth', 1);

  colormap(args_in.cmap);
  colorbar;
  caxis(args_in.color_range);
## title("inverted sphere rE");
  set(gcf, 'filename', [args_in.title,' inverted sphere rE'])
  grid off;
  axis off;
end

function polar_Directivity(ant, showIm)
  figure('visible', showIm, 'filename', 'polar directivity');
  hold on;

  ant = toVectorShape(ant);

  r_grid = 1.3;
  r1 = 1;
  r2 = 1/sqrt(2);
  r3 = 1/sqrt(10);
  grid_handles = [ ...
    polar(linspace(0,2*pi,361), ones(1,361)*r1), ...
    polar(linspace(0,2*pi,361), ones(1,361)*r2), ...
    polar(linspace(0,2*pi,361), ones(1,361)*r3), ...
    polar([0 0], [0 r_grid]), ...
    polar([0 pi/4], [0 r_grid]), ...
    polar([0 pi/2], [0 r_grid]), ...
    polar([0 3*pi/4], [0 r_grid]), ...
    polar([0 pi], [0 r_grid]), ...
    polar([0 5*pi/4], [0 r_grid]), ...
    polar([0 3*pi/2], [0 r_grid]), ...
    polar([0 7*pi/4], [0 r_grid])
  ];
  set(grid_handles, 'color', [.9 .9 .9], 'linewidth', 0.1);
  thick_handles = [
    text(r3*cosd(67.5), r3*sind(67.5), num2str(20*log(r3)), ...
      'horizontalalignment', 'center', ...
      'verticalalignment', 'bottom'), ...
    text(r2*cosd(67.5), r2*sind(67.5), num2str(20*log(r2)), ...
      'horizontalalignment', 'center', ...
      'verticalalignment', 'bottom'), ...
    text(r1*cosd(67.5), r1*sind(67.5), num2str(20*log(r1)), ...
      'horizontalalignment', 'center', ...
      'verticalalignment', 'bottom'), ...
    text(r_grid, 0, "0^\\circ"), ...
    text(r_grid*cosd(45), r_grid*sind(45), "45^\\circ", ...
      'verticalalignment', 'bottom'), ...
    text(r_grid*cosd(90), r_grid*sind(90), "90^\\circ", ...
      'verticalalignment', 'bottom'), ...
    text(r_grid*cosd(135), r_grid*sind(135), "135^\\circ", ...
      'horizontalalignment', 'right', ...
      'verticalalignment', 'bottom'), ...
    text(r_grid*cosd(180), r_grid*sind(180), "180^\\circ", ...
```

```
      'horizontalalignment', 'right'), ...
    text(r_grid*cosd(225), r_grid*sind(225), "225^\\circ", ...
      'horizontalalignment', 'right', ...
      'verticalalignment', 'top'), ...
    text(r_grid*cosd(270), r_grid*sind(270), "270^\\circ", ...
      'verticalalignment', 'top'), ...
    text(r_grid*cosd(315), r_grid*sind(315), "315^\\circ", ...
      'verticalalignment', 'top')
  ];

  ids = ant.THETA==90;

  THETA = ant.PHI(ids);
  RHO = abs(ant.E(ids));
  h = polar(deg2rad(THETA), RHO);
  set(h, 'color', 'b');
  D = RHO(2:end)-RHO(1:end-1);
  DD = shift(D,-1)-D;
  idsD = find(sign(D) != sign(shift(D,1)));
  if !isempty(idsD)
    idsC = idsD==shift(idsD+1,1);
    idsD(idsC) = [];
    idsDD = idsD(DD(idsD)<0);
    [XDD, YDD] = pol2cart(deg2rad(THETA(idsDD)),RHO(idsDD));
    line(XDD, YDD, 'linestyle', 'none', 'marker', '*', 'color', 'g');
    [XDD, YDD] = pol2cart(deg2rad(THETA(idsDD)),RHO(idsDD)+0.05);
    for i = 1:length(idsDD)
      n = 20*log10(RHO(idsDD(i)));
      n(n*n<1e-2) = 0;
      text(XDD(i), YDD(i), num2str(n,3), ...
        'horizontalalignment', 'center', ...
        'verticalalignment', 'middle');
    end
  end

  title('polar␣directivity');
  axis equal;
  axis off;
  grid on;
  xlabel('x');
  ylabel('y');
end
```

Apêndice 9.37 – initConstants.m

```
clear all;
close all;
clc;

%Constantes
global f = 4.33e8; % <= Frequência de operação [Hz]
global c = 299792458; % <= Velocidade da luz [m/s]
global lambda = c/f; % <= Comprimento de onda [m]
global k = 2*pi/lambda; % <= Número de onda [rad/m]
```

Apêndice 9.38 – initiateArrayProblem1.m

```
##clear all;
close all;
clc;

%Constantes
global f = 4.33e8; % <= Frequência de operação [Hz]
global c = 299792458; % <= Velocidade da luz [m/s]
global lambda = c/f; % <= Comprimento de onda [m]
global k = 2*pi/lambda; % <= Número de onda [rad/m]

base_antenna = readAntenna('1Y-4EL.csv');
##base_antenna = readAntenna('antenna-Yagi-4Elements.csv');

%Problem 1
  x0 = 0;y0 = 0;
  Nx = 1;Ny = 1;
  dx = 0;dy = 0;
  default_phi_orientation = 0;
  default_theta_orientation = 0;
  theta_samplei = 0;
  theta_samplef = 180;
  phi_samplei = -180;
  phi_samplef = 180;
  Ntheta = 91;
  Nphi = 181;

array = gridArrayConstructor(base_antenna,
  default_phi_orientation, default_theta_orientation,
  x0, Nx, dx, y0, Ny, dy,
  theta_samplei, theta_samplef, Ntheta,
  phi_samplei, phi_samplef, Nphi);
array.Name = 'Octave 1';

field_name = 'angEtheta'
plot_type = 'inverted sphere'

tic;
array = evaluateArray(array);
toc;
disp(['Time elapsed for calculating ',array.Name,': ',num2str(toc),' seconds']);
displayResults(array, field_name, plot_type);

antv = readAntenna("1Y-4EL.csv");
antv.Name = 'HFSS 1';
displayResults(antv, field_name, plot_type);
```

Apêndice 9.39 – initiateArrayProblem2.m

```
##clear all;
close all;
clc;

%Constantes
global f = 4.33e8; % <= Frequência de operação [Hz]
global c = 299792458; % <= Velocidade da luz [m/s]
global lambda = c/f; % <= Comprimento de onda [m]
global k = 2*pi/lambda; % <= Número de onda [rad/m]

base_antenna = readAntenna('antenna-Yagi-4Elements.csv');
##base_antenna = readAntenna('1Y-4EL.csv');

%Problem 2
  x0 = 0;y0 = 0;
  Nx = 1;Ny = 2;
  dx = 0;dy = 0;
  default_phi_orientation = 0;
  default_theta_orientation = 0;
  theta_samplei = 0;
  theta_samplef = 180;
  phi_samplei = -180;
  phi_samplef = 180;
  Ntheta = 91;
  Nphi = 181;

array = gridArrayConstructor(base_antenna,
  default_phi_orientation, default_theta_orientation,
  x0, Nx, dx, y0, Ny, dy,
  theta_samplei, theta_samplef, Ntheta,
  phi_samplei, phi_samplef, Nphi);

array.antennas(1).magI = 1;
array.antennas(2).position = lambda*[0 1.5 0]';
array.antennas(2).beta = 45;
array.antennas(2).alpha = 0;
array.antennas(2).magI = 1;
array.Name = 'Octave 2';

field_name = 'angEtheta'
plot_type = 'inverted sphere'

tic;
array = evaluateArray(array);
toc;
disp(['Time elapsed for calculating ',array.Name,': ',num2str(toc),' seconds']);
displayResults(array, field_name, plot_type);

antv = readAntenna("2Y-4EL.csv");
antv.Name = 'HFSS 2';
displayResults(antv, field_name, plot_type);
```

Apêndice 9.40 – initiateArrayProblem3.m

```
##clear all;
close all;
clc;

%Constantes
global f = 4.33e8; % <= Frequência de operação [Hz]
global c = 299792458; % <= Velocidade da luz [m/s]
global lambda = c/f; % <= Comprimento de onda [m]
global k = 2*pi/lambda; % <= Número de onda [rad/m]

base_antenna = readAntenna('antenna-Yagi-4Elements.csv');

%Problem 3
  x0 = 0;y0 = 0;
  Nx = 1;Ny = 3;
  dx = 0;dy = 0;
  default_phi_orientation = 0;
  default_theta_orientation = 0;
  theta_samplei = 0;
  theta_samplef = 180;
  phi_samplei = -180;
  phi_samplef = 180;
  Ntheta = 91;
  Nphi = 181;

array = gridArrayConstructor(base_antenna,
  default_phi_orientation, default_theta_orientation,
  x0, Nx, dx, y0, Ny, dy,
  theta_samplei, theta_samplef, Ntheta,
  phi_samplei, phi_samplef, Nphi);

array.antennas(2).position = lambda*[0 1.5 0]';
array.antennas(2).beta = 45;
array.antennas(2).alpha = 0;
array.antennas(3).position = lambda*[0 -1.5 0]';
array.antennas(3).beta = -45;
array.antennas(3).alpha = -45;
array.Name = 'Octave 3';

field_name = 'magE db'
plot_type = 'inverted sphere'

tic;
array = evaluateArray(array);
toc;
disp(['Time elapsed for calculating ',array.Name,': ',num2str(toc),' seconds']);
displayResults(array, field_name, plot_type);

antv = readAntenna("3Y-4EL.csv");
antv.Name = 'HFSS 3';
displayResults(antv, field_name, plot_type);
```

Apêndice 9.41 – initiateArrayProblem4.m

```
##clear all;
close all;
clc;

%Constantes
global f = 4.33e8; % <= Frequência de operação [Hz]
global c = 299792458; % <= Velocidade da luz [m/s]
global lambda = c/f; % <= Comprimento de onda [m]
global k = 2*pi/lambda; % <= Número de onda [rad/m]

base_antenna = readAntenna('antenna-Yagi-4Elements.csv');

%Problem 4
  x0 = 0;y0 = 0;
  Nx = 1;Ny = 4;
  dx = 0;dy = 0;
  default_phi_orientation = 0;
  default_theta_orientation = 0;
  theta_samplei = 0;
  theta_samplef = 180;
  phi_samplei = -180;
  phi_samplef = 180;
  Ntheta = 91;
  Nphi = 181;

array = gridArrayConstructor(base_antenna,
  default_phi_orientation, default_theta_orientation,
  x0, Nx, dx, y0, Ny, dy,
  theta_samplei, theta_samplef, Ntheta,
  phi_samplei, phi_samplef, Nphi);

array.antennas(2).position = lambda*[0 1.5 0]';
array.antennas(2).beta = 45;
array.antennas(2).alpha = 0;
array.antennas(3).position = lambda*[0 -1.5 0]';
array.antennas(3).beta = -45;
array.antennas(3).alpha = -45;
array.antennas(4).position = lambda*[0.34, -3.14, 1.423]';
array.antennas(4).beta = 135;
array.antennas(4).alpha = -45;
array.Name = 'Octave 4';

field_name = 'magE db'
plot_type = 'inverted sphere'

tic;
array = evaluateArray(array);
toc;
disp(['Time elapsed for calculating ',array.Name,': ',num2str(toc),' seconds']);
displayResults(array, field_name, plot_type);

antv = readAntenna("4Y-4EL.csv");
antv.Name = 'HFSS 4';
displayResults(antv, field_name, plot_type);
```

Apêndice 9.42 – initiateArrayProblem5.m

```
##clear all;
close all;
clc;

%Constantes
global f = 4.33e8; % <= Frequência de operação [Hz]
global c = 299792458; % <= Velocidade da luz [m/s]
global lambda = c/f; % <= Comprimento de onda [m]
global k = 2*pi/lambda; % <= Número de onda [rad/m]

base_antenna = readAntenna('antenna-Yagi-4Elements.csv');

%Problem 5
  x0 = 0;y0 = 0;
  Nx = 1;Ny = 5;
  dx = 0;dy = 0;
  default_phi_orientation = 0;
  default_theta_orientation = 0;
  theta_samplei = 0;
  theta_samplef = 180;
  phi_samplei = -180;
  phi_samplef = 180;
  Ntheta = 91;
  Nphi = 181;

array = gridArrayConstructor(base_antenna,
  default_phi_orientation, default_theta_orientation,
  x0, Nx, dx, y0, Ny, dy,
  theta_samplei, theta_samplef, Ntheta,
  phi_samplei, phi_samplef, Nphi);

array.antennas(2).position = lambda*[0 1.5 0]';
array.antennas(2).beta = 45;
array.antennas(2).alpha = 0;
array.antennas(3).position = lambda*[0 -1.5 0]';
array.antennas(3).beta = -45;
array.antennas(3).alpha = -45;
array.antennas(4).position = lambda*[0.34, -3.14, 1.423]';
array.antennas(4).beta = 135;
array.antennas(4).alpha = -45;
array.antennas(5).position = lambda*[0.83, 1.19, -0.72]';
array.antennas(5).beta = 14;
array.antennas(5).alpha = 72;
array.Name = 'Octave 5';

field_name = 'magE db'
plot_type = 'inverted sphere'

tic;
array = evaluateArray(array);
toc;
disp(['Time elapsed for calculating ',array.Name,': ',num2str(toc),' seconds']);
displayResults(array, field_name, plot_type);

antv = readAntenna("5Y-4EL.csv");
```

```
antv.Name = 'HFSS 5';
displayResults(antv, field_name, plot_type);
```

Apêndice 9.43 – main.m

```
##clear all;
close all;
clc;

%Constantes
global f = 4.33e8; % <= Frequência de operação [Hz]
global c = 299792458; % <= Velocidade da luz [m/s]
global lambda = c/f; % <= Comprimento de onda [m]
global k = 2*pi/lambda; % <= Número de onda [rad/m]

tic;
base_antenna = readAntenna('antenna-Dipole.csv');
disp(['Time␣elapsed␣for␣reading␣base␣antenna:␣',num2str(toc),'␣seconds']);

tic;

ant = idealDipoleAntenna(0.5, -180, 180, 91, 0, 180, 91);
tg_ant = rotateElectricFields(ant, 45, 0);
arr = gridArrayConstructor(ant,
  0, 0,
  0, 1, 0,
  0, 1, 0,
  0, 180, 91,
  -180, 180, 91);
disp(['Time␣elapsed␣for␣initiating␣array:␣',num2str(toc),'␣seconds']);

tic;
arr = evaluateArray(arr);
disp(['Time␣elapsed␣for␣evaluating␣array:␣',num2str(toc),'␣seconds']);

arr = rotateElectricFields(arr, 0, 70);

tic;
displayResults(tg_ant);
displayResults(ant);
disp(['Time␣elapsed␣for␣displaying␣results:␣',num2str(toc),'␣seconds']);

tic;
arr = optimization(arr, tg_ant);
disp(['Time␣elapsed␣for␣achieving␣target␣antenna:␣',num2str(toc),'␣seconds']);
```

Apêndice 9.44 – optimization.m

```
function [array, X, C] = optimization(array, tg_antenna, optpar)
  disp('Extracting␣initial␣guess␣from␣current␣antenna␣array.');
  X0 = zeros(1,optpar.N*array.N_antennas);
  for i = 1:array.N_antennas
    n = i - 1;
    idx = 1;
    if optpar.x
      X0(optpar.N*n+idx) = array.antennas(i).position(1);
      idx++;
    end
    if optpar.y
      X0(optpar.N*n+idx) = array.antennas(i).position(2);
      idx++;
    end
    if optpar.z
      X0(optpar.N*n+idx) = array.antennas(i).position(3);
      idx++;
    end
    if optpar.beta
      X0(optpar.N*n+idx) = array.antennas(i).beta;
      idx++;
    end
    if optpar.alpha
      X0(optpar.N*n+idx) = array.antennas(i).alpha;
    end
  endfor

  function C = costfunction(X, array, tg_antenna, optpar)
    global f;
    global c;
    global lambda;
    global k;

    for i = 1:array.N_antennas
      n = i-1;
      idx = 1;
      if optpar.x
        array.antennas(i).position(1) = X(optpar.N*n+idx);
        idx++;
      end
      if optpar.y
        array.antennas(i).position(2) = X(optpar.N*n+idx);
        idx++;
      end
      if optpar.z
        array.antennas(i).position(3) = X(optpar.N*n+idx);
        idx++;
      end
      if optpar.beta
        array.antennas(i).beta = X(optpar.N*n+idx);
        idx++;
      end
      if optpar.alpha
        array.antennas(i).alpha = X(optpar.N*n+idx);
      end
```

```
    endfor

    array = evaluateArray(array);
    dEphi = abs(array.Ephi) - abs(tg_antenna.Ephi);
    dEtheta = abs(array.Etheta) - abs(tg_antenna.Etheta);
  ## C = sum(dEphi.*conj(dEphi) + dEtheta.*conj(dEtheta), 2);
    C = sum(sum(dEphi.*dEphi + dEtheta.*dEtheta));
    disp(['Evaluated␣cost␣function␣C␣=␣',num2str(C)]);%,' with X = ', num2str(X)]);
  end

  disp('Optimization␣started.')
  [X, FVAL, INFO, OUTPUT, FJAC] = fsolve(@(X) costfunction(X, array, tg_antenna, optpar), X0);

  disp(['Optimization␣finalized␣with␣cost␣function␣C␣=␣',num2str(FVAL)]);%,', Position X = ',
      ↪ num2str(X(1:3)), ', beta = ', num2str(X(4)), ' and alpha = ', num2str(X(5))]);

  for i = 1:array.N_antennas
    n = i-1;
    idx = 1;
    if optpar.x
      array.antennas(i).position(1) = X(optpar.N*n+idx);
      idx++;
    end
    if optpar.y
      array.antennas(i).position(2) = X(optpar.N*n+idx);
      idx++;
    end
    if optpar.z
      array.antennas(i).position(3) = X(optpar.N*n+idx);
      idx++;
    end
    if optpar.beta
      array.antennas(i).beta = X(optpar.N*n+idx);
      idx++;
    end
    if optpar.alpha
      array.antennas(i).alpha = X(optpar.N*n+idx);
    end
  endfor
  array = evaluateArray(array);

  disp(INFO);
  disp(OUTPUT);
end
```

Apêndice 9.45 – polarplot3d.m

Apêndice 9.46 – radiationDiagram.m

```
function radiationDiagram(antenna, beta, alpha, showIm)
  N = 301;
  angles = linspace(-pi,pi,N);
  points = zeros(3, N);
  points(1,:) = cos(angles);
  points(2,:) = sin(angles);

  points = rotz(beta)*roty(alpha)*points;
  interp_phi = atan2d(points(2,:), points(1,:));
  interp_theta = atan2d(sqrt(dot(points(1:2,:),points(1:2,:),1)),points(3,:));

  antenna = toMeshShape(antenna);
  Ephi = interp2(antenna.THETA_samplings,antenna.PHI_samplings, antenna.Ephi, interp_theta,
      ↪ interp_phi);
  Etheta = interp2(antenna.THETA_samplings,antenna.PHI_samplings, antenna.Etheta, interp_theta,
      ↪ interp_phi);

  figure('visible', showIm, 'filename', 'radiation␣diagram␣rE');
  hold on;

  line(rad2deg(angles), Ephi);

  grid on;
  xlabel("Angle");
  ylabel("rE");
  hold off;
end
```

Apêndice 9.47 – readAntenna.m

```
function antenna = readAntenna(filename)
  antenna = struct(
    'header', [],
    'Name', ['Antenna␣imported␣from␣file␣',filename],
    'PHI_samplings', [],
    'THETA_samplings', [],
    'PHI', [],
    'THETA', [],
    'k_hat', [],
    'phi_hat', [],
    'theta_hat', [],
    'Ephi', [],
    'Etheta', [],
    'E', [],
    'magI', 1,
    'phaseI', 0,
    'position', [0 0 0],
    'alpha', 0,
    'beta', 0,
    'referenceSystem', 'local',
    'data_shape', 'vector',
    'N_samples', [],
    'N_phi', [],
    'N_theta', []);

  if !isempty(filename)
    ##antennas_dir = 'C:\Users\160047412\OneDrive - unb.br\LoraAEB\Antennas';
## antennas_dir = '/media/vitinho/DADOS/TCC/Antennas';
## antennas_dir = '/mnt/325947A912590BDE/TCC/Antennas';
    antennas_dir = fileparts(mfilename('fullpath'));
    antennas_dir = strsplit(antennas_dir, filesep);
    antennas_dir = strjoin([antennas_dir(1:end-2), "Antennas"], filesep);
    FID = fopen([antennas_dir, filesep, filename]);
    antenna.header = fgetl(FID);
    C = textscan(FID, "%q,%f,%f,%f␣%f,%f␣%f");
  ## antenna.variations = C(1);
    antenna.PHI = rad2deg(cell2mat(C(2)))';
    antenna.THETA = rad2deg(cell2mat(C(3)))';
    antenna.PHI_samplings = antenna.PHI(antenna.THETA==antenna.THETA(1));
    antenna.THETA_samplings = antenna.THETA(antenna.PHI==antenna.PHI(1));

    antenna.N_theta = length(antenna.THETA_samplings);
    antenna.N_phi = length(antenna.PHI_samplings);
    antenna.N_samples = antenna.N_theta*antenna.N_phi;

    antenna.k_hat = zeros(3,antenna.N_samples);
    antenna.phi_hat = zeros(3,antenna.N_samples);
    antenna.theta_hat = zeros(3,antenna.N_samples);

    cp = cosd(antenna.PHI);
    sp = sind(antenna.PHI);
    ct = cosd(antenna.THETA);
    st = sind(antenna.THETA);

    antenna.k_hat(1,:) = st.*cp;
```

```
    antenna.k_hat(2,:) = st.*sp;
    antenna.k_hat(3,:) = ct;
    antenna.phi_hat(1,:) = -sp;
    antenna.phi_hat(2,:) = cp;
    antenna.theta_hat(1,:) = ct.*cp;
    antenna.theta_hat(2,:) = ct.*sp;
    antenna.theta_hat(3,:) = -st;

## antenna.Ephi = transpose(cell2mat(C(4)) .* exp(1j*cell2mat(C(5))));
## antenna.Etheta = transpose(cell2mat(C(6)) .* exp(1j*cell2mat(C(7))));
    antenna.Ephi = transpose(cell2mat(C(4)) .* exp(-1j*cell2mat(C(5))));
    antenna.Etheta = transpose(cell2mat(C(6)) .* exp(-1j*cell2mat(C(7))));

    #Normalize electric fields
    a = antenna.Ephi.*conj(antenna.Ephi);
    b = antenna.Etheta.*conj(antenna.Etheta);
    antenna.E = sqrt(a + b);
    max_magE = max(max(antenna.E));

    antenna.E /= max_magE;
    antenna.Ephi /= max_magE;
    antenna.Etheta /= max_magE;

    fclose(FID);
  end
end
```

Apêndice 9.48 – resample.m

```
function antenna = resample(antenna, theta_samplei, theta_samplef, Ntheta, phi_samplei,
    ↪ phi_samplef, Nphi)
  Nsamples = Nphi*Ntheta;

  PHI_samplings = linspace(phi_samplei,phi_samplef,Nphi);
  THETA_samplings = linspace(theta_samplei,theta_samplef,Ntheta);
  [THETA, PHI] = meshgrid(THETA_samplings, PHI_samplings);
  THETA = THETA(:)';
  PHI = PHI(:)';

  cp = cosd(PHI);
  sp = sind(PHI);
  ct = cosd(THETA);
  st = sind(THETA);

  k_hat = zeros(3,Nsamples);
  phi_hat = zeros(3,Nsamples);
  theta_hat = zeros(3,Nsamples);

  k_hat(1,:) = st.*cp;
  k_hat(2,:) = st.*sp;
  k_hat(3,:) = ct;
  phi_hat(1,:) = -sp;
  phi_hat(2,:) = cp;
  theta_hat(1,:) = ct.*cp;
  theta_hat(2,:) = ct.*sp;
  theta_hat(3,:) = st;

  antenna.THETA_samplings = THETA_samplings;
  antenna.PHI_samplings = PHI_samplings;
  antenna.THETA = THETA;
  antenna.PHI = PHI;
  antenna.k_hat = k_hat;
  antenna.phi_hat = phi_hat;
  antenna.theta_hat = theta_hat;

  antenna.data_shape = 'vector';

  antenna.N_samples = Nsamples;
  antenna.N_phi = Nphi;
  antenna.N_theta = Ntheta;
end
```

Apêndice 9.49 – rotateElectricFields.m

```
function ant = rotateElectricFields(ant, alpha, beta)
  ant = toVectorShape(ant);
  tool_ant = ant;

  R = roty(-alpha)*rotz(-beta);
  rE_vec = R*(ant.Etheta.*ant.theta_hat + ant.Ephi.*ant.phi_hat);
  tool_ant.k_hat = R*ant.k_hat;

  tool_ant.THETA = atan2d(sqrt(dot(tool_ant.k_hat(1:2,:), tool_ant.k_hat(1:2,:), 1)), tool_ant.
      ↪ k_hat(3,:));
  tool_ant.PHI = atan2d(tool_ant.k_hat(2,:), tool_ant.k_hat(1,:));

  ct = cosd(tool_ant.THETA);
  st = sind(tool_ant.THETA);
  cp = cosd(tool_ant.PHI);
  sp = sind(tool_ant.PHI);
  tool_ant.phi_hat = zeros(3,tool_ant.N_samples);
  tool_ant.theta_hat = zeros(3,tool_ant.N_samples);
  tool_ant.phi_hat(1,:) = -sp;
  tool_ant.phi_hat(2,:) = cp;
  tool_ant.theta_hat(1,:) = ct.*cp;
  tool_ant.theta_hat(2,:) = ct.*sp;
  tool_ant.theta_hat(3,:) = -st;
  R = R';
  tool_ant.theta_hat = R*tool_ant.theta_hat;
  tool_ant.phi_hat = R*tool_ant.phi_hat;

  ant = toMeshShape(ant);
  tool_ant = toMeshShape(tool_ant);

  tool_ant.Etheta = interp2(ant.THETA, ant.PHI, ant.Etheta, tool_ant.THETA, tool_ant.PHI);
  tool_ant.Ephi = interp2(ant.THETA, ant.PHI, ant.Ephi, tool_ant.THETA, tool_ant.PHI);

  ant = toVectorShape(ant);
  tool_ant = toVectorShape(tool_ant);

  rE_vec = tool_ant.theta_hat.*tool_ant.Etheta + tool_ant.phi_hat.*tool_ant.Ephi;

  ant.Etheta = dot(rE_vec, ant.theta_hat, 1);
  ant.Ephi = dot(rE_vec, ant.phi_hat, 1);

  #Re-evaluate rE total
  a = ant.Ephi.*conj(ant.Ephi);
  b = ant.Etheta.*conj(ant.Etheta);
  ant.E = sqrt(a + b);
end
```

Apêndice 9.50 – saveGraphs.m

```
##clear all;
close all;
clc;

ant = readAntenna('Dipolo.csv');
```

Apêndice 9.51 – script.m

```
##initiateArrayProblem1;
##initiateArrayProblem2;
initiateArrayProblem3;
initiateArrayProblem4;
initiateArrayProblem5;
```

Apêndice 9.52 – testingScript.m

```
##clear all;
close all;
clc;

%Constantes
global f = 4.33e8; % <= Frequência de operação [Hz]
global c = 299792458; % <= Velocidade da luz [m/s]
global lambda = c/f; % <= Comprimento de onda [m]
global k = 2*pi/lambda; % <= Número de onda [rad/m]

base_antenna = readAntenna('antenna-Yagi-4Elements.csv');

%Problem 3
  x0 = 0;y0 = 0;
  Nx = 1;Ny = 1;
  dx = 0;dy = 0;
  default_phi_orientation = 0;
  default_theta_orientation = 0;
  theta_samplei = 0;
  theta_samplef = 180;
  phi_samplei = -180;
  phi_samplef = 180;
  Ntheta = 91;
  Nphi = 91;

array = gridArrayConstructor(base_antenna,
  default_phi_orientation, default_theta_orientation,
  x0, Nx, dx, y0, Ny, dy,
  theta_samplei, theta_samplef, Ntheta,
  phi_samplei, phi_samplef, Nphi);
array.antennas(1).position = lambda*[0, 1.0, 0]';

array = evaluateArray(array);

disp(['Time␣elapsed␣for␣calculating␣',array.Name,':␣',num2str(toc),'␣seconds']);
displayResults(array);
```

Apêndice 9.53 – toMeshShape.m

```
function ant = toMeshShape(ant)
  if strcmp(ant.data_shape,'grid')
    return;
  end

  mesh_shape = [ant.N_phi, ant.N_theta];

  ant.THETA = reshape(ant.THETA, mesh_shape);
  ant.PHI = reshape(ant.PHI, mesh_shape);
  ant.k_hat = permute(reshape(ant.k_hat, [3, mesh_shape]), [3,2,1]);
  ant.theta_hat = permute(reshape(ant.theta_hat, [3, mesh_shape]), [3,2,1]);
  ant.phi_hat = permute(reshape(ant.phi_hat, [3, mesh_shape]), [3,2,1]);
  ant.Etheta = reshape(ant.Etheta, mesh_shape);
  ant.Ephi = reshape(ant.Ephi, mesh_shape);
  ant.E = reshape(ant.E, mesh_shape);

  ant.data_shape = 'grid';
end
```

Apêndice 9.54 – toVectorShape.m

```
function ant = toVectorShape(ant)
  if strcmp(ant.data_shape,'vector')
    return;
  end

  ant.THETA = ant.THETA(:)';
  ant.PHI = ant.PHI(:)';
  ant.k_hat = reshape(permute(ant.k_hat, [3,2,1]), [3, ant.N_samples]);
  ant.theta_hat = reshape(permute(ant.theta_hat, [3,2,1]), [3, ant.N_samples]);
  ant.phi_hat = reshape(permute(ant.phi_hat, [3,2,1]), [3, ant.N_samples]);
  ant.Etheta = ant.Etheta(:);
  ant.Ephi = transpose(ant.Ephi(:));
  ant.E = transpose(ant.E(:));

  ant.data_shape = 'vector';
end
```

Apêndice 9.55 – validation1Y_RT.m

```
##clear all;
close all;
clc;

%Constantes
global f = 4.33e8; % <= Frequência de operação [Hz]
global c = 299792458; % <= Velocidade da luz [m/s]
global lambda = c/f; % <= Comprimento de onda [m]
global k = 2*pi/lambda; % <= Número de onda [rad/m]

##base_antenna = readAntenna('1Y-4EL.csv');
base_antenna = readAntenna('antenna-Yagi-4Elements.csv');

%Problem 1
  x0 = 0;y0 = 0;
  Nx = 1;Ny = 1;
  dx = 0;dy = 0;
  default_phi_orientation = 0;
  default_theta_orientation = 0;
  theta_samplei = 0;
  theta_samplef = 180;
  phi_samplei = -180;
  phi_samplef = 180;
  Ntheta = 91;
  Nphi = 181;

array = gridArrayConstructor(base_antenna,
  default_phi_orientation, default_theta_orientation,
  x0, Nx, dx, y0, Ny, dy,
  theta_samplei, theta_samplef, Ntheta,
  phi_samplei, phi_samplef, Nphi);
array.Name = 'Octave 1';
array.antennas(1).position = lambda*[0 1.5 0]';
array.antennas(1).beta = 45;
array.antennas(1).alpha = 0;
array.antennas(1).magI = 1;

field_name = 'angEtheta'
plot_type = 'inverted sphere'

tic;
array = evaluateArray(array);
toc;
disp(['Time elapsed for calculating ',array.Name,': ',num2str(toc),' seconds']);
displayResults(array, field_name, plot_type);

antv = readAntenna("1Y-4EL-Rotated-Translated.csv");
antv.Name = 'HFSS 1';
displayResults(antv, field_name, plot_type);
```

Apêndice 9.56 – validation3Dipoles.m

```
##clear all;
close all;
clc;

%Constantes
global f = 4.33e8; % <= Frequência de operação [Hz]
global c = 299792458; % <= Velocidade da luz [m/s]
global lambda = c/f; % <= Comprimento de onda [m]
global k = 2*pi/lambda; % <= Número de onda [rad/m]

base_antenna = readAntenna('Dipole.csv');

%Problem 3 Dipoles
  x0 = 0;y0 = 0;
  Nx = 1;Ny = 3;
  dx = 0;dy = 0;
  default_phi_orientation = 0;
  default_theta_orientation = 0;
  theta_samplei = 0;
  theta_samplef = 180;
  phi_samplei = -180;
  phi_samplef = 180;
  Ntheta = 91;
  Nphi = 181;

array = gridArrayConstructor(base_antenna,
  default_phi_orientation, default_theta_orientation,
  x0, Nx, dx, y0, Ny, dy,
  theta_samplei, theta_samplef, Ntheta,
  phi_samplei, phi_samplef, Nphi);

array.antennas(1).position = lambda*[0 0 0]';
array.antennas(2).position = lambda*[-0.25 0.25 0]';
array.antennas(3).position = lambda*[-0.25 -0.25 0]';
array.Name = 'Octave 3 Dipoles';

field_name = 'E'
plot_type = 'inverted sphere'

tic;
array = evaluateArray(array);
toc;
disp(['Time elapsed for calculating ',array.Name,': ',num2str(toc),' seconds']);
displayResults(array, field_name, plot_type);

antv = readAntenna("Dipoles3.csv");
antv.Name = 'HFSS 3 Dipoles';
displayResults(antv, field_name, plot_type);
```

Apêndice 9.57 – validationCustom.m

```
##clear all;
close all;
clc;

%Constantes
global f = 4.33e8; % <= Frequência de operação [Hz]
global c = 299792458; % <= Velocidade da luz [m/s]
global lambda = c/f; % <= Comprimento de onda [m]
global k = 2*pi/lambda; % <= Número de onda [rad/m]

base_antenna = readAntenna('antenna-Yagi-4Elements.csv');

%Problem Validation Custom
  x0 = 0;y0 = 0;
  Nx = 1;Ny = 1;
  dx = 0;dy = 0;
  default_phi_orientation = 0;
  default_theta_orientation = 0;
  theta_samplei = 0;
  theta_samplef = 180;
  phi_samplei = -180;
  phi_samplef = 180;
  Ntheta = 91;
  Nphi = 181;

array = gridArrayConstructor(base_antenna,
  default_phi_orientation, default_theta_orientation,
  x0, Nx, dx, y0, Ny, dy,
  theta_samplei, theta_samplef, Ntheta,
  phi_samplei, phi_samplef, Nphi);

array.antennas(1).position = lambda*[0 1.5 0]';
array.antennas(1).beta = 45;
array.antennas(1).alpha = 10;
array.Name = 'Octave 3';

field_name = 'angEtheta'
plot_type = 'inverted sphere'

tic;
array = evaluateArray(array);
toc;
disp(['Time elapsed for calculating ',array.Name,': ',num2str(toc),' seconds']);
displayResults(array, field_name, plot_type);
```

Apêndice 9.58 – validationDipole.m

```
##clear all;
close all;
clc;

%Constantes
global f = 4.33e8; % <= Frequência de operação [Hz]
global c = 299792458; % <= Velocidade da luz [m/s]
global lambda = c/f; % <= Comprimento de onda [m]
global k = 2*pi/lambda; % <= Número de onda [rad/m]

base_antenna = readAntenna('Dipole.csv');
##base_antenna = readAntenna('1Y-4EL.csv');

%Problem Translated Dipole
  x0 = 0;y0 = 0;
  Nx = 1;Ny = 1;
  dx = 0;dy = 0;
  default_phi_orientation = 0;
  default_theta_orientation = 0;
  theta_samplei = 0;
  theta_samplef = 180;
  phi_samplei = -180;
  phi_samplef = 180;
  Ntheta = 91;
  Nphi = 181;

array = gridArrayConstructor(base_antenna,
  default_phi_orientation, default_theta_orientation,
  x0, Nx, dx, y0, Ny, dy,
  theta_samplei, theta_samplef, Ntheta,
  phi_samplei, phi_samplef, Nphi);

array.Name = 'Octave Dipole';

field_name = 'E'
plot_type = 'inverted sphere'

tic;
array = evaluateArray(array);
toc;
disp(['Time elapsed for calculating ',array.Name,': ',num2str(toc),' seconds']);
displayResults(array, field_name, plot_type);

antv = readAntenna("Dipole.csv");
antv.Name = 'HFSS Dipole';
displayResults(antv, field_name, plot_type);
```

Apêndice 9.59 – validationDipoleTranslated.m

```
##clear all;
close all;
clc;

%Constantes
global f = 4.33e8; % <= Frequência de operação [Hz]
global c = 299792458; % <= Velocidade da luz [m/s]
global lambda = c/f; % <= Comprimento de onda [m]
global k = 2*pi/lambda; % <= Número de onda [rad/m]

base_antenna = readAntenna('Dipole.csv');
##base_antenna = readAntenna('1Y-4EL.csv');

%Problem Translated Dipole
  x0 = 0;y0 = 0;
  Nx = 1;Ny = 1;
  dx = 0;dy = 0;
  default_phi_orientation = 0;
  default_theta_orientation = 0;
  theta_samplei = 0;
  theta_samplef = 180;
  phi_samplei = -180;
  phi_samplef = 180;
  Ntheta = 91;
  Nphi = 181;

array = gridArrayConstructor(base_antenna,
  default_phi_orientation, default_theta_orientation,
  x0, Nx, dx, y0, Ny, dy,
  theta_samplei, theta_samplef, Ntheta,
  phi_samplei, phi_samplef, Nphi);

array.antennas(1).position = lambda*[1 0 0]';
array.Name = 'Octave Dipole';

field_name = 'E'
plot_type = 'inverted sphere'

tic;
array = evaluateArray(array);
toc;
disp(['Time elapsed for calculating ',array.Name,': ',num2str(toc),' seconds']);
displayResults(array, field_name, plot_type);

antv = readAntenna("DipoleTranslated.csv");
antv.Name = 'HFSS Dipole';
displayResults(antv, field_name, plot_type);
```

9.2 Python

Apêndice 9.60 – generate_attachments_files.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Nov 5 21:49:21 2023

@author: vitinho
"""

import sys
import os

def write_entry(f, filepath, filename, language):
    f.write("\n\\newpage")
    f.write("\n\\lstinputlisting[language=")
    f.write(language)
    f.write(",␣caption=")
    f.write(filename.replace("_", "\\_"))
    f.write(",␣basicstyle=\\ttfamily\scriptsize]")
    f.write("{" + os.path.join("attachments", filepath, filename) + "}")

def process_file(f, filepath, file):
    filename = os.path.basename(file)
    if filename.endswith(".m"):
        language = "Octave"
    elif filename.endswith(".py"):
        language = "Python"
    else:
        return
    write_entry(f, filepath, filename, language)

def process_folder(f, curfolder, depth=0):
    if depth==0:
        f.write("\n\n\\section{" + curfolder + "}")
    folders = [os.path.join(curfolder, folder) \
              for folder in os.listdir(curfolder) \
                  if os.path.isdir(os.path.join(curfolder, folder))]
    for folder in folders:
        process_folder(f, folder, depth+1)

    files = [os.path.join(curfolder, folder) \
              for folder in os.listdir(curfolder) \
                  if os.path.isfile(os.path.join(curfolder, folder))]

    for file in files:
        process_file(f, curfolder, file)

with open("attachments.tex", "w", encoding="utf-8") as f:
    f.write("\\printindex")
    f.write("\n\\lstlistoflistings\n")
    f.write("\n%␣\\input{attachments/software-description}")

    process_file(f, '', __file__)
```

```
folders = [folder \
          for folder in os.listdir(os.getcwd()) \
              if os.path.isdir(folder)]
for folder in folders:
    process_folder(f, folder)

f.write("\n\n\\newpage")
```

Apêndice 9.61 – invertedSphereDefinition.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 18 18:55:55 2023

@author: vitinho
"""

import sys
import os
import header

#python_path = header.python_path
results_dir = os.path.join(header.results_dir, 'Validation')
#home_dir = header.home_dir
antennas_dir = header.antennas_dir

import numpy as np

import Antenna
import Array

import matplotlib.pyplot as plt

theta=np.linspace(0, 180, 91)
phi=np.linspace(-180, 180, 91)

file_name = 'antenna-Yagi-4Elements.csv'
file_path = os.path.join(antennas_dir, file_name)
hfss_yagi4EL = Antenna.load_from_file(file_path,
                              name='Yagi 4EL',
                              theta=theta,
                              phi=phi,
                              load_mesh_from_file=False)

array_rotated = Array.Array(
    name='Array 1Y4El',
    theta=theta.copy(),
    phi=phi.copy(),
    antennas=[hfss_yagi4EL])
array_rotated.antennas[0].set_orientation(roll=60, elevation=-30,
                                          azimuth=-130)
array_rotated.evaluate()

plot='3d Polar'
field='F'
color='Color by magnitude'
in_dB = False

export_directory = os.path.join(results_dir,
                                'InvertedSphere')
if not os.path.exists(export_directory):
    os.mkdir(export_directory)

import ResultFigure
```

```
import Result

figure = ResultFigure.ResultFigure()
plot = '2d␣Polar␣Patch'
Result.Result(tab=figure,
             title='',
             antenna=array_rotated,
             field=field,
             color=color,
             plot=plot,
             # ticks_flag=False,
             in_dB=True,
             Ntheta=91,
             Nphi=91,)
figure.draw()
fname = os.path.join(
    export_directory, '2d␣polar␣patch' + '.png')
figure.figure.savefig(fname)
plt.close('all')

figure = ResultFigure.ResultFigure()
plot = '2d␣Polar␣Patch␣Type␣2'
Result.Result(tab=figure,
             title='',
             antenna=array_rotated,
             field=field,
             color=color,
             plot=plot,
             # ticks_flag=False,
             in_dB=True,
             Ntheta=91,
             Nphi=91,)
figure.draw()
fname = os.path.join(
    export_directory, '2d␣polar␣patch␣2' + '.png')
figure.figure.savefig(fname)
plt.close('all')
```

Apêndice 9.62 – Antenna.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 26 17:48:55 2023

@author: Vítor Lima Aguirra
"""

import pickle
import time

import numpy as np
import csv
from scipy.interpolate import RegularGridInterpolator

from NumpyExpressionParser import NumpyExpressionParser as NEP
import MyMath
import AntennaEditorFrame

constants = dict()
constants["c"] = 299792458 # m/s
constants["f"] = 433e6 # Hz
constants["eta"] = 120 * np.pi
constants["lam"] = constants["c"] / constants["f"] # m
constants["w"] = 2 * np.pi * constants["f"] # rad/s
constants["k"] = 2 * np.pi / constants["lam"] # rad/m


class Antenna:
    EditorFrame = AntennaEditorFrame.AntennaEditorFrame

    def __init__(
        self,
        constants=None,
        name="new antenna",
        current_magnitude=1,
        current_phase=0,
        theta=np.linspace(0, 180, 21),
        phi=np.linspace(-180, 180, 21),
        local_theta_deg=np.linspace(0, 180, 91),
        local_phi_deg=np.linspace(-180, 180, 91),
        elevation=0,
        azimuth=0,
        roll=0,
        x=0,
        y=0,
        z=0,
        evaluate_as="ideal dipole",
    ):
        # self.constants=constants
        self.name = name
        self.current_magnitude = current_magnitude
        self.current_phase = current_phase
        self.theta = theta
        self.phi = phi
        self.local_theta_deg = local_theta_deg
```

```
        self.local_phi_deg = local_phi_deg
        self.elevation = elevation
        self.azimuth = azimuth
        self.roll = roll
        self.x = x
        self.y = y
        self.z = z
        self.evaluate_as = evaluate_as

        self.mesh_phi, self.mesh_theta = np.meshgrid(
            np.radians(self.phi), np.radians(self.theta)
        )
        self.sin_mesh_phi = np.sin(self.mesh_phi)
        self.sin_mesh_theta = np.sin(self.mesh_theta)
        self.shape = (self.theta.size, self.phi.size)

        self.F = np.zeros(self.shape)
        self.Fphi = np.zeros(self.shape, dtype=np.csingle)
        self.Ftheta = np.zeros(self.shape, dtype=np.csingle)
        self.Frhcp = np.zeros(self.shape, dtype=np.csingle)
        self.Flhcp = np.zeros(self.shape, dtype=np.csingle)
        self.Fx = np.zeros(self.shape, dtype=np.csingle)
        self.Fy = np.zeros(self.shape, dtype=np.csingle)
        self.Fz = np.zeros(self.shape, dtype=np.csingle)
        self.Fref = np.zeros(self.shape, dtype=np.csingle)
        self.Fcross = np.zeros(self.shape, dtype=np.csingle)

        self.listeners = []
        self.antenna_size = 0.5
        self.silent = False
        self.local_mesh_N_theta = 91
        self.local_mesh_N_phi = 91

        self.evaluation_time = 0

        self.local_theta = np.radians(self.local_theta_deg)
        self.local_phi = np.radians(self.local_phi_deg)
        self.local_theta_rad = self.local_theta
        self.local_phi_rad = self.local_phi

        self.evaluation_arguments = dict()
        self.evaluation_arguments["dipole␣length"] = 0.5
        self.evaluation_arguments["loop␣dipole␣area"] = 0.5
        self.evaluation_arguments["expression␣theta"] = (
            "(cos(k*L*lam/2*cos(theta))␣-␣"
            + "cos(k*L*lam/2))/(sin(theta)+(sin(theta)==0))"
        )
        self.evaluation_arguments["expression␣phi"] = "S*k*sin(theta)"
        self.evaluation_arguments["isotropic␣on"] = "theta"
        self.evaluation_arguments["file␣path"] = ""
        self.evaluation_arguments["force␣reload"] = False
        self.evaluation_arguments["load␣mesh␣from␣file"] = False

        self.evaluate_local_mesh()
        self.evaluate_R()
        self.evaluate_Rtheta()
        self.evaluate_Rphi()
```

```
        self.evaluate_hats()

        self.local_mesh_flag = False
        self.R_flag = False
        self.Relevation_flag = True
        self.Razimuth_flag = True
        self.Rtheta_flag = False
        self.Rphi_flag = False
        self.hats_flag = False
        self.LG_interp_mesh_flag = True
        self.local_field_flag = True
        self.ok = False

    def notify(self, caller, event):
        self.ok = False
        self.mark_update(
            '"'
            + str(caller)
            + '" called notify with event "'
            + event
            + '"'
        )

    def mark_update(self, event):
        if self.silent:
            return
        for l in self.listeners:
            l.notify(self, event)

    def set_name(self, name):
        self.name = name
        self.mark_update("renamed")

    def set_evaluation_method(self, evaluation_method):
        self.evaluate_as = evaluation_method
        self.local_field_flag = True
        self.ok = False

    def set_orientation(
        self, roll=None, azimuth=None, elevation=None
    ):
        if roll is not None:
            self.roll = roll
        if azimuth is not None:
            self.azimuth = azimuth
        if elevation is not None:
            self.elevation = elevation

        self.R_flag = True
        self.LG_interp_mesh_flag = True
        self.ok = False

    def set_position(self, x=None, y=None, z=None):
        if x is not None:
            self.x = x
        if y is not None:
            self.y = y
```

```
        if z is not None:
            self.z = z

    def set_current(self, magnitude=None, phase=None):
        if magnitude is not None:
            self.current_magnitude = magnitude
        if phase is not None:
            self.current_phase = phase

        self.ok = False

    def resample(self, theta, phi):
        self.phi = phi
        self.theta = theta
        self.mesh_phi, self.mesh_theta = np.meshgrid(
            np.radians(self.phi), np.radians(self.theta)
        )
        self.shape = (self.theta.size, self.phi.size)
        self.F = np.zeros(self.shape)
        self.Fphi = np.zeros(self.shape, dtype=np.csingle)
        self.Ftheta = np.zeros(self.shape, dtype=np.csingle)
        self.Frhcp = np.zeros(self.shape, dtype=np.csingle)
        self.Flhcp = np.zeros(self.shape, dtype=np.csingle)
        self.Fx = np.zeros(self.shape, dtype=np.csingle)
        self.Fy = np.zeros(self.shape, dtype=np.csingle)
        self.Fz = np.zeros(self.shape, dtype=np.csingle)
        self.Fref = np.zeros(self.shape, dtype=np.csingle)
        self.Fcross = np.zeros(self.shape, dtype=np.csingle)

        self.Relevation_flag = True
        self.Razimuth_flag = True
        self.Rtheta_flag = True
        self.Rphi_flag = True
        self.hats_flag = True
        self.LG_interp_mesh_flag = True
        self.ok = False

    def evaluate_local_mesh(self):
        self.local_mesh_flag = False

        self.local_mesh_phi, self.local_mesh_theta = np.meshgrid(
            self.local_phi, self.local_theta
        )
        self.local_shape = (
            self.local_theta.size,
            self.local_phi.size,
        )
        self.local_mesh_theta_rad = self.local_mesh_theta
        self.local_mesh_phi_rad = self.local_mesh_phi
        self.sin_local_mesh_phi = np.sin(self.local_mesh_phi_rad)
        self.sin_local_mesh_theta = np.sin(self.local_mesh_theta_rad)

        self.local_hat_k = np.zeros(
            (self.local_theta.size, self.local_phi.size, 3)
        )
        self.local_hat_theta = np.zeros_like(self.local_hat_k)
        self.local_hat_phi = np.zeros_like(self.local_hat_k)
```

```
        self.local_hat_k[:, :, 2] = 1
        self.local_hat_theta[:, :, 0] = 1
        self.local_hat_phi[:, :, 1] = 1

        Rtheta = MyMath.roty(self.local_mesh_theta_rad)
        Rphi = MyMath.rotz(self.local_mesh_phi_rad)
        R = np.matmul(Rphi, Rtheta)
        self.local_hat_k = np.matmul(
            R, self.local_hat_k[:, :, :, np.newaxis]
        ).squeeze()
        self.local_hat_theta = np.matmul(
            R, self.local_hat_theta[:, :, :, np.newaxis]
        ).squeeze()
        self.local_hat_phi = np.matmul(
            R, self.local_hat_phi[:, :, :, np.newaxis]
        ).squeeze()

    def evaluate_R(self):
        self.R_flag = False

        cp = np.cos(np.radians(self.azimuth))
        sp = np.sin(np.radians(self.azimuth))
        Rbeta = np.zeros((3, 3))
        Rbeta[0, 0] = cp
        Rbeta[0, 1] = sp
        Rbeta[1, 0] = -sp
        Rbeta[1, 1] = cp
        Rbeta[2, 2] = 1
        ct = np.cos(np.radians(self.elevation))
        st = np.sin(np.radians(self.elevation))
        Ralpha = np.zeros((3, 3))
        Ralpha[0, 0] = ct
        Ralpha[0, 2] = -st
        Ralpha[2, 0] = st
        Ralpha[1, 1] = 1
        Ralpha[2, 2] = ct
        cr = np.cos(np.radians(self.roll))
        sr = np.sin(np.radians(self.roll))
        Rroll = np.zeros((3, 3))
        Rroll[0, 0] = 1
        Rroll[1, 1] = cr
        Rroll[1, 2] = sr
        Rroll[2, 1] = -sr
        Rroll[2, 2] = cr
        self.R = np.zeros((1, 1, 3, 3))
        self.R[0, 0, :, :] = Rroll @ Ralpha @ Rbeta

    def evaluate_Rtheta(self):
        self.Rtheta_flag = False

        ct = np.cos(self.mesh_theta)
        st = np.sin(self.mesh_theta)
        self.Rtheta = np.zeros((self.theta.size, self.phi.size, 3, 3))
        self.Rtheta[:, :, 0, 0] = ct
        self.Rtheta[:, :, 0, 2] = st
        self.Rtheta[:, :, 2, 0] = -st
        self.Rtheta[:, :, 1, 1] = 1
```

```
        self.Rtheta[:, :, 2, 2] = ct

    def evaluate_Rphi(self):
        self.Rphi_flag = False

        cp = np.cos(self.mesh_phi)
        sp = np.sin(self.mesh_phi)
        self.Rphi = np.zeros((self.theta.size, self.phi.size, 3, 3))
        self.Rphi[:, :, 0, 0] = cp
        self.Rphi[:, :, 0, 1] = -sp
        self.Rphi[:, :, 1, 0] = sp
        self.Rphi[:, :, 1, 1] = cp
        self.Rphi[:, :, 2, 2] = 1

    def evaluate_hats(self):
        self.hats_flag = False

        self.hat_k = np.zeros((self.theta.size, self.phi.size, 3))
        self.hat_theta = np.zeros((self.theta.size, self.phi.size, 3))
        self.hat_phi = np.zeros((self.theta.size, self.phi.size, 3))
        self.hat_k[:, :, 2] = 1
        self.hat_theta[:, :, 0] = 1
        self.hat_phi[:, :, 1] = 1

        self.hat_k = MyMath.rotate(self.hat_k, self.Rtheta)
        self.hat_theta = MyMath.rotate(self.hat_theta, self.Rtheta)

        self.hat_k = MyMath.rotate(self.hat_k, self.Rphi)
        self.hat_theta = MyMath.rotate(self.hat_theta, self.Rphi)
        self.hat_phi = MyMath.rotate(self.hat_phi, self.Rphi)

    def evaluate_LG_interp_mesh(self):
        # L - Local
        # G - Global
        # GL - Global on Local for vectors or
        # Global to Local for matrices
        # LG - Local on Global for vectors or
        # Local to Global for matrices

        self.LG_interp_mesh_flag = False

        GL_hat_k = MyMath.rotate(self.hat_k, self.R)
        GL_hat_theta = MyMath.rotate(self.hat_theta, self.R)

        GL_x = GL_hat_k[:, :, 0]
        GL_y = GL_hat_k[:, :, 1]
        GL_z = GL_hat_k[:, :, 2]

        self.GL_interp_mesh_theta = np.arctan2(
            np.sqrt(GL_y * GL_y + GL_x * GL_x), GL_z
        )

        ids = (self.GL_interp_mesh_theta > 3) + (
            self.GL_interp_mesh_theta < 0.1
        )

        GL_x[ids] = GL_hat_theta[ids, 0]
```

```
        GL_y[ids] = GL_hat_theta[ids, 1]
        self.GL_interp_mesh_phi = np.arctan2(GL_y, GL_x)

        ct = np.cos(self.GL_interp_mesh_theta)
        st = np.sin(self.GL_interp_mesh_theta)
        Rtheta = np.zeros_like(self.Rtheta)
        Rtheta[:, :, 0, 0] = ct
        Rtheta[:, :, 0, 2] = st
        Rtheta[:, :, 2, 0] = -st
        Rtheta[:, :, 1, 1] = 1
        Rtheta[:, :, 2, 2] = ct

        cp = np.cos(self.GL_interp_mesh_phi)
        sp = np.sin(self.GL_interp_mesh_phi)
        Rphi = np.zeros_like(self.Rphi)
        Rphi[:, :, 0, 0] = cp
        Rphi[:, :, 0, 1] = -sp
        Rphi[:, :, 1, 0] = sp
        Rphi[:, :, 1, 1] = cp
        Rphi[:, :, 2, 2] = 1

        L_interp_hat_theta = np.zeros(
            (self.theta.size, self.phi.size, 3)
        )
        L_interp_hat_phi = np.zeros_like(L_interp_hat_theta)
        L_interp_hat_theta[:, :, 0] = 1
        L_interp_hat_phi[:, :, 1] = 1

        LG_R = np.swapaxes(self.R, 2, 3)
        self.LG_interp_hat_theta = MyMath.rotate(
            MyMath.rotate(
                MyMath.rotate(L_interp_hat_theta, Rtheta), Rphi
            ),
            LG_R,
        )
        self.LG_interp_hat_phi = MyMath.rotate(
            MyMath.rotate(
                MyMath.rotate(L_interp_hat_phi, Rtheta), Rphi
            ),
            LG_R,
        )

    def get_reference_polarization(self):
        sp = np.sin(self.mesh_phi)
        cp = np.cos(self.mesh_phi)
        st = np.sin(self.mesh_theta)
        ct = np.cos(self.mesh_theta)
        hat_i_ref_x = -(1 - ct) * sp * cp
        hat_i_ref_y = 1 - sp * sp * (1 - ct)
        hat_i_ref_z = -st * sp
        hat_i_ref = (
            np.array([hat_i_ref_x, hat_i_ref_y, hat_i_ref_z])
            .swapaxes(0, 1)
            .swapaxes(1, 2)
        )

        hat_i_cross_x = 1 - cp * cp * (1 - ct)
```

```
        hat_i_cross_y = -(1 - ct) * sp * cp
        hat_i_cross_z = -st * cp
        hat_i_cross = (
            np.array([hat_i_cross_x, hat_i_cross_y, hat_i_cross_z])
            .swapaxes(0, 1)
            .swapaxes(1, 2)
        )

        return hat_i_ref, hat_i_cross

    def get_polarization_matrix(self):
        sp = np.sin(self.mesh_phi)
        cp = np.cos(self.mesh_phi)
        st = np.sin(self.mesh_theta)
        ct = np.cos(self.mesh_theta)

        a_11 = 1 - st * st * cp * cp
        a_12 = -st * st * sp * cp
        a_13 = -st * ct * cp
        a_21 = -st * st * sp * cp
        a_22 = 1 - st * st * sp * sp
        a_23 = -st * ct * sp
        a_31 = -st * ct * cp
        a_32 = -st * ct * sp
        a_33 = 1 - ct * ct
        matrix = np.array(
            [
                [a_11, a_12, a_13],
                [a_21, a_22, a_23],
                [a_31, a_32, a_33],
            ]
        )
        return matrix.swapaxes(0, 2).swapaxes(1, 3)

    def calculate_reference_fields(self):
        F = (
            np.array([self.Fx, self.Fy, self.Fz])
            .swapaxes(0, 1)
            .swapaxes(1, 2)
        )
        polarization_matrix = self.get_polarization_matrix()
        E = np.squeeze(polarization_matrix @ F[:, :, :, np.newaxis])

        hat_i_ref, hat_i_cross = self.get_reference_polarization()

        self.Fref[:, :] = np.multiply(E, hat_i_ref).sum(2)
        self.Fcross[:, :] = np.multiply(E, hat_i_cross).sum(2)

    def evaluate_local_field(self):
        self.local_field_flag = False

        self.local_Fphi = np.zeros(self.local_shape)
        self.local_Ftheta = np.zeros(self.local_shape)

        if self.evaluate_as == "isotropic":
            if self.evaluation_arguments["isotropic␣on"] == "both":
                self.local_Ftheta[:] = 0.7071067811865475 # 1/sqrt(2)
```

```
            self.local_Fphi[:] = 0.7071067811865475 # 1/sqrt(2)
        elif self.evaluation_arguments["isotropic␣on"] == "theta":
            self.local_Ftheta[:] = 1
            self.local_Fphi[:] = 0
        elif self.evaluation_arguments["isotropic␣on"] == "phi":
            self.local_Ftheta[:] = 0
            self.local_Fphi[:] = 1
    elif self.evaluate_as == "ideal␣dipole":
        if "dipole␣length" in self.evaluation_arguments.keys():
            self.ideal_dipole(
                self.evaluation_arguments["dipole␣length"]
            )
        else:
            raise Exception(
                "Tried␣to␣calculate␣ideal␣dipole␣"
                + "field␣without␣dipole␣length."
            )
    if self.evaluate_as == "ideal␣loop␣dipole":
        if "loop␣dipole␣area" in self.evaluation_arguments.keys():
            self.ideal_loop_dipole(
                self.evaluation_arguments["loop␣dipole␣area"]
            )
        else:
            raise Exception(
                "Tried␣to␣calculate␣ideal␣dipole␣"
                + "field␣without␣dipole␣length."
            )
    elif self.evaluate_as == "load␣file":
        if "file␣path" in self.evaluation_arguments.keys():
            self.load_file(self.evaluation_arguments["file␣path"])
        else:
            raise Exception(
                "Tried␣to␣load␣file␣" + "without␣file␣path."
            )
    elif self.evaluate_as == "expressions":
        if (
            "expression␣theta" in self.evaluation_arguments.keys()
            and "expression␣phi"
            in self.evaluation_arguments.keys()
        ):
            self.eval_expression(
                self.evaluation_arguments["expression␣theta"],
                self.evaluation_arguments["expression␣phi"],
            )
        else:
            raise Exception(
                "Tried␣to␣evaluate␣expression␣"
                + "without␣expression."
            )

    a = np.absolute(self.local_Fphi)
    b = np.absolute(self.local_Ftheta)
    self.local_F = np.sqrt(a * a + b * b)

    # Normalize local fields
    max_magF = np.max(self.local_F)
    if max_magF > 1e-14:
```

```
            self.local_F = self.local_F / max_magF
            self.local_Ftheta = self.local_Ftheta / max_magF
            self.local_Fphi = self.local_Fphi / max_magF

    def evaluate(self):
        if self.ok:
            return

        t0 = time.time()

        if self.local_mesh_flag:
            self.evaluate_local_mesh()
        if self.R_flag:
            self.evaluate_R()
        if self.Rtheta_flag:
            self.evaluate_Rtheta()
        if self.Rphi_flag:
            self.evaluate_Rphi()
        if self.hats_flag:
            self.evaluate_hats()
        if self.local_field_flag:
            self.evaluate_local_field()
        if self.LG_interp_mesh_flag:
            self.evaluate_LG_interp_mesh()

        fit_points = (self.local_theta, self.local_phi)
        interp_points = (
            self.GL_interp_mesh_theta,
            self.GL_interp_mesh_phi,
        )

        values = self.local_Ftheta.copy()
        if values.dtype == np.dtype("complex64"):
            values = np.array(values, dtype=np.dtype("complex128"))
        interp = RegularGridInterpolator(
            fit_points, values, method="linear"
        )
        Ftheta = interp(interp_points)

        values = self.local_Fphi.copy()
        if values.dtype == np.dtype("complex64"):
            values = np.array(values, dtype=np.dtype("complex128"))
        interp = RegularGridInterpolator(
            fit_points, values, method="linear"
        )
        Fphi = interp(interp_points)

        self.Ftheta = Ftheta * (
            self.LG_interp_hat_theta * self.hat_theta
        ).sum(2) + Fphi * (
            self.LG_interp_hat_phi * self.hat_theta
        ).sum(
            2
        )
        self.Fphi = Ftheta * (
            self.LG_interp_hat_theta * self.hat_phi
        ).sum(2) + Fphi * (self.LG_interp_hat_phi * self.hat_phi).sum(
```

```
            2
        )

        a = np.absolute(self.Fphi)
        b = np.absolute(self.Ftheta)
        self.F = np.sqrt(a * a + b * b)

        # Normalize fields
        max_F = np.max(self.F)
        if max_F>1e-14:
            self.Ftheta = self.Ftheta/max_F
            self.Fphi = self.Fphi/max_F
            self.F = self.F/max_F

        # Circular polarization
        self.Frhcp = (self.Ftheta - 1j * self.Fphi) / MyMath.sqrt2
        self.Flhcp = (self.Ftheta + 1j * self.Fphi) / MyMath.sqrt2

        # Cartesian components fields
        vector_F = (
            self.Ftheta[:, :, np.newaxis] * self.hat_theta
            + self.Fphi[:, :, np.newaxis] * self.hat_phi
        )
        self.Fx = vector_F[:, :, 0]
        self.Fy = vector_F[:, :, 1]
        self.Fz = vector_F[:, :, 2]

        # Reference and Cross fields
        self.calculate_reference_fields()

        self.evaluation_time = time.time() - t0
        self.ok = True
        self.mark_update("evaluated")

    def ideal_dipole(self, L):
        st = np.sin(self.local_mesh_theta)
        ids = st != 0

        A = constants["k"] * L * constants["lam"] / 2

        self.local_Ftheta[ids] = (
            np.cos(A * np.cos(self.local_mesh_theta[ids])) - np.cos(A)
        ) / st[ids]

    def ideal_loop_dipole(self, S):
        pass

    def load_file(self, file_path):
        with open(file_path, "r") as f:
            reader = csv.reader(f)

            variations = []
            phi = []
            theta = []
            rEphi = []
            rEtheta = []
```

```
        self.header = reader.__next__()
        for line in reader:
            variations.append(line[0])
            phi.append(float(line[1]))
            theta.append(float(line[2]))
            a = line[3].split()
            rEphi.append(float(a[0]) * np.exp(1j * float(a[1])))
            a = line[4].split()
            rEtheta.append(float(a[0]) * np.exp(1j * float(a[1])))

        phi = np.array(phi)
        theta = np.array(theta)
        Fphi = (
            4
            * np.pi
            * np.array(rEphi)
            / (1j * constants["eta"] * constants["k"])
        )
        Ftheta = (
            4
            * np.pi
            * np.array(rEtheta)
            / (1j * constants["eta"] * constants["k"])
        )
        idsphi = theta == theta[0]
        idstheta = phi == phi[0]
        phi = phi[idsphi]
        theta = theta[idstheta]
        if self.evaluation_arguments["load␣mesh␣from␣file"]:
            self.local_phi = phi
            self.local_theta = theta
            self.local_mesh_flag = True
            self.local_shape = (
                self.local_theta.size,
                self.local_phi.size,
            )

            self.local_Fphi = np.reshape(
                Fphi, (theta.size, phi.size)
            )
            self.local_Ftheta = np.reshape(
                Ftheta, (theta.size, phi.size)
            )
        else:
            fit_points = (theta, phi)
            interp_points = (
                self.local_mesh_theta,
                self.local_mesh_phi,
            )

            values = np.reshape(Ftheta, (theta.size, phi.size))
            interp = RegularGridInterpolator(
                fit_points, values, method="linear"
            )
            self.local_Ftheta = interp(interp_points)

            values = np.reshape(Fphi, (theta.size, phi.size))
```

```
            interp = RegularGridInterpolator(
                fit_points, values, method="linear"
            )
            self.local_Fphi = interp(interp_points)

        self.evaluation_arguments["loaded file"] = file_path
        self.evaluation_arguments["force reload"] = False

    def eval_expression(self, expression_theta, expression_phi):
        new_vars = {
            "L": self.evaluation_arguments["dipole length"],
            "S": self.evaluation_arguments["loop dipole area"],
            "theta": self.local_mesh_theta,
            "phi": self.local_mesh_phi,
            "mesh_theta": self.local_mesh_theta,
            "mesh_phi": self.local_mesh_phi,
        }
        for key in constants.keys():
            new_vars[key] = constants[key]
        self.local_Ftheta = NEP.eval(
            expression=expression_theta, variables=new_vars
        )
        self.local_Fphi = NEP.eval(
            expression=expression_phi, variables=new_vars
        )
        self.local_Ftheta = np.broadcast_to(
            self.local_Ftheta, self.local_shape
        )
        self.local_Fphi = np.broadcast_to(
            self.local_Fphi, self.local_shape
        )

    def interpolate_at(
        self, interp_mesh_theta_deg, interp_mesh_phi_deg, field
    ):
        fit_points = (self.theta, self.phi)
        interp_points = (interp_mesh_theta_deg, interp_mesh_phi_deg)

        values = field
        if values.dtype == np.dtype("complex64"):
            values = np.array(values, dtype=np.dtype("complex128"))
        interp = RegularGridInterpolator(
            fit_points, values, method="linear"
        )
        interp_field = interp(interp_points)

        return interp_field

    def copy(self):
        antenna = Antenna(
            constants=constants,
            name=self.name,
            current_magnitude=self.current_magnitude,
            current_phase=self.current_phase,
            phi=self.phi.copy(),
            theta=self.theta.copy(),
            elevation=self.elevation,
```

```
            azimuth=self.azimuth,
            roll=self.roll,
            x=self.x,
            y=self.y,
            z=self.z,
            evaluate_as=self.evaluate_as,
        )

        antenna.evaluation_arguments = (
            self.evaluation_arguments.copy()
        )

        antenna.evaluate()
        return antenna


if __name__ == "__main__":
    import tkinter as tk
    import math

    constants = dict()
    constants["c"] = 299792458 # m/s
    constants["f"] = 433e6 # Hz
    constants["eta"] = 120 * math.pi
    constants["lam"] = constants["c"] / constants["f"] # m
    constants["w"] = 2 * math.pi * constants["f"] # rad/s
    constants["k"] = 2 * math.pi / constants["lam"] # rad/m

    antenna = Antenna(constants=constants, name="loaded antenna")
    antenna.evaluate_as = antenna.load_file
    root = tk.Tk()
    root.geometry("0x0")
    file_path = tk.filedialog.askopenfilename(parent=root)
    root.destroy()
    if file_path != "":
        antenna.evaluation_arguments["file path"] = file_path
        antenna.evaluation_arguments["force reload"] = False
        antenna.evaluation_arguments["load mesh from file"] = True
        antenna.evaluate()

    def export_to_file(self, filename):
        with open(filename, mode="wb") as f:
            pickle.dump(self, f)


def load_from_file(file_path, load_mesh_from_file=False, **kw):
    antenna = Antenna(**kw)
    antenna.set_evaluation_method("load file")
    antenna.evaluation_arguments["file path"] = file_path
    antenna.evaluate()

    return antenna
```

Apêndice 9.63 – AntennaEditorFrame.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 5 14:56:38 2023

@author: 160047412
"""

import tkinter as tk
from tkinter import ttk
import numpy as np


class AntennaEditorFrame(tk.Frame):
    def __init__(self,
                 antenna,
                 on_finish=True,
                 on_done=None,
                 on_cancel=None,
                 master=None,
                 **kw):
        tk.Frame.__init__(self, master=master, **kw)
        self.antenna = antenna
        self.on_finish = on_finish
        self.on_done = on_done
        self.on_cancel = on_cancel

        self.init_variables()
        self.init_layout()
        self.radiobutton_change()

    def init_variables(self,):
        self.name = tk.StringVar(value=self.antenna.name)
        self.current_magnitude = tk.DoubleVar(
            value=self.antenna.current_magnitude)
        self.current_phase = tk.DoubleVar(
            value=self.antenna.current_phase)
        self.theta_initial = tk.DoubleVar(
            value=np.min(self.antenna.theta))
        self.theta_final = tk.DoubleVar(
            value=np.max(self.antenna.theta))
        self.theta_N = tk.IntVar(value=self.antenna.theta.size)
        self.phi_initial = tk.DoubleVar(
            value=np.min(self.antenna.phi))
        self.phi_final = tk.DoubleVar(value=np.max(self.antenna.phi))
        self.phi_N = tk.IntVar(value=self.antenna.phi.size)
        self.elevation = tk.DoubleVar(value=self.antenna.elevation)
        self.azimuth = tk.DoubleVar(value=self.antenna.azimuth)
        self.roll = tk.DoubleVar(value=self.antenna.roll)
        self.x = tk.StringVar(value=str(self.antenna.x))
        self.y = tk.StringVar(value=str(self.antenna.y))
        self.z = tk.StringVar(value=str(self.antenna.z))

        if self.antenna.evaluate_as == 'isotropic':
            self.radiobutton_variable = tk.StringVar(
                value='Isotropic')
```

```
        elif self.antenna.evaluate_as == 'ideal dipole':
            self.radiobutton_variable = tk.StringVar(
                value='Ideal dipole')
        elif self.antenna.evaluate_as == 'ideal loop dipole':
            self.radiobutton_variable = tk.StringVar(
                value='Ideal loop dipole')
        elif self.antenna.evaluate_as == 'expressions':
            self.radiobutton_variable = tk.StringVar(
                value='Expressions')
        elif self.antenna.evaluate_as == 'load file':
            self.radiobutton_variable = tk.StringVar(
                value='Load file')

        self.ideal_dipole_length = tk.DoubleVar(
            value=self.antenna.evaluation_arguments['dipole length'])
        self.ideal_loop_dipole_area = tk.DoubleVar(
            value=self.antenna.evaluation_arguments[
                'loop dipole area'])
        self.expression_theta = tk.StringVar(
            value=self.antenna.evaluation_arguments[
                'expression theta'])
        self.expression_phi = tk.StringVar(
            value=self.antenna.evaluation_arguments['expression phi'])
        self.isotropic_radiobutton_variable = tk.StringVar(
            value=self.antenna.evaluation_arguments['isotropic on'])
        self.file_path = tk.StringVar(
            value=self.antenna.evaluation_arguments['file path'])
        self.load_mesh_from_file = tk.IntVar(
            value=self.antenna.evaluation_arguments[
                'load mesh from file'])

    def init_layout(self):
        fr_top = tk.Frame(master=self)
        fr_top.pack(side='top', fill='both')
        fr = ttk.LabelFrame(master=fr_top, text='Antenna name:')
        fr.pack(side='left', fill='both')
        ttk.Entry(master=fr, textvariable=self.name).pack(
            side='left', fill='both')
        fr = ttk.LabelFrame(master=fr_top, text='Current magnitude:')
        fr.pack(side='left', fill='both')
        ttk.Entry(
            master=fr,
            textvariable=self.current_magnitude).pack(
                side='left', fill='both')
        fr = ttk.LabelFrame(master=fr_top, text='Current phase:')
        fr.pack(side='left', fill='both')
        ttk.Entry(master=fr, textvariable=self.current_phase).pack(
            side='left', fill='both')
        tk.Button(
            master=fr_top,
            text='update', command=self._on_update).pack(
                side='left', fill='both')

        fr = ttk.LabelFrame(
            master=self,
            text='Phi angle samples: linspace(start, stop, num)')
        fr.pack(side='top', fill='both')
```

```
        ttk.Label(master=fr, text='Start').grid(row=0, column=0)
        ttk.Label(master=fr, text='Stop').grid(row=0, column=1)
        ttk.Label(master=fr, text='Num').grid(row=0, column=2)
        ttk.Entry(master=fr, textvariable=self.phi_initial).grid(
            row=1, column=0)
        ttk.Entry(master=fr, textvariable=self.phi_final).grid(
            row=1, column=1)
        ttk.Entry(master=fr, textvariable=self.phi_N).grid(
            row=1, column=2)
        fr = ttk.LabelFrame(
            master=self,
            text='Theta␣angle␣samples:␣linspace(start,␣stop,␣num)')
        fr.pack(side='top', fill='both')
        ttk.Label(master=fr, text='Start').grid(row=0, column=0)
        ttk.Label(master=fr, text='Stop').grid(row=0, column=1)
        ttk.Label(master=fr, text='Num').grid(row=0, column=2)
        ttk.Entry(master=fr, textvariable=self.theta_initial).grid(
            row=1, column=0)
        ttk.Entry(master=fr, textvariable=self.theta_final).grid(
            row=1, column=1)
        ttk.Entry(master=fr, textvariable=self.theta_N).grid(
            row=1, column=2)

        fr = tk.Frame(master=self)
        fr.pack(side='top', fill='y')

        fr2 = ttk.LabelFrame(master=fr, text='Orientation')
        fr2.pack(side='left', fill='both')
        fr3 = tk.Frame(master=fr2, border=2)
        fr3.pack(side='top', fill='both')
        ttk.Label(master=fr3, text='Elevation',
                  width=12).pack(side='left', fill='y')
        ttk.Entry(master=fr3, textvariable=self.elevation, width=10,
                  justify="center").pack(side='right', fill='y')
        fr3 = tk.Frame(master=fr2, border=2)
        fr3.pack(side='top', fill='both')
        ttk.Label(master=fr3, text='Azimuth', width=12).pack(
            side='left', fill='y')
        ttk.Entry(master=fr3, textvariable=self.azimuth, width=10,
                  justify="center").pack(side='right', fill='y')
        fr3 = tk.Frame(master=fr2, border=2)
        fr3.pack(side='top', fill='both')
        ttk.Label(master=fr3, text='Roll', width=12).pack(
            side='left', fill='y')
        ttk.Entry(master=fr3, textvariable=self.roll, width=10,
                  justify="center").pack(side='right', fill='y')

        fr2 = ttk.LabelFrame(master=fr, text='Position')
        fr2.pack(side='top', fill='both')
        fr3 = tk.Frame(master=fr2, border=2)
        fr3.pack(side='top', fill='both')
        ttk.Label(master=fr3, text='X', width=3).pack(
            side='left', fill='y')
        ttk.Entry(master=fr3, textvariable=self.x, width=10,
                  justify="center").pack(side='right', fill='y')
        fr3 = tk.Frame(master=fr2, border=2)
        fr3.pack(side='top', fill='both')
```

```
ttk.Label(master=fr3, text='Y', width=3).pack(
    side='left', fill='y')
ttk.Entry(master=fr3, textvariable=self.y, width=10,
          justify="center").pack(side='right', fill='y')
fr3 = tk.Frame(master=fr2, border=2)
fr3.pack(side='top', fill='both')
ttk.Label(master=fr3, text='Z', width=3).pack(
    side='left', fill='both')
ttk.Entry(master=fr3, textvariable=self.z, width=10,
          justify="center").pack(side='right', fill='both')

fr2 = ttk.LabelFrame(master=fr, text='Radiation␣pattern␣(F)')
fr2.pack(side=tk.TOP, fill=tk.BOTH)
fr = tk.Frame(master=self)
fr.pack(side='top', fill='both')
radiobutton_frame = tk.LabelFrame(master=fr, text='Options')
radiobutton_frame.pack(side=tk.LEFT, fill=tk.BOTH)
tk.Radiobutton(
    master=radiobutton_frame,
    text='Isotropic', value='Isotropic',
    variable=self.radiobutton_variable,
    command=self.radiobutton_change
    ).pack(side=tk.TOP, fill=tk.BOTH)
tk.Radiobutton(
    master=radiobutton_frame, text='Ideal␣dipole',
    value='Ideal␣dipole',
    variable=self.radiobutton_variable,
    command=self.radiobutton_change
    ).pack(side=tk.TOP, fill=tk.BOTH)
tk.Radiobutton(
    master=radiobutton_frame, text='Ideal␣loop␣dipole',
    value='Ideal␣loop␣dipole',
    variable=self.radiobutton_variable,
    command=self.radiobutton_change
    ).pack(side=tk.TOP, fill=tk.BOTH)
tk.Radiobutton(
    master=radiobutton_frame,
    text='Expressions', value='Expressions',
    variable=self.radiobutton_variable,
    command=self.radiobutton_change
    ).pack(side=tk.TOP, fill=tk.BOTH)
tk.Button(master=radiobutton_frame, text='Load␣from␣file...',
          command=self.load_file).pack(
              side=tk.TOP, fill=tk.BOTH)
electric_field_frame = tk.Frame(master=fr)
electric_field_frame.pack(
    side=tk.LEFT, expand=True, fill=tk.BOTH)
self.isotropic_frame = tk.LabelFrame(
    master=electric_field_frame, text='Isotropic␣antenna')
tk.Radiobutton(
    master=self.isotropic_frame, text='Isotropic␣on␣both',
    value='both',
    variable=self.isotropic_radiobutton_variable).pack(
        side=tk.TOP, fill=tk.BOTH)
tk.Radiobutton(
    master=self.isotropic_frame, text='Isotropic␣on␣theta',
    value='theta',
```

```
            variable=self.isotropic_radiobutton_variable).pack(
                side=tk.TOP, fill=tk.BOTH)
        tk.Radiobutton(
            master=self.isotropic_frame,
            text='Isotropic␣on␣phi', value='phi',
            variable=self.isotropic_radiobutton_variable).pack(
                side=tk.TOP, fill=tk.BOTH)
        self.ideal_dipole_frame = tk.LabelFrame(
            master=electric_field_frame,
            text='Ideal␣dipole␣length␣(L)')
        self.ideal_loop_dipole_frame = tk.LabelFrame(
            master=electric_field_frame,
            text='Ideal␣loop␣dipole␣area␣(S)')
        self.expression_frame = tk.LabelFrame(
            master=electric_field_frame,text='Expressions')
        self.load_file_frame = tk.LabelFrame(
            master=electric_field_frame, text='Load␣file')
        ttk.Label(master=self.expression_frame,
                  text='Expression␣theta').grid(row=0, column=0)
        ttk.Label(master=self.expression_frame,
                  text='Expression␣phi').grid(row=1, column=0)
        ttk.Entry(
            master=self.ideal_dipole_frame,
            textvariable=self.ideal_dipole_length).pack(
                side='left', fill='x')
        ttk.Entry(
            master=self.ideal_loop_dipole_frame,
            textvariable=self.ideal_loop_dipole_area
            ).pack(side='left', fill='x')
        ttk.Entry(
            master=self.expression_frame,
            textvariable=self.expression_theta).grid(row=0, column=1)
        ttk.Entry(
            master=self.expression_frame,
            textvariable=self.expression_phi).grid(row=1, column=1)
        ttk.Entry(
            master=self.load_file_frame,
            textvariable=self.file_path).pack(
                side='top', expand=True, fill='x')
        tk.Checkbutton(
            master=self.load_file_frame, text='Load␣mesh␣from␣file',
            variable=self.load_mesh_from_file).pack(
                side='bottom', fill='both')
        if self.on_finish:
            fr = ttk.LabelFrame(master=self, text='Finish')
            fr.pack(side='top', fill='both')
            ttk.Button(
                master=fr,
                text='Done', command=self._on_done).pack(
                    side=tk.LEFT, fill=tk.BOTH)
            ttk.Button(
                master=fr,
                text='Cancel',
                command=self.on_cancel).pack(
                    side=tk.LEFT, fill=tk.BOTH)

    def _on_update(self):
```

```
        if self.radiobutton_variable.get() == 'Isotropic':
            evaluate_as = 'isotropic'
        elif self.radiobutton_variable.get() == 'Ideal dipole':
            evaluate_as = 'ideal dipole'
        elif self.radiobutton_variable.get() == 'Ideal loop dipole':
            evaluate_as = 'ideal loop_dipole'
        elif self.radiobutton_variable.get() == 'Expressions':
            evaluate_as = 'expressions'
        elif self.radiobutton_variable.get() == 'Load file':
            evaluate_as = 'load file'
        self.antenna.set_evaluation_method(evaluate_as)
        self.antenna.evaluation_arguments['dipole length'] = \
            self.ideal_dipole_length.get()
        self.antenna.evaluation_arguments['loop dipole area'] = \
            self.ideal_loop_dipole_area.get()
        self.antenna.evaluation_arguments['expression theta'] = \
            self.expression_theta.get()
        self.antenna.evaluation_arguments['expression phi'] = \
            self.expression_phi.get()
        self.antenna.evaluation_arguments['isotropic on'] = \
            self.isotropic_radiobutton_variable.get()
        self.antenna.evaluation_arguments['file path'] = \
            self.file_path.get()
        self.antenna.evaluation_arguments['force reload'] = True
        self.antenna.evaluation_arguments['load mesh from file'] = \
            self.load_mesh_from_file.get()

        self.antenna.set_current(
            magnitude=self.current_magnitude.get(),
            phase=self.current_phase.get())

        self.antenna.set_position(
            x=self.x.get(), y=self.y.get(), z=self.z.get())

        self.antenna.set_orientation(elevation=self.elevation.get(),
                                     azimuth=self.azimuth.get(),
                                     roll=self.roll.get())

        self.antenna.resample(
            theta=np.linspace(
                self.theta_initial.get(),
                self.theta_final.get(),
                self.theta_N.get()),
            phi=np.linspace(
                self.phi_initial.get(),
                self.phi_final.get(),
                self.phi_N.get()))

        self.antenna.set_name(name=self.name.get())

        self.antenna.evaluate()

    def _on_done(self):
        self._on_update()
        if self.on_done is not None:
            self.on_done()
```

```
    def radiobutton_change(self):
        self.isotropic_frame.pack_forget()
        self.ideal_dipole_frame.pack_forget()
        self.ideal_loop_dipole_frame.pack_forget()
        self.expression_frame.pack_forget()
        self.load_file_frame.pack_forget()
        if self.radiobutton_variable.get() == 'Isotropic':
            self.isotropic_frame.pack(side=tk.LEFT, fill=tk.X)
        elif self.radiobutton_variable.get() == 'Ideal␣dipole':
            self.ideal_dipole_frame.pack(side=tk.LEFT, fill=tk.X)
        elif self.radiobutton_variable.get() == 'Ideal␣loop␣dipole':
            self.ideal_loop_dipole_frame.pack(side=tk.LEFT, fill=tk.X)
        elif self.radiobutton_variable.get() == 'Expressions':
            self.expression_frame.pack(side=tk.LEFT, fill=tk.X)
        elif self.radiobutton_variable.get() == 'Load␣file':
            self.load_file_frame.pack(side='left',
                                      expand=True,
                                      fill='x')

    def load_file(self):
        self.ideal_dipole_frame.pack_forget()
        self.ideal_loop_dipole_frame.pack_forget()
        self.expression_frame.pack_forget()
        self.load_file_frame.pack_forget()
        self.radiobutton_variable.set('Load␣file')
        file_path = tk.filedialog.askopenfilename(
            initialfile=self.file_path.get())
        if file_path != '':
            self.file_path.set(file_path)
        self.load_file_frame.pack(side='left', expand=True, fill='x')


if __name__ == "__main__":
    from Antenna import Antenna

    constants = dict()
    constants['c'] = 299792458 # m/s
    constants['f'] = 433e6 # Hz
    constants['eta'] = 120*np.pi
    constants['lam'] = constants['c']/constants['f'] # m
    constants['w'] = 2*np.pi*constants['f'] # rad/s
    constants['k'] = 2*np.pi/constants['lam'] # rad/m

    antenna = Antenna(constants=constants)

    root = tk.Tk()

    def on_done():
        print('done')
        root.destroy()

    def on_cancel():
        print('cancel')
        root.destroy()

    AntennaEditorFrame(
        antenna=antenna,
```

```
        on_done=on_done,
        on_cancel=on_cancel,
        master=root).pack(side=tk.LEFT, fill=tk.BOTH)
    root.mainloop()
```

Apêndice 9.64 – App.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 26 01:30:47 2023

@author: Vítor Lima Aguirra
"""

import tkinter as tk
import tkinter.filedialog
from tkinter import ttk
import numpy as np

import ProjectTreeview

class App(tk.Tk):
    def __init__(self,
                 antennas=[],
                 analyses=[],
                 results=[],
                 ):
        tk.Tk.__init__(self)
        self.title('EletromagArrays')
        self.name = 'New project'
        self.updater_id = None

        self.init_variables()
        self.init_layout()

        self.constants = dict()
        self.evaluate_constants()

        self.antennas = []
        self.analyses = []
        self.optims = []
        self.result_tabs = []

        color = '#303030'
        self.configure(background=color)
        style = ttk.Style(self)
        # set ttk theme to "clam" which support the
        # fieldbackground option
        style.theme_use("clam")
        style.configure("Treeview",
                        background=color,
                        fieldbackground=color,
                        foreground="white")

        if len(antennas) != 0:
            self.add_antennas(antennas)

    def init_variables(self):
        self.frequency = tk.DoubleVar(value=433e6)

    def evaluate_constants(self):
        self.constants['c'] = 299792458 # m/s
```

```
        self.constants['f'] = self.frequency.get() # Hz
        self.constants['eta'] = 120*np.pi
        self.constants['lam'] = self.constants['c'] / \
            self.constants['f'] # m
        self.constants['w'] = 2*np.pi * \
            self.constants['f'] # rad/s
        self.constants['k'] = 2*np.pi / \
            self.constants['lam'] # rad/m

    def mark_update(self, event):
        for l in self.constants_listeners:
            l.notify(self)

    def init_layout(self):
        self.treeview = ProjectTreeview.ProjectTreeview(
            self, master=self)
        self.treeview.pack(side='left', fill='both')

        self.tabs = ttk.Notebook(master=self)
        self.tabs.pack(side='top', fill='both')

    def add_antenna(self, antenna):
        self.antennas.append(antenna)
        self.treeview.add_antenna(antenna)

    def add_analysis(self, analysis):
        self.analyses.append(analysis)
        self.treeview.add_analysis(analysis)

    def add_optim(self, optim):
        self.optims.append(optim)
        self.treeview.add_optim(optim)

    def add_tab(self, tab):
        self.result_tabs.append(tab)
        self.treeview.add_tab(tab)
        self.tabs.add(tab, text=tab.name)
        for result in tab.results:
            self.add_result(tab, result)

    def add_result(self, tab, result):
        self.treeview.add_result(tab, result)

    def remove_antenna(self, antenna):
        self.antennas.remove(antenna)
        self.treeview.remove_antenna(antenna)

    def remove_analysis(self, analysis):
        self.analyses.remove(analysis)
        self.treeview.remove_analysis(analysis)

    def remove_optim(self, optim):
        self.optims.remove(optim)
        self.treeview.remove_optim(optim)

    def remove_tab(self, tab):
        self.tabs.remove(tab)
```

```
        self.treeview.remove_tab(tab)

    def remove_result(self, tab, result):
        tab.results.remove(result)
        self.treeview.remove_result(tab, result)

    def new_file(self):
        print("new␣file")

    def open_file(self):
        print("open␣file")

    def save_file(self):
        print("save␣file")

    def save_file_as(self):
        print("save␣file␣as")

    def close_file(self):
        print("close␣file")

    def edit_constants(self):
        root = tk.Toplevel()
        fr = ttk.LabelFrame(
            master=root, text='Speed␣of␣light')
        fr.pack(side='top', fill='both')
        tk.Label(
            master=fr,
            text='299792458m/s').pack(side='top')
        fr = ttk.LabelFrame(
            master=root, text='Frequency')
        fr.pack(side='top', fill='both')
        ttk.Entry(master=fr,
                  textvariable=self.frequency).pack(
                      side='top')
        fr = ttk.LabelFrame(master=root, text='Eta')
        fr.pack(side='top', fill='both')
        tk.Label(master=fr, text='%.2f' %
                 self.constants['eta']).pack(side='top')
        fr = ttk.LabelFrame(master=root, text='Lambda')
        fr.pack(side='top', fill='both')
        tk.Label(master=fr, text='%.2f' %
                 self.constants['lam']).pack(side='top')
        fr = ttk.LabelFrame(master=root, text='w')
        fr.pack(side='top', fill='both')
        tk.Label(master=fr, text='%.2f' %
                 self.constants['w']).pack(side='top')
        fr = ttk.LabelFrame(master=root, text='k')
        fr.pack(side='top', fill='both')
        tk.Label(master=fr, text='%.2f' %
                 self.constants['k']).pack(side='top')
        fr = tk.Frame(master=root)
        fr.pack(side='top', fill='both')
        ttk.Button(master=fr, text='Close',
                   command=root.destroy).pack(side='top')
        root.mainloop()
        self.evaluate_constants()
```

```
        self.mark_update('constants update')

    def mainloop(self):
        self.updater_id = self.after(100, self.updater_thread)
        tk.Tk.mainloop(self)

    def updater_thread(self):
        for antenna in [antenna
                for antenna
                in self.antennas
                if str(type(antenna)) == \
                    "<class: 'Antenna.Antenna'>"]:
            if not antenna.ok:
                antenna.evaluate()

        for array in [antenna
                  for antenna in \
                  self.antennas if \
                  str(type(antenna)) == \
                  "<class: 'Array.Array'>"]:
            if not array.ok:
                array.evaluate()

        for tab in self.result_tabs:
            if not tab.ok:
                tab.update()
        self.updater_id = self.after(2000,self.updater_thread)

    def add_antennas(self, antennas):
        for antenna in antennas:
            self.add_antenna(antenna)

    # def add_result(self, tab, result_struct):
            #

    def on_closing(self):
        self.is_alive = False
        if self.updater_id is not None:
            print("cancelling thread")
            self.after_cancel(self.updater_id)
        print("Destroying")
        tk.Tk.destroy(self)


if __name__ == "__main__":

    import sys
    import os

    import header

    # Create the application
    app = App()
    try:
        import LoadHFSSYagis
        import LoadHFSSValidationArrays
        import LoadValidationArrays
```

```
Ntheta = 91
Nphi = 91

antennas = LoadHFSSYagis.run(
    Ntheta=Ntheta, Nphi=Nphi, elevation=-90)
antennas.update( \
    LoadHFSSValidationArrays.run(
        Ntheta=Ntheta, Nphi=Nphi))
antennas.update( \
    LoadValidationArrays.run(
        Ntheta=Ntheta, Nphi=Nphi))

print('Evaluation␣time␣of␣1Y-4El:␣' + \
     str(round(antennas['array_validation_1Y4EL'].\
              evaluation_time*100000)/100) + 'ms')
print('Evaluation␣time␣of␣2Y-4El:␣' + \
     str(round(antennas['array_validation_2Y4EL'].\
              evaluation_time*100000)/100) + 'ms')
print('Evaluation␣time␣of␣3Y-4El:␣' + \
     str(round(antennas['array_validation_3Y4EL'].\
              evaluation_time*100000)/100) + 'ms')
print('Evaluation␣time␣of␣4Y-4El:␣' + \
     str(round(antennas['array_validation_4Y4EL'].\
              evaluation_time*100000)/100) + 'ms')
print('Evaluation␣time␣of␣5Y-4El:␣' + \
     str(round(antennas['array_validation_5Y4EL'].\
              evaluation_time*100000)/100) + 'ms')

for antenna in antennas.values():
    app.add_antenna(antenna)

# import ValidationHFSS
# ValidationHFSS.run(app=app, antennas=antennas)

import Array

Ntheta=91
Nphi=181

theta=np.linspace(0, 180, Ntheta)
phi=np.linspace(-180, 180, Nphi)

array = Array.Array(name='Custom',
                    theta=theta.copy(),
                    phi=phi.copy(),
                    antennas=[
                        antennas['hfss_yagi4EL'].copy(),
                        antennas['hfss_yagi4EL'].copy(),
                        ])
array.antennas[0].set_position(x=0,y=-0.25,z=0)
array.antennas[0].set_orientation(
    elevation=-90,azimuth=0,roll=0)
array.antennas[0].set_current(
    magnitude=1.0,
    phase=0)
array.antennas[1].set_position(x=0,y=0.25,z=0)
```

```
array.antennas[1].set_orientation(
    elevation=-90,azimuth=0,roll=90)
array.antennas[1].set_current(
    magnitude=1.0,
    phase=90)
array.evaluate()

app.add_antenna(array)

import ResultFrame
import Result

Ntheta = 21
Nphi = 21
field = 'Fref'
# plot = '2d Polar Patch Type 2'
plot = '3d␣Surface'

tab = ResultFrame.ResultFrame(
    master=app.tabs,
    name='Tab',
    columns=2,rows=1)
Result.Result(tab=tab,
             title='2␣Yagis␣4␣Elements',
             name='2␣Yagis␣4␣Elements',
             antenna=array,
             field=field,
             plot=plot,
             ticks_flag=False,
             in_dB=True,
             column=1,row=1,
             Ntheta=Ntheta,
             Nphi=Nphi)
Result.Result(tab=tab,
             title='HFSS␣5Y4EL',
             name='HFSS␣5Y4EL',
             antenna=antennas['HFSS_5Y4EL'],
             field=field,
             plot=plot,
             ticks_flag=False,
             in_dB=True,
             column=2,row=1,
             Ntheta=Ntheta,
             Nphi=Nphi)
app.add_tab(tab)

# plot = '2d Polar Patch'

# field = 'F'
# tab = ResultFrame.ResultFrame(
# master=app.tabs,
# name=field,
# columns=1,rows=1)
# Result.Result(tab=tab,
# title='Custom Array',
# name='Custom Array',
# antenna=array,
```

```
        # field=field,
        # plot=plot,
        # ticks_flag=False,
        # in_dB=True,
        # column=1,row=1,
        # Ntheta=Ntheta,
        # Nphi=Nphi)
        # app.add_tab(tab)

        # plot = '2d Polar Patch Type 2'

        # field = 'Fref'
        # tab = ResultFrame.ResultFrame(
        # master=app.tabs,
        # name=field,
        # columns=1,rows=1)
        # Result.Result(tab=tab,
        # title='Yagi 2 Elements',
        # name='Yagi 2 Elements',
        # antenna=antennas['hfss_yagi2EL'],
        # field=field,
        # plot=plot,
        # ticks_flag=False,
        # in_dB=True,
        # column=1,row=1,
        # Ntheta=Ntheta,
        # Nphi=Nphi)
        # app.add_tab(tab)

        # field = 'Fcross'
        # tab = ResultFrame.ResultFrame(
        # master=app.tabs,
        # name=field,
        # columns=1,rows=1)
        # Result.Result(tab=tab,
        # title='Custom',
        # name='Custom',
        # antenna=array,
        # field=field,
        # plot=plot,
        # ticks_flag=False,
        # in_dB=True,
        # column=1,row=1,
        # Ntheta=Ntheta,
        # Nphi=Nphi)
        # app.add_tab(tab)

        # field = 'Frhcp'
        # tab = ResultFrame.ResultFrame(
        # master=app.tabs,
        # name=field,
        # columns=1,rows=1)
        # Result.Result(tab=tab,
        # title='Custom',
        # name='Custom',
        # antenna=array,
        # field=field,
```

```
        # plot=plot,
        # ticks_flag=False,
        # in_dB=True,
        # column=1,row=1,
        # Ntheta=Ntheta,
        # Nphi=Nphi)
        # app.add_tab(tab)

        # field = 'Flhcp'
        # tab = ResultFrame.ResultFrame(
        # master=app.tabs,
        # name=field,
        # columns=1,rows=1)
        # Result.Result(tab=tab,
        # title='Custom',
        # name='Custom',
        # antenna=array,
        # field=field,
        # plot=plot,
        # ticks_flag=False,
        # in_dB=True,
        # column=1,row=1,
        # Ntheta=Ntheta,
        # Nphi=Nphi)
        # app.add_tab(tab)

        # Main application loop
        app.mainloop()
    except Exception as e:
        # If some error occur, destroy the application to close the
        # window, then show the error
        app.destroy()
        raise e
```

Apêndice 9.65 – AppValidation1.py

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 24 13:23:36 2023

@author: 160047412
"""

import sys
import os

import header

import numpy as np

import Antenna
import Array

theta=np.linspace(0, 180, 91)
phi=np.linspace(-180, 180, 91)

file_name = 'antenna-Yagi-4Elements.csv'
file_path = os.path.join(header.antennas_dir, file_name)
hfss_yagi4EL = Antenna.load_from_file(file_path,
                              name='Yagi 4EL',
                              theta=theta,
                              phi=phi,
                              load_mesh_from_file=False)

array_1Y4EL = Array.Array(
    name='Array 1Y4El',
    theta=theta.copy(),
    phi=phi.copy(),
    antennas=[hfss_yagi4EL])
array_1Y4EL.evaluate()

file_name = '1Y-4EL.csv'
file_path = os.path.join(header.antennas_dir, file_name)
hfss_yagi4EL = Antenna.load_from_file(file_path,
                                  name='HFSS Y4EL',
                                  theta=theta,
                                  phi=phi,
                                  load_mesh_from_file=False)

import App
import ResultFrame
import Result

app = App.App()

app.add_antenna(hfss_yagi4EL)
app.add_antenna(array_1Y4EL)
app.add_antenna(hfss_yagi4EL)

plot='2d Polar Patch Type 2'
field='Ftheta'
```

```
color='Color␣by␣phase'
in_dB = False

tab_1 = ResultFrame.ResultFrame(
    master=app.tabs,
    name='Phase',
    columns=2)
result_1 = Result.Result(tab=tab_1,
                         name='Array',
                         title='Array',
                         antenna=array_1Y4EL,
                         plot=plot,
                         field=field,
                         color=color,
                         in_dB=in_dB,
                         position=1)
result_2 = Result.Result(tab=tab_1,
                         name='HFSS',
                         title='HFSS',
                         antenna=hfss_yagi4EL,
                         plot=plot,
                         field=field,
                         color=color,
                         in_dB=in_dB,
                         position=2)
app.add_tab(tab_1)

field='F'
color='Color␣by␣magnitude'
tab_2 = ResultFrame.ResultFrame(
    master=app.tabs,
    name='Magnitude',
    columns=2)
result_1 = Result.Result(tab=tab_2,
                         name='Array',
                         title='Array',
                         antenna=array_1Y4EL,
                         plot=plot,
                         field=field,
                         color=color,
                         in_dB=in_dB,
                         position=1)
result_2 = Result.Result(tab=tab_2,
                         name='HFSS',
                         title='HFSS',
                         antenna=hfss_yagi4EL,
                         plot=plot,
                         field=field,
                         color=color,
                         in_dB=in_dB,
                         position=2)
app.add_tab(tab_2)

app.mainloop()
```

Apêndice 9.66 – AppValidation2.py

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 24 14:30:55 2023

@author: 160047412
"""

import sys
import os

import header

import numpy as np

import Antenna
import Array

theta=np.linspace(0, 180, 91)
phi=np.linspace(-180, 180, 91)

file_name = 'antenna-Yagi-4Elements.csv'
file_path = os.path.join(header.antennas_dir, file_name)
hfss_yagi4EL = Antenna.load_from_file(file_path,
                              name='Yagi␣4EL',
                              theta=theta,
                              phi=phi,
                              load_mesh_from_file=False)
hfss_yagi4EL.set_position(x=0.0,y=0.0,z=0)

array_2Y4EL = Array.Array(
    name='Array␣2Y4El',
    theta=theta.copy(),
    phi=phi.copy(),
    antennas=[
        Antenna.load_from_file(
            file_path,
            name='Yagi␣4EL',
            theta=theta,
            phi=phi,
            load_mesh_from_file=False),
        Antenna.load_from_file(
            file_path,
            name='Yagi␣4EL',
            theta=theta,
            phi=phi,
            load_mesh_from_file=False),
        ])
array_2Y4EL.antennas[0].set_orientation(
    roll=0, elevation=0, azimuth=0)
array_2Y4EL.antennas[0].set_position(x=0.0, y=0.0, z=0)
array_2Y4EL.antennas[1].set_orientation(
    roll=0, elevation=0, azimuth=45)
array_2Y4EL.antennas[1].set_position(x=0.0, y=1.5, z=0)
## Information for all 5 yagis (as a backup)
# array_2Y4EL.antennas[2].set_orientation(
```

```
# roll=0, elevation=-45, azimuth=-45)
# array_2Y4EL.antennas[2].set_position(x=0, y=-1.5, z=0)
# array_2Y4EL.antennas[3].set_orientation(
# roll=0, elevation=-45, azimuth=135)
# array_2Y4EL.antennas[3].set_position(x=0.34, y=-3.14, z=1.423)
# array_2Y4EL.antennas[4].set_orientation(
# roll=0, elevation=72, azimuth=14)
# array_2Y4EL.antennas[4].set_position(x=0.83, y=1.19, z=-0.72)
array_2Y4EL.evaluate()

file_name = '2Y-4EL.csv'
file_path = os.path.join(header.antennas_dir, file_name)
hfss_2Y4EL = Antenna.load_from_file(file_path,
                                    name='HFSS␣Y4EL',
                                    theta=theta,
                                    phi=phi,
                                    load_mesh_from_file=False)

import App
import ResultFrame
import Result

app = App.App()

app.add_antenna(hfss_yagi4EL)
app.add_antenna(array_2Y4EL)
app.add_antenna(hfss_2Y4EL)

plot='2d␣Polar␣Patch␣Type␣2'
field='Ftheta'
color='Color␣by␣phase'
in_dB = False

tab_1 = ResultFrame.ResultFrame(
    master=app.tabs,
    name='Phase',
    columns=2)
result_1 = Result.Result(tab=tab_1,
                         name='Array',
                         title='Array',
                         antenna=array_2Y4EL,
                         Ntheta=91,
                         Nphi=91,
                         plot=plot,
                         field=field,
                         color=color,
                         in_dB=in_dB,
                         position=1)
result_2 = Result.Result(tab=tab_1,
                         name='HFSS',
                         title='HFSS',
                         antenna=hfss_2Y4EL,
                         Ntheta=91,
                         Nphi=91,
                         plot=plot,
                         field=field,
                         color=color,
```

```
                        in_dB=in_dB,
                        position=2)
app.add_tab(tab_1)

field='F'
color='Color␣by␣magnitude'
tab_2 = ResultFrame.ResultFrame(
    master=app.tabs,
    name='Magnitude',
    columns=2)
result_1 = Result.Result(tab=tab_2,
                        name='Array',
                        title='Array',
                        antenna=array_2Y4EL,
                        Ntheta=91,
                        Nphi=91,
                        plot=plot,
                        field=field,
                        color=color,
                        in_dB=in_dB,
                        position=1)
result_2 = Result.Result(tab=tab_2,
                        name='HFSS',
                        title='HFSS',
                        antenna=hfss_2Y4EL,
                        Ntheta=91,
                        Nphi=91,
                        plot=plot,
                        field=field,
                        color=color,
                        in_dB=in_dB,
                        position=2)
app.add_tab(tab_2)

app.mainloop()
```

Apêndice 9.67 – AppValidation5.py

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 24 22:17:11 2023

@author: 160047412
"""
import sys
import os

import header

import numpy as np

import Antenna
import Array

theta=np.linspace(0, 180, 91)
phi=np.linspace(-180, 180, 91)

file_name = 'antenna-Yagi-4Elements.csv'
file_path = os.path.join(header.antennas_dir, file_name)

hfss_yagi4EL = Antenna.load_from_file(
    file_path,
    name='Yagi␣4EL',
    theta=theta,
    phi=phi,
    load_mesh_from_file=False)
hfss_yagi4EL.set_position(x=0.0,y=0.0,z=0)

array_5Y4EL = Array.Array(
    name='Array␣2Y4El',
    theta=theta.copy(),
    phi=phi.copy(),
    antennas=[Antenna.load_from_file(
        file_path,
        name='Yagi␣4EL',
        theta=theta,
        phi=phi,
        load_mesh_from_file=False),
        Antenna.load_from_file(
            file_path,
            name='Yagi␣4EL',
            theta=theta,
            phi=phi,
            load_mesh_from_file=False),
        Antenna.load_from_file(
            file_path,
            name='Yagi␣4EL',
            theta=theta,
            phi=phi,
            load_mesh_from_file=False),
        Antenna.load_from_file(
            file_path,
            name='Yagi␣4EL',
```

```
            theta=theta,
            phi=phi,
            load_mesh_from_file=False),
        Antenna.load_from_file(
            file_path,
            name='Yagi 4EL',
            theta=theta,
            phi=phi,
            load_mesh_from_file=False),
        ])
array_5Y4EL.antennas[0].set_orientation(
    roll=0, elevation=0, azimuth=0)
array_5Y4EL.antennas[0].set_position(x=0.0, y=0.0, z=0)
array_5Y4EL.antennas[1].set_orientation(
    roll=0, elevation=0, azimuth=45)
array_5Y4EL.antennas[1].set_position(x=0.0, y=1.5, z=0)
array_5Y4EL.antennas[2].set_orientation(
    roll=0, elevation=-45, azimuth=-45)
array_5Y4EL.antennas[2].set_position(x=0, y=-1.5, z=0)
array_5Y4EL.antennas[3].set_orientation(
    roll=0, elevation=-45, azimuth=135)
array_5Y4EL.antennas[3].set_position(x=0.34, y=-3.14, z=1.423)
array_5Y4EL.antennas[4].set_orientation(
    roll=0, elevation=72, azimuth=14)
array_5Y4EL.antennas[4].set_position(x=0.83, y=1.19, z=-0.72)
array_5Y4EL.evaluate()

file_name = '5Y-4EL.csv'
file_path = os.path.join(header.antennas_dir, file_name)
hfss_5Y4EL = Antenna.load_from_file(file_path,
                                    name='HFSS Y4EL',
                                    theta=theta,
                                    phi=phi,
                                    load_mesh_from_file=False)

import App
import ResultFrame
import Result

app = App.App()

app.add_antenna(hfss_yagi4EL)
app.add_antenna(array_5Y4EL)
app.add_antenna(hfss_5Y4EL)

plot='2d Polar Patch Type 1'
field='Fref'
color='Color by phase'
in_dB = True

tab_1 = ResultFrame.ResultFrame(
    master=app.tabs,
    name='Phase',
    columns=2)
result_1 = Result.Result(tab=tab_1,
                         name='Array',
                         title='Array',
```

```
                        Ntheta=91,
                        Nphi=91,
                        antenna=array_5Y4EL,
                        plot=plot,
                        field=field,
                        color=color,
                        in_dB=in_dB,
                        position=1)
result_2 = Result.Result(tab=tab_1,
                        name='HFSS',
                        title='HFSS',
                        Ntheta=91,
                        Nphi=91,
                        antenna=hfss_5Y4EL,
                        plot=plot,
                        field=field,
                        color=color,
                        in_dB=in_dB,
                        position=2)
app.add_tab(tab_1)

field='Fref'
color='Color␣by␣magnitude'
tab_2 = ResultFrame.ResultFrame(
    master=app.tabs,
    name='Magnitude',
    columns=2)
result_1 = Result.Result(tab=tab_2,
                        name='Array',
                        title='Array',
                        Ntheta=91,
                        Nphi=91,
                        antenna=array_5Y4EL,
                        plot=plot,
                        field=field,
                        color=color,
                        in_dB=in_dB,
                        position=1)
result_2 = Result.Result(tab=tab_2,
                        name='HFSS',
                        title='HFSS',
                        Ntheta=91,
                        Nphi=91,
                        antenna=hfss_5Y4EL,
                        plot=plot,
                        field=field,
                        color=color,
                        in_dB=in_dB,
                        position=2)
app.add_tab(tab_2)

app.mainloop()
```

Apêndice 9.68 – Array.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 26 17:49:39 2023

@author: 160047412
"""

import numpy as np

import Antenna
import MyMath
import ArrayEditorFrame


class Array(Antenna.Antenna):
    EditorFrame = ArrayEditorFrame.ArrayEditorFrame

    def __init__(self, antennas=None,
                 constants=None,
                 x_mirror=False,
                 y_mirror=False,
                 z_mirror=False,
                 current_mirror=False,
                 azimuth_symmetry=False,
                 **kw):
        if 'name' not in kw.keys():
            kw['name'] = 'new␣array'
        Antenna.Antenna.__init__(self, **kw)

        if antennas is not None:
            self.antennas = antennas
        else:
            self.antennas = []
        self.x_mirror = x_mirror
        self.y_mirror = y_mirror
        self.z_mirror = z_mirror
        self.current_mirror = current_mirror
        self.azimuth_symmetry = azimuth_symmetry

    def add_antenna(self, antenna):
        self.antennas.append(antenna)
        antenna.listeners.append(self)
        self.mark_update('antenna␣added')

    def evaluate_local_field(self):
        if len(self.antennas) == 0:
            return

        self.local_field_flag = False

        self.local_Fphi = np.zeros(self.local_shape)
        self.local_Ftheta = np.zeros(self.local_shape)

        for antenna in self.antennas:
            antenna.evaluate()
```

```
        Ftheta = antenna.interpolate_at(
            np.degrees(self.local_mesh_theta),
            np.degrees(self.local_mesh_phi), antenna.Ftheta)

        Fphi = antenna.interpolate_at(
            np.degrees(self.local_mesh_theta),
            np.degrees(self.local_mesh_phi), antenna.Fphi)

        x = antenna.x
        y = antenna.y
        z = antenna.z
        phase = np.radians(antenna.current_phase)
        current = antenna.current_magnitude*(np.cos(phase) +
                                        1j*np.sin(phase))

        p0 = Antenna.constants['lam'] *\
            np.array([x, y, z]).reshape((1, 1, 3))
        Af = current*np.exp((1j)*Antenna.constants['k'] *
                            (self.local_hat_k*p0).sum(2))

        self.local_Ftheta = self.local_Ftheta + Af*Ftheta
        self.local_Fphi = self.local_Fphi + Af*Fphi

    a = np.absolute(self.local_Fphi)
    b = np.absolute(self.local_Ftheta)
    self.local_F = np.sqrt(a*a + b*b)
    max_magF = np.max(self.local_F)

    if max_magF > 1e-14:
        self.local_F = self.local_F/max_magF
        self.local_Ftheta = self.local_Ftheta/max_magF
        self.local_Fphi = self.local_Fphi/max_magF

    self.local_Frhcp = (self.local_Ftheta -
                        1j*self.local_Fphi)/MyMath.sqrt2
    self.local_Flhcp = (self.local_Ftheta +
                        1j*self.local_Fphi)/MyMath.sqrt2

def copy(self):
    antennas = [antenna.copy() for antenna in self.antennas]
    array = Array(
        name=self.name,
        x_mirror=self.x_mirror,
        y_mirror=self.y_mirror,
        z_mirror=self.z_mirror,
        current_mirror=self.current_mirror,
        azimuth_symmetry=self.azimuth_symmetry,
        antennas=antennas,
        current_magnitude=self.current_magnitude,
        current_phase=self.current_phase,
        phi=self.phi.copy(), theta=self.theta.copy(),
        elevation=self.elevation,
        azimuth=self.azimuth, roll=self.roll,
        x=self.x, y=self.y, z=self.z)

    array.LG_interp_hat_theta = self.LG_interp_hat_theta.copy()
```

```
        array.LG_interp_hat_phi = self.LG_interp_hat_phi.copy()
        array.GL_interp_mesh_theta = self.GL_interp_mesh_theta.copy()
        array.GL_interp_mesh_phi = self.GL_interp_mesh_phi.copy()

        array.local_F = self.local_F.copy()
        array.local_Ftheta = self.local_Ftheta.copy()
        array.local_Fphi = self.local_Fphi.copy()
        array.local_Frhcp = self.local_Frhcp.copy()
        array.local_Flhcp = self.local_Flhcp.copy()
        array.F = self.F.copy()
        array.Ftheta = self.Ftheta.copy()
        array.Fphi = self.Fphi.copy()
        array.Frhcp = self.Frhcp.copy()
        array.Flhcp = self.Flhcp.copy()

        array.local_mesh_flag = self.local_mesh_flag
        array.R_flag = self.R_flag
        array.Rtheta_flag = self.Rtheta_flag
        array.Rphi_flag = self.Rphi_flag
        array.hats_flag = self.hats_flag
        array.local_field_flag = self.local_field_flag
        array.LG_interp_mesh_flag = self.LG_interp_mesh_flag
        array.ok = self.ok

        return array
```

Apêndice 9.69 – ArrayEditorFrame.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 6 23:18:10 2023

@author: 160047412
"""

import tkinter as tk
from tkinter import ttk
import numpy as np

import Antenna
import AntennaEditorFrame
from NumpyExpressionParser import NumpyExpressionParser as NEP


class ArrayEditorFrame(tk.Frame):
    def __init__(self,
                 array,
                 app,
                 on_finish=True,
                 on_done=None,
                 on_cancel=None,
                 master=None,
                 **kw):
        tk.Frame.__init__(self,
                          master=master,
                          width=300,
                          height=200,
                          **kw)
        self.app = app
        self.array = array
        self.on_finish = on_finish
        self.on_done = on_done
        self.on_cancel = on_cancel

        self.antennas = [antenna for antenna in self.array.antennas]
        self.current_editing_antenna = None
        self.current_editing_frame = None

        self.init_variables()
        self.init_layout()

        self.select_antenna_lstbx.bind(
            '<<ListboxSelect>>',
            self.on_antenna_selection)

    def init_variables(self):
        self.name = tk.StringVar(value=self.array.name)
        self.current_magnitude = tk.DoubleVar(
            value=self.array.current_magnitude)
        self.current_phase = tk.DoubleVar(
            value=self.array.current_phase)
        self.phi_initial = tk.DoubleVar(value=np.min(self.array.phi))
        self.phi_final = tk.DoubleVar(value=np.max(self.array.phi))
```

```
        self.phi_N = tk.IntVar(value=self.array.phi.size)
        self.theta_initial = tk.DoubleVar(
            value=np.min(self.array.theta))
        self.theta_final = tk.DoubleVar(
            value=np.max(self.array.theta))
        self.theta_N = tk.IntVar(value=self.array.theta.size)
        self.azimuth = tk.DoubleVar(value=self.array.azimuth)
        self.elevation = tk.DoubleVar(value=self.array.elevation)
        self.position = tk.StringVar(
            value=str([self.array.x, self.array.y, self.array.z]))
        self.x_mirror_variable = tk.IntVar(value=self.array.x_mirror)
        self.y_mirror_variable = tk.IntVar(value=self.array.y_mirror)
        self.z_mirror_variable = tk.IntVar(value=self.array.z_mirror)
        self.current_mirror_variable = tk.IntVar(
            value=self.array.current_mirror)
        self.azimuth_symmetry_variable = tk.IntVar(
            value=self.array.azimuth_symmetry)

        self.edit_antenna_frame_label = tk.StringVar()

    def init_layout(self):
        fr_left = tk.Frame(master=self)
        fr_left.pack(side='left', fill='both')
        fr_left_top = tk.Frame(master=fr_left)
        fr_left_top.pack(side='top', fill='both')
        fr_left_top_left = ttk.LabelFrame(
            master=fr_left_top,
            text='Array␣name:')
        fr_left_top_left.pack(side='left', fill='both')
        ttk.Entry(
            master=fr_left_top_left,
            textvariable=self.name).pack(side='left', fill='both')

        fr = ttk.LabelFrame(
            master=fr_left_top_left,
            text='Current␣magnitude:')
        fr.pack(side='left', fill='both')
        ttk.Entry(
            master=fr,
            textvariable=self.current_magnitude).pack(
                side='left',
                fill='both')

        fr = ttk.LabelFrame(
            master=fr_left_top_left,
            text='Current␣phase:')
        fr.pack(side='left', fill='both')
        ttk.Entry(master=fr, textvariable=self.current_phase).pack(
            side='left',
            fill='both')

        tk.Button(
            master=fr_left_top_left,
            text='update',
            command=self._on_update).pack(side='left', fill='both')

        fr_left_top = ttk.LabelFrame(
```

```
        master=fr_left,
        text='Phi␣angle␣samples:␣linspace(start,␣stop,␣num)')
    fr_left_top.pack(side='top', fill='both')
    ttk.Label(
        master=fr_left_top,
        text='Start').grid(row=0, column=0)
    ttk.Label(
        master=fr_left_top, text='Stop').grid(row=0, column=1)
    ttk.Label(master=fr_left_top, text='Num').grid(
        row=0, column=2)
    ttk.Entry(
        master=fr_left_top,
        textvariable=self.phi_initial).grid(row=1, column=0)
    ttk.Entry(
        master=fr_left_top,
        textvariable=self.phi_final).grid(row=1, column=1)
    ttk.Entry(
        master=fr_left_top,
        textvariable=self.phi_N).grid(row=1, column=2)
    fr_left_top = ttk.LabelFrame(
        master=fr_left,
        text='Theta␣angle␣samples:␣linspace(start,␣stop,␣num)')
    fr_left_top.pack(side='top', fill='both')
    ttk.Label(master=fr_left_top, text='Start').grid(
        row=0,
        column=0)
    ttk.Label(master=fr_left_top, text='Stop').grid(
        row=0,
        column=1)
    ttk.Label(master=fr_left_top, text='Num').grid(
        row=0,
        column=2)
    ttk.Entry(
        master=fr_left_top,
        textvariable=self.theta_initial).grid(row=1, column=0)
    ttk.Entry(
        master=fr_left_top,
        textvariable=self.theta_final).grid(row=1, column=1)
    ttk.Entry(
        master=fr_left_top,
        textvariable=self.theta_N).grid(row=1, column=2)

    fr_left_top = tk.Frame(master=fr_left)
    fr_left_top.pack(side='top', fill='both')
    fr_left_top_left = ttk.LabelFrame(
        master=fr_left_top,
        text='Elevation')
    fr_left_top_left.pack(side='left', fill='both')
    ttk.Entry(
        master=fr_left_top_left,
        textvariable=self.elevation
        ).pack(side='top', fill='both')
    fr_left_top_left = ttk.LabelFrame(
        master=fr_left_top,
        text='Azimuth')
    fr_left_top_left.pack(side='left', fill='both')
    ttk.Entry(
```

```
            master=fr_left_top_left,
            textvariable=self.azimuth).pack(side='top', fill='both')
        fr_left_top_left = ttk.LabelFrame(
            master=fr_left_top,
            text='Position')
        fr_left_top_left.pack(side='left', fill='both')
        ttk.Entry(
            master=fr_left_top_left,
            textvariable=self.position).pack(side='top', fill='both')

        fr_left_top = tk.Frame(master=fr_left)
        fr_left_top.pack(side='top', fill='both')
        mirror_frame = ttk.LabelFrame(
            master=fr_left_top,
            text='Mirror:')
        mirror_frame.pack(side='left', fill='both')
        tk.Checkbutton(
            master=mirror_frame,
            text='YZ␣plane',
            variable=self.x_mirror_variable
        ).grid(row=0, column=0)
        tk.Checkbutton(
            master=mirror_frame,
            text='ZX␣plane',
            variable=self.y_mirror_variable
        ).grid(row=1, column=0)
        tk.Checkbutton(
            master=mirror_frame,
            text='XY␣plane',
            variable=self.z_mirror_variable
        ).grid(row=2, column=0)
        tk.Checkbutton(
            master=mirror_frame,
            text='Current',
            variable=self.current_mirror_variable
        ).grid(row=3, column=0)
        symmetry_frame = ttk.LabelFrame(
            master=fr_left_top,
            text='Symmetry:')
        symmetry_frame.pack(side='left', fill='both')
        tk.Checkbutton(
            master=symmetry_frame,
            text='Azimuth',
            variable=self.azimuth_symmetry_variable
        ).grid(row=0, column=1)

        if self.on_finish:
            fr_left_top = ttk.LabelFrame(
                master=fr_left, text='Finish')
            fr_left_top.pack(side='top', fill='both')
            ttk.Button(
                master=fr_left_top,
                text='Done',
                command=self._on_done
            ).pack(side=tk.LEFT, fill=tk.BOTH)
            ttk.Button(
                master=fr_left_top,
```

```
            text='Cancel',
            command=self.on_cancel
        ).pack(side=tk.LEFT, fill=tk.BOTH)

    fr_left = ttk.LabelFrame(master=self, text='Antennas')
    fr_left.pack(side='left', fill='both')
    tk.Button(
        master=fr_left,
        text='add', command=self.on_add_antenna).pack(
        side='top',
        fill='both')
    tk.Button(
        master=fr_left,
        text='remove', command=self.on_remove_antenna).pack(
        side='top',
        fill='both')
    self.select_antenna_lstbx = tk.Listbox(master=fr_left)
    self.select_antenna_lstbx.pack(side='top', fill='both')
    for antenna in self.antennas:
        self.select_antenna_lstbx.insert(tk.END, antenna.name)

    self.edit_antenna_frame = tk.Frame(master=self)

def on_antenna_selection(self, event=None):
    selection = self.select_antenna_lstbx.curselection()
    if len(selection) == 0:
        return
    if self.current_editing_frame is not None:
        self.current_editing_frame._on_done()
        self.current_editing_frame.destroy()
        self.array.evaluate()
    self.edit_antenna_frame.pack_forget()
    self.current_editing_antenna = (
        [antenna for antenna in self.antennas])[selection[0]]
    if str(self.current_editing_antenna.EditorFrame) == \
            "<class␣'AntennaEditorFrame.AntennaEditorFrame'>":
        self.current_editing_frame = \
            AntennaEditorFrame.AntennaEditorFrame(
                antenna=self.current_editing_antenna,
                on_finish=False, master=self.edit_antenna_frame)
    elif str(self.current_editing_antenna.EditorFrame) == \
            "<class␣'ArrayEditorFrame.ArrayEditorFrame'>":
        self.current_editing_frame = ArrayEditorFrame(
            array=self.current_editing_antenna,
            app=self.app,
            on_finish=False,
            master=self.edit_antenna_frame)
    self.current_editing_frame.pack(side='left', fill='both')
    self.edit_antenna_frame.pack(side='left', fill='both')

def on_add_antenna(self):
    root = tk.Toplevel()
    N_antennas = tk.IntVar(value=1)

    def on_new():
        for i in range(N_antennas.get()):
            antenna = Antenna()
```

```
            self.array.antennas.append(antenna)
            self.antennas.append(antenna)
            self.select_antenna_lstbx.insert(tk.END, antenna.name)
        root.destroy()

    def on_copy_from():
        def on_done():
            sel = lstbx.curselection()
            if len(sel) > 0:
                selected_antenna = self.app.antennas[sel[0]]
                for i in range(N_antennas.get()):
                    antenna = selected_antenna.copy()
                    self.array.antennas.append(antenna)
                    self.antennas.append(antenna)
                    self.select_antenna_lstbx.insert(
                        tk.END, antenna.name)
            root.destroy()
        for c in root.winfo_children():
            c.destroy()
        tk.Entry(master=root, textvariable=N_antennas).pack(
            side='top', fill='both')
        lstbx = tk.Listbox(master=root)
        lstbx.pack(side='left', fill='both')
        tk.Button(master=root, text='done', command=on_done).pack(
            side='left', fill='both')
        for antenna in self.app.antennas:
            lstbx.insert(tk.END, antenna.name)

    def on_cancel():
        root.destroy()
    tk.Entry(
        master=root,
        textvariable=N_antennas
        ).pack(side='top', fill='both')
    tk.Button(
        master=root,
        text='New',
        command=on_new
        ).pack(side='top', fill='both')
    tk.Button(
        master=root,
        text='Copy␣from',
        command=on_copy_from
        ).pack(side='top', fill='both')
    tk.Button(
        master=root,
        text='Cancel',
        command=on_cancel
        ).pack(side='top', fill='both')
    root.mainloop()

def on_remove_antenna(self):
    selection = self.select_antenna_lstbx.curselection()
    if len(selection) == 0:
        return
    if self.current_editing_frame is not None:
        self.current_editing_frame._on_done()
```

```
            self.current_editing_frame.destroy()
        antenna = ([antenna for antenna in self.antennas])[
            selection[0]]
        self.array.antennas.remove(antenna)
        self.select_antenna_lstbx.delete(0, tk.END)
        self.antennas = [antenna for antenna in self.array.antennas]
        for antenna in self.antennas:
            self.select_antenna_lstbx.insert(tk.END, antenna.name)

    def _on_done(self):
        self._on_update()
        if self.on_done is not None:
            self.on_done()

    def _on_update(self):
        if self.current_editing_frame is not None:
            self.current_editing_frame._on_update()

        self.array.set_current(
            magnitude=self.current_magnitude.get(),
            phase=self.current_phase.get())

        # self.array.set_symmetry(
        # x_mirror=self.x_mirror_variable.get() == 1,
        # y_mirror=self.y_mirror_variable.get() == 1,
        # z_mirror=self.z_mirror_variable.get() == 1,
        # current_mirror=self.current_mirror_variable.get() == 1,
        # azimuth_symmetry=self.
        # azimuth_symmetry_variable.get() == 1)

        position = NEP.eval(expression=self.position.get())
        self.array.set_position(
            x=position[0], y=position[1], z=position[2])

        self.array.set_orientation(elevation=self.elevation.get(),
                                   azimuth=self.azimuth.get())

        self.array.resample(
            theta=np.linspace(
                self.theta_initial.get(),
                self.theta_final.get(), self.theta_N.get()),
            phi=np.linspace(
                self.phi_initial.get(),
                self.phi_final.get(), self.phi_N.get()))

        self.array.set_name(name=self.name.get())

        self.array.evaluate()
```

Apêndice 9.70 – CreateMenu.py

```
# -*- coding: utf-8 -*-
""" tk_ToolTip_class101.py
gives a Tkinter widget a tooltip as the mouse is above the widget
tested with Python27 and Python34 by vegaseat 09sep2014
www.daniweb.com/programming/software-development/code/
    484591/a-tooltip-class-for-tkinter

Modified to include a delay time by Victor Zaccardo, 25mar16
Modified to include a dynamic tooltip for ttk.combobox by Vítor Lima,
    11dec2022
Modified to create a popup menu by Vítor Lima, 08mar2023
"""

try:
    # for Python2
    import Tkinter as tk
except ImportError:
    # for Python3
    import tkinter as tk


class CreateMenu(object):
    """
    create a tooltip for a given widget
    """

    def __init__(self, widget, text='widget␣info'):
        self.waittime = 500 # miliseconds
        self.wraplength = 180 # pixels
        self.widget = widget
        self.text = text
        # self.widget.bind("<Enter>", self.enter)
        # self.widget.bind("<Leave>", self.leave)
        self.widget.bind("<ButtonPress>", self.leave)
        self.widget.bind("<Button-3>", self.showtip)
        self.id = None
        self.tw = None

    def enter(self, event=None):
        self.schedule()

    def leave(self, event=None):
        self.unschedule()
        self.hidetip()

    def schedule(self):
        self.unschedule()
        self.id = self.widget.after(self.waittime, self.showtip)

    def unschedule(self):
        id = self.id
        self.id = None
        if id:
            self.widget.after_cancel(id)
```

```
    def showtip(self, event):
        x = event.x
        y = event.y
        x += self.widget.winfo_rootx() + 25
        y += self.widget.winfo_rooty() + 20
        # creates a toplevel window
        if self.tw:
            self.tw.destroy()
        self.tw = tk.Toplevel(self.widget)
        # Leaves only the label and removes the app window
        self.tw.wm_overrideredirect(True)
        self.tw.wm_geometry("+%d+%d" % (x, y))
        if self.widget.create_drop_menu(self.tw, event) is False:
            self.hidetip()

    def hidetip(self):
        tw = self.tw
        self.tw = None
        if tw:
            tw.destroy()
```

Apêndice 9.71 – CreateToolTip.py

```
# -*- coding: utf-8 -*-
""" tk_ToolTip_class101.py
gives a Tkinter widget a tooltip as the mouse is above the widget
tested with Python27 and Python34 by vegaseat 09sep2014
www.daniweb.com/programming/software-development/code/
    484591/a-tooltip-class-for-tkinter

Modified to include a delay time by Victor Zaccardo, 25mar16
Modified to include a dynamic tooltip for ttk.combobox by Vítor Lima,
    11dec2022
"""

try:
    # for Python2
    import Tkinter as tk
except ImportError:
    # for Python3
    import tkinter as tk


class ComboboxToolTip(object):
    """
    create a tooltip for a given combobox
    """

    def __init__(self, combobox, text_dict):
        self.waittime = 500 # miliseconds
        self.wraplength = 180 # pixels
        self.combobox = combobox
        self.text_dict = text_dict
        self.combobox.bind("<Enter>", self.enter)
        self.combobox.bind("<Leave>", self.leave)
        self.combobox.bind("<ButtonPress>", self.leave)
        self.id = None
        self.tw = None

    def enter(self, event=None):
        self.schedule()

    def leave(self, event=None):
        self.unschedule()
        self.hidetip()

    def schedule(self):
        self.unschedule()
        self.id = self.combobox.after(self.waittime, self.showtip)

    def unschedule(self):
        id = self.id
        self.id = None
        if id:
            self.combobox.after_cancel(id)

    def showtip(self, event=None):
        x = y = 0
```

```
        x, y, cx, cy = self.combobox.bbox("insert")
        x += self.combobox.winfo_rootx() + 25
        y += self.combobox.winfo_rooty() + 20
        # creates a toplevel window
        self.tw = tk.Toplevel(self.combobox)
        # Leaves only the label and removes the app window
        self.tw.wm_overrideredirect(True)
        self.tw.wm_geometry("+%d+%d" % (x, y))
        text = self.text_dict[self.combobox['values'][self.combobox.current()]]
        label = tk.Label(self.tw, text=text, justify='left',
                         background="#ffffff", relief='solid', borderwidth=1,
                         wraplength=self.wraplength)
        label.pack(ipadx=1)

    def hidetip(self):
        tw = self.tw
        self.tw = None
        if tw:
            tw.destroy()


class CreateToolTip(object):
    """
    create a tooltip for a given widget
    """

    def __init__(self, widget, text='widget␣info'):
        self.waittime = 500 # miliseconds
        self.wraplength = 180 # pixels
        self.widget = widget
        self.text = text
        self.widget.bind("<Enter>", self.enter)
        self.widget.bind("<Leave>", self.leave)
        self.widget.bind("<ButtonPress>", self.leave)
        self.id = None
        self.tw = None

    def enter(self, event=None):
        self.schedule()

    def leave(self, event=None):
        self.unschedule()
        self.hidetip()

    def schedule(self):
        self.unschedule()
        self.id = self.widget.after(self.waittime, self.showtip)

    def unschedule(self):
        id = self.id
        self.id = None
        if id:
            self.widget.after_cancel(id)

    def showtip(self, event=None):
        x = y = 0
        x, y, cx, cy = self.widget.bbox("insert")
```

```
        x += self.widget.winfo_rootx() + 25
        y += self.widget.winfo_rooty() + 20
        # creates a toplevel window
        self.tw = tk.Toplevel(self.widget)
        # Leaves only the label and removes the app window
        self.tw.wm_overrideredirect(True)
        self.tw.wm_geometry("+%d+%d" % (x, y))
        label = tk.Label(self.tw, text=self.text, justify='left',
                         background="#ffffff", relief='solid', borderwidth=1,
                         wraplength=self.wraplength)
        label.pack(ipadx=1)

    def hidetip(self):
        tw = self.tw
        self.tw = None
        if tw:
            tw.destroy()


# testing ...
if __name__ == '__main__':
    root = tk.Tk()
    btn1 = tk.Button(root, text="button␣1")
    btn1.pack(padx=10, pady=5)
    button1_ttp = CreateToolTip(btn1,
                                'Neque␣porro␣quisquam␣est␣qui␣dolorem␣' +
                                'ipsum␣quia␣dolor␣sit␣amet,␣'
                                'consectetur,␣adipisci␣velit.␣Neque␣porro␣' +
                                'quisquam␣est␣qui␣dolorem␣ipsum␣'
                                'quia␣dolor␣sit␣amet,␣consectetur,␣adipisci␣' +
                                'velit.␣Neque␣porro␣quisquam␣'
                                'est␣qui␣dolorem␣ipsum␣quia␣dolor␣sit␣amet,␣' +
                                'consectetur,␣adipisci␣velit.')

    btn2 = tk.Button(root, text="button␣2")
    btn2.pack(padx=10, pady=5)
    button2_ttp = CreateToolTip(btn2,
                                "First␣thing's␣first,␣I'm␣the␣realest.␣" +
                                "Drop␣this␣and␣let␣the␣whole␣world␣"
                                "feel␣it.␣And␣I'm␣still␣in␣the␣Murda␣" +
                                "Bizness.␣I␣could␣hold␣you␣down,␣like␣"
                                "I'm␣givin'␣lessons␣in␣␣physics.␣You␣" +
                                "should␣want␣a␣bad␣Vic␣like␣this.")
    root.mainloop()
```

Apêndice 9.72 – CustomOptimization.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 4 14:20:23 2023

@author: vitinho
"""

import time
import numpy as np
import scipy.optimize

# OPTIONS
save_graphs = True

class CustomOptimization:
    methods = [
        'Nelder-Mead',
        'Powell',
        'CG',
        'BFGS',
        'Newton-CG',
        'L-BFGS-B',
        'TNC',
        'COBYLA',
        'SLSQP',
        'trust-ngc',
        'trust-exact',
        'trust-krylov'
        ]
    def __init__(self,
                 cost_function,
                 x_map,
                 x0=None,
                 name='new optimization',
                 method='L-BFGS-B',
                 working_array=None,
                 target_antenna=None,
                 bounds=None,
                 options={
                     # 'disp':True,
                     'eps':0.01,
                     'gtol':0.1,
                     'xrtol':0.1,
                     'maxiter':30
                     },
                 disp=False):
        self.cost_function=cost_function
        self.x_map=x_map
        self.x0=x0
        self.name=name
        self.method=method
        self.working_array=working_array
        self.target_antenna=target_antenna
        self.bounds=bounds
```

```
        self.options=options
        self.disp=disp

        self.result = None
        self.cost = None
        self.x = []

        if self.x0 is None:
            self.get_x0_from_array()

    def get_x0_from_array(self):
        self.x0 = []
        for e in x_map:
            antenna = e['antenna']
            for variable in e['variables']:
                if variable=='x':
                    x0_element = antenna.x
                elif variable=='y':
                    x0_element = antenna.y
                elif variable=='z':
                    x0_element = antenna.z
                elif variable=='elevation':
                    x0_element = antenna.elevation/180 + 0.5
                elif variable=='azimuth':
                    x0_element = antenna.azimuth/360 + 0.5
                elif variable=='roll':
                    x0_element = antenna.roll/360 + 0.5
                elif variable=='current␣magnitude':
                    x0_element = antenna.current_magnitude
                elif variable=='current␣phase':
                    x0_element = antennas.current_phase/360 + 0.5
                self.x0.append(x0_element)
        N_2 = int(len(self.working_array.antennas)/2)
        if len(self.working_array.antennas)&1:
            n = 1
        else:
            n = 0
        for i in range(N_2):
            antenna_2 = self.working_array.antennas[n + i]
            antenna_1 = self.working_array.antennas[n + N_2 + i]
            antenna_2.set_position(x=antenna_1.x,
                                   y=-antenna_1.y,
                                   z=antenna_1.z)
            antenna_2.set_orientation(elevation=antenna_1.elevation,
                                      azimuth=(antenna_1.azimuth+360)%360-180,
                                      roll=antenna_1.roll)
            antenna_2.set_current(magnitude=antenna_1.current_magnitude,
                                  phase=180 + antenna_1.current_phase)

    def cost_function_helper(self, x, *args):
        self.number_of_evaluations += 1

        N_2 = int(len(self.working_array.antennas)/2)
        if len(self.working_array.antennas) & 1:
            k = 0
            i = 0
            for variable in self.x_map[0]['variables']:
```

```
            current_assertion = self.get_assertion(variable)
            current_assertion(self.working_array.antennas[0], x[k])
            k = k+1
        i = i+1
        n = 1
    else:
        n = 0
    k = 0
    i = 0
    for e in self.x_map:
        for variable in e['variables']:
            current_assertion = self.get_assertion(variable)
            current_assertion(self.working_array.antennas[n+i], x[k], 1)
            current_assertion(self.working_array.antennas[n+N_2+i], x[k], -1)
            k = k+1
        i = i+1

    for antenna in self.working_array.antennas:
        antenna.ok = False

    self.working_array.local_field_flag = True
    self.working_array.ok = False
    self.working_array.evaluate()

    cost = self.cost_function(
        self.target_antenna,
        self.working_array)
    if self.disp:
        print('evaluating␣with␣' + str(x) + '␣cost␣' + str(cost))
        for i in range(len(self.working_array.antennas)):
            antenna = self.working_array.antennas[i]
            print('\tantenna␣{i}:␣'.format(i=i) + antenna.name)
            print('\t\televation:␣{e}'.format(
                e=antenna.elevation))
            print('\t\tazimuth:␣{a}'.format(a=antenna.azimuth))
            print('\t\troll:␣{a}'.format(a=antenna.roll))
            print('\t\tx:␣{x}'.format(x=antenna.x))
            print('\t\ty:␣{y}'.format(y=antenna.y))
            print('\t\tz:␣{z}'.format(z=antenna.z))
            print('\t\tcurrent␣magnitude:␣{magnitude}'.format(
                magnitude=antenna.current_magnitude))
            print('\t\tcurrent␣phase:␣{phase}'.format(
                phase=antenna.current_phase))

    if self.cost is None:
        self.cost = cost
        self.x = x
    elif cost < self.cost:
        self.cost = cost
        self.x = x
    return cost

def get_assertion(self, variable):
    if variable=='x':
        current_assertion = self.set_x
    elif variable=='y':
        current_assertion = self.set_y
```

```
        elif variable=='z':
            current_assertion = self.set_z
        elif variable=='elevation':
            current_assertion = self.set_elevation
        elif variable=='azimuth':
            current_assertion = self.set_azimuth
        elif variable=='roll':
            current_assertion = self.set_roll
        elif variable=='current␣magnitude':
            current_assertion = self.set_current_magnitude
        elif variable=='current␣phase':
            current_assertion = self.set_current_phase
        return current_assertion

    def run(self):
        if not self.target_antenna.ok:
            self.target_antenna.evaluate()
        self.start_time = time.time()

        self.working_x_map = []
        self.number_of_evaluations = 0
        self.result = scipy.optimize.minimize(
            fun=self.cost_function_helper,x0=self.x0,
            method=self.method,
            bounds=self.bounds,
            options=self.options)
        self.cost_function_helper(self.x)

        self.elapsed_time = time.time()-self.start_time

        if self.disp:
            print('elapsed␣time␣was␣' + str(self.elapsed_time))
            print('number␣of␣evaluations␣was␣' +
                  str(self.number_of_evaluations))
            print('final␣cost:␣{}'.format(self.cost))
            print('final␣array␣have␣{N}␣antennas:'.format(
                N=len(self.working_array.antennas)))
            for i in range(len(self.working_array.antennas)):
                antenna = self.working_array.antennas[i]
                print('\tantenna␣{i}:␣'.format(i=i) + antenna.name)
                print('\t\televation:␣{e}'.format(
                    e=antenna.elevation))
                print('\t\tazimuth:␣{a}'.format(a=antenna.azimuth))
                print('\t\troll:␣{a}'.format(a=antenna.roll))
                print('\t\tx:␣{x}'.format(x=antenna.x))
                print('\t\ty:␣{y}'.format(y=antenna.y))
                print('\t\tz:␣{z}'.format(z=antenna.z))
                print('\t\tcurrent␣magnitude:␣{magnitude}'.format(
                    magnitude=antenna.current_magnitude))
                print('\t\tcurrent␣phase:␣{phase}'.format(
                    phase=antenna.current_phase))

        return self.result

    def set_elevation(self, antenna,x, s=1):
        antenna.set_orientation(elevation=(x-0.5)*180)
```

```
    def set_azimuth(self, antenna,x, s=1):
        if s==1:
            antenna.set_orientation(azimuth=(x-0.5)*360)
        else:
            antenna.set_orientation(azimuth=180 + (x-0.5)*360)

    def set_roll(self, antenna,x, s=1):
        antenna.set_orientation(roll=(x-0.5)*360)

    def set_x(self, antenna,x, s=1):
        antenna.set_position(x=s*x)

    def set_y(self, antenna,y, s=1):
        antenna.set_position(y=s*y)

    def set_z(self, antenna,z, s=1):
        antenna.set_position(z=s*z)

    def set_current_magnitude(self, antenna,magnitude, s=1):
        antenna.set_current(magnitude=magnitude)

    def set_current_phase(self, antenna,phase, s=1):
        antenna.set_current(phase=(phase-0.5)*360)

def cost_function(target, array):
    cost = (np.abs(np.abs(target.Frhcp)-np.abs(array.Frhcp))*target.sin_mesh_theta).sum()
    cost += (np.abs(np.abs(target.Flhcp)-np.abs(array.Flhcp))*target.sin_mesh_theta).sum()

    return cost

import sys
import os
import header

import matplotlib.pyplot as plt
plt.close('all')

import pickle

print("Loading␣antennas")

import LoadHFSSYagis
antennas = LoadHFSSYagis.run(Ntheta=91,
                                                Nphi=91)

print("Loading␣export␣results␣directory")

N = 8
export_directory = os.path.join(
    header.results_dir,
    'Optimization',
    'CustomOptimization')
if not os.path.exists(export_directory):
    os.mkdir(export_directory)

print("Creating␣target␣distribution")
```

```
import Antenna
import Array

Ntheta=91
Nphi=181

theta=np.linspace(0, 180, Ntheta)
phi=np.linspace(-180, 180, Nphi)

target_antenna = Antenna.Antenna(
    name='Target␣Antenna',
    theta=theta.copy(),
    phi=phi.copy())
target_antenna.evaluate_as = 'expressions'
U1 = '(U(-pi/2,phi)-U(pi/2,phi))'
U2 = '(U(radians(60),theta)-U(radians(80),theta))'
U3 = '(U(radians(100),theta)-U(radians(120),theta))'
U4 = '('+U2+'+'+U3+')'
U5 = '(U(radians(80),theta)-U(radians(100),theta))'
target_antenna.evaluation_arguments['expression␣theta'] = \
    '0.1␣-␣0.9*1j*('+U1+'*'+U5+')'
target_antenna.evaluation_arguments['expression␣phi'] = \
    '0.1␣+␣0.9*('+U1+'*'+U5+')'
target_antenna.set_orientation(elevation=-90,azimuth=90)
target_antenna.evaluate()

print("Creating␣working␣array")

antennas = [antennas['hfss_yagi2EL'].copy() for a in range(N)]
initial_array = Array.Array(
    name='Initial␣array',
    theta=theta.copy(),
    phi=phi.copy(),
    antennas=antennas
    )
# k = -0.1
k = -1
N_2 = int(N/2)
if N&1:
    initial_array.antennas[0].set_position(
        # x=2*(np.random.rand()-0.5)*len(initial_array.antennas),
        # y=2*(np.random.rand()-0.5)*len(initial_array.antennas),
        x = 0,
        y = 0,
        z=0)
    initial_array.antennas[0].set_orientation(
        elevation=-90,
        azimuth=0,
        roll=0)
    initial_array.antennas[0].set_current(
        magnitude = 1,#/(2*i+1),
        phase = 0)
    n = 1
else:
    n = 0
for i in range(int(N/2)):
    # x=2*(np.random.rand()-0.5)*len(initial_array.antennas),
```

```
    # y=2*(np.random.rand()-0.5)*len(initial_array.antennas),
    x = 0
    y = i*0.5 + 0.25*(1-n)
    z = 0
    elevation = -90
    azimuth = 0
    roll = 90*(k/2+0.5)
    magnitude = 1#/(2*i+1)
    phase = 90*(k/2+0.5)
    initial_array.antennas[n+i].set_position(
        x = x,
        y = y,
        z = z)
    initial_array.antennas[n+i].set_orientation(
        elevation=elevation,
        azimuth=azimuth,
        roll=roll)
    initial_array.antennas[n+i].set_current(
        magnitude = magnitude,
        phase = phase)
    initial_array.antennas[n+N_2+i].set_position(
        x = x,
        y = -y,
        z = z)
    initial_array.antennas[n+N_2+i].set_orientation(
        elevation=elevation,
        azimuth=(azimuth+360)%360 - 180,
        roll=roll)
    initial_array.antennas[n+N_2+i].set_current(
        magnitude = magnitude,
        phase = (phase+360)%360 - 180)
    k = k*-1
initial_array.evaluate()

if save_graphs:
    print("Exporting␣initial␣state")

    import ExportResults

    ExportResults.run([
        initial_array,
        ], export_directory,
        fields=[
            'F',
            'Fref',
            'Fcross',
            'Frhcp',
            'Flhcp'
            ])

print("Optimizing")

working_array = initial_array.copy()

variables = [
    # 'x',
    # 'y',
```

```
    'elevation',
    # 'azimuth',
    # 'roll',
    'current␣magnitude',
    # 'current phase',
    ]
x_map = [
    dict(
        antenna = working_array.antennas[i],
        variables = variables,
        ) for i in range(int(N/2)+(N&1))
    ]

bounds_element = []
if 'x' in variables:
    bounds_element.append((0,10))
if 'y' in variables:
    bounds_element.append((0,10))
if 'z' in variables:
    bounds_element.append((0,10))
if 'elevation' in variables:
    bounds_element.append((0,1))
if 'azimuth' in variables:
    bounds_element.append((0,1))
if 'roll' in variables:
    bounds_element.append((0,1))
if 'current␣magnitude' in variables:
    bounds_element.append((0,None))
if 'current␣phase' in variables:
    bounds_element.append((0,1))

bounds = []
for i in range(int(N/2)+(N&1)):
    bounds += bounds_element

optim = CustomOptimization(
    cost_function=cost_function,
    x_map = x_map,
    target_antenna=target_antenna,
    working_array=working_array,
    disp=True,
    bounds=bounds,
    # method='Nelder-Mead',
    # method='Powell',
    # tol = 1,
    # options=dict(maxiter=300),
    )

result = optim.run()
final_array = optim.working_array

## Export result array with python module "pickle" for easy loading
# filename = os.path.join(
# export_directory,
# 'Optimization with {N} antennas and cost {cost}.dat'.format(
# N=len(optim.working_array.antennas),
# cost=optim.result.fun))
```

```
# with open(filename, mode='wb') as f:
# pickle.dump(optim, f)

print(result)

final_array.name = 'Result Array'

if save_graphs:
    print("Exporting results")

    ExportResults.run([
        initial_array,
        target_antenna,
        final_array,
        ], export_directory,
        fields=[
            'F',
            # 'Fref',
            # 'Fcross',
            'Frhcp',
            'Flhcp'
            ])

    import ExportTable

    initial_array.name = 'Arranjo inicial'
    final_array.name = 'Arranjo final otimizado'

    ExportTable.export_table(
        export_directory,
        arrays=[
            initial_array,
            final_array
            ],
        captions = [
            "Arranjo inicial",
            "Resultados da otimização de polarização linear " +
                "com custo final de {:.2f}".format(optim.cost)
            ])

initial_cost = cost_function(initial_array, target_antenna)
final_cost = cost_function(final_array, target_antenna)
print("Custo inicial: " + str(round(initial_cost*100)/100))
print("Custo final: " + str(round(final_cost*100)/100))
print("Melhora de " + str(round((initial_cost-final_cost)/initial_cost*10000)/100) + "%")

if __name__=="__main__":

    print("Starting visualization")

    import App
    import ResultFrame
    import Result

    app = App.App(antennas=[target_antenna, final_array])
    try:
        Ntheta = 91
```

```
Nphi = 181
field = 'F'
plot = '2d␣Polar␣Patch'

tab = ResultFrame.ResultFrame(master=app.tabs,name=field,
                              columns=2,rows=1)
Result.Result(tab=tab,
              title='Result',
              name='Result',
              antenna=final_array,
              field=field,
              plot=plot,
              ticks_flag=False,
              in_dB=True,
              column=1,row=1,
              Ntheta=Ntheta,
              Nphi=Nphi)
Result.Result(tab=tab,
              title='Target',
              name='Target',
              antenna=target_antenna,
              field=field,
              plot=plot,
              ticks_flag=False,
              in_dB=True,
              column=2,row=1,
              Ntheta=Ntheta,
              Nphi=Nphi)
app.add_tab(tab)

field = 'Frhcp'
plot = '2d␣Polar␣Patch'

tab = ResultFrame.ResultFrame(master=app.tabs,name=field,
                              columns=2,rows=1)
Result.Result(tab=tab,
              title='Result',
              name='Result',
              antenna=final_array,
              field=field,
              plot=plot,
              ticks_flag=False,
              in_dB=True,
              column=1,row=1,
              Ntheta=Ntheta,
              Nphi=Nphi)
Result.Result(tab=tab,
              title='Target',
              name='Target',
              antenna=target_antenna,
              field=field,
              plot=plot,
              ticks_flag=False,
              in_dB=True,
              column=2,row=1,
              Ntheta=Ntheta,
              Nphi=Nphi)
```

```
        app.add_tab(tab)

        field = 'Flhcp'
        plot = '2d␣Polar␣Patch'

        tab = ResultFrame.ResultFrame(master=app.tabs,name=field,
                                      columns=2,rows=1)
        Result.Result(tab=tab,
                      title='Result',
                      name='Result',
                      antenna=final_array,
                      field=field,
                      plot=plot,
                      ticks_flag=False,
                      in_dB=True,
                      column=1,row=1,
                      Ntheta=Ntheta,
                      Nphi=Nphi)
        Result.Result(tab=tab,
                      title='Target',
                      name='Target',
                      antenna=target_antenna,
                      field=field,
                      plot=plot,
                      ticks_flag=False,
                      in_dB=True,
                      column=2,row=1,
                      Ntheta=Ntheta,
                      Nphi=Nphi)
        app.add_tab(tab)

        # Main application loop
        app.mainloop()
    except Exception as e:
        # If some error occur, destroy the application to close
        # the window, then show the error
        app.destroy()
        raise e
```

Apêndice 9.73 – ExportCompare.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 20 23:34:00 2023

@author: 160047412
"""

import sys
import os
path = os.path.split(__file__)[0]
path = os.path.split(path)[0]
sys.path.insert(0, path)
path = os.path.split(path)[0]
home_directory = os.path.split(path)[0]
antennas_dir=os.path.join(home_directory, 'Antennas')
import numpy as np

import matplotlib.pyplot as plt

import ResultFigure
import Result

def run(antennas, export_directory):
    fields = [
        'F',
        # 'Fref',
        # 'Fcross',
        # 'Ftheta',
        # 'Fphi'
        ]

    if not os.path.exists(export_directory):
        os.mkdir(export_directory)
    for antenna_pair in antennas:
        for field in fields:
            figure = ResultFigure.ResultFigure()
            plot = '2d Polar Patch'
            result = Result.Result(tab=figure,
                                   title='',
                                   field=field,
                                   plot='custom',
                                   ticks_flag=False,
                                   in_dB=False)

            def draw_comparison(result, tab):
                result.axes = tab.request_axes(
                    requester=result,
                    projection='polar',
                    position=1)
                result.axes.set_title(result.title)
                result.radius_ticks = [22.5, 45, 66.5, 90]

                interp_theta_deg = np.linspace(0,90,81)
                interp_phi_deg = np.linspace(-180,180,91)
```

```
    interp_mesh_phi_deg,interp_mesh_theta_deg = \
        np.meshgrid(interp_phi_deg,interp_theta_deg)

    field_0,color_0 = result.get_field(antenna_pair[0])
    field_1,color_1 = result.get_field(antenna_pair[1])

    color_0 = antenna_pair[0].interpolate_at(
        interp_mesh_theta_deg,
        interp_mesh_phi_deg, color_0)
    color_1 = antenna_pair[1].interpolate_at(
        interp_mesh_theta_deg,
        interp_mesh_phi_deg, color_1)

    color = 20*np.log10(np.abs(color_0) - np.abs(color_1))

    color[color<result.dynamic_scaling_dB] = \
        result.dynamic_scaling_dB
    color[np.isnan(color)] = result.dynamic_scaling_dB


    if result.color=='Color␣by␣magnitude':
        cmap = 'jet'
    elif result.color=='Color␣by␣phase':
        cmap = 'twilight'

    color_max = color.max()
    color_min = color.min()
    if color_max!=color_min:
        C = (color-color_min)/(color_max-color_min)
        rgb = plt.colormaps[cmap](C)
    else:
        rgb = list(color.shape)
        rgb.append(4)
        rgb = np.zeros(tuple(rgb))
        rgb[:,:,3] = 1
        rgb[:,:,2] = 1

    result.graphical_objects = result.axes.pcolormesh(
        np.radians(interp_mesh_phi_deg),
        90*np.sin(np.radians(interp_mesh_theta_deg)),
        color,
shading='gouraud',
cmap=cmap,
        vmin=result.colorbar_min,
        vmax=result.colorbar_max)
    plt.colorbar(result.graphical_objects,ax=result.axes,
                 extend='both',
                 pad=0.1)
    result.axes.grid(False)
    result.axis = []
    r = np.array([0, 90])
    degree_sign = u"\N{DEGREE␣SIGN}"
    for angle in [0,
          np.pi/4,
          np.pi/2,
          3*np.pi/4,
          np.pi,
```

```
                    -3*np.pi/4,
                    -np.pi/2,
                    -np.pi/4]:
                result.axis.append(result.axes.plot(np.array([angle, angle]), r, 'w',
                    ↪ linewidth=0.3))
            a = np.linspace(0, 2*np.pi, 360)
            for radius in result.radius_ticks:
                result.axis.append(result.axes.plot(a, radius*np.ones_like(a), 'w', linewidth
                    ↪ =0.3))
            result.axes.set_rticks(result.radius_ticks)
            result.axes.set_yticklabels([str(i) + degree_sign for i in result.radius_ticks])

        result.custom_draw = draw_comparison
        figure.draw()
        fname = os.path.join(
            export_directory,
            antenna_pair[0].name +
                '␣and␣' + \
                antenna_pair[1].name +
                '␣' + field +
                '␣' + plot +
                '␣compare.png')
        figure.figure.savefig(fname)
        plt.close('all')

if __name__=='__main__':
    import Scripts.AntennasLoaders.LoadHFSSYagis
    antennas = Scripts.AntennasLoaders.LoadHFSSYagis.run(
        Ntheta=91, Nphi=91)

    import Scripts.AntennasLoaders.LoadHFSSValidationArrays
    antennas.update(Scripts.AntennasLoaders.LoadHFSSValidationArrays.run(
        Ntheta=91, Nphi=91))

    import Scripts.AntennasLoaders.LoadValidationArrays
    antennas.update(Scripts.AntennasLoaders.LoadValidationArrays.run(
        Ntheta=91, Nphi=91))

    export_directory = os.path.join(home_directory,
                                    'Python',
                                    'ExportedResults',
                                    'Comparisons')
    if not os.path.exists(export_directory):
        os.mkdir(export_directory)
    Scripts.ExportCompare.run([
                               (antennas['array_validation_1Y4EL'],
                               antennas['HFSS_1Y4EL']),
                               (antennas['array_validation_2Y4EL'],
                               antennas['HFSS_2Y4EL']),
                               (antennas['array_validation_3Y4EL'],
                               antennas['HFSS_3Y4EL']),
                               (antennas['array_validation_4Y4EL'],
                               antennas['HFSS_4Y4EL']),
                               (antennas['array_validation_5Y4EL'],
                               antennas['HFSS_5Y4EL'])
                               ], export_directory)
```

Apêndice 9.74 – exportOptimizationCircular.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Aug 2 21:15:26 2023

@author: vitinho
"""

import sys
import os
import header

import matplotlib.pyplot as plt
plt.close('all')

print("Loading antennas")

import LoadHFSSYagis
antennas = LoadHFSSYagis.run(Ntheta=91,
                                                Nphi=91)

print("Loading export results directory")

export_directory = os.path.join(
    header.results_dir,
    'Optimization',
    'CircularOptimization')
if not os.path.exists(export_directory):
    os.mkdir(export_directory)

print("Creating target distribution")

import numpy as np

import Array

Ntheta=91
Nphi=181

theta=np.linspace(0, 180, Ntheta)
phi=np.linspace(-180, 180, Nphi)

target_antenna = Array.Array(name='Target Antenna',
                          theta=theta.copy(),
                          phi=phi.copy(),
                          antennas=[
                              antennas['hfss_yagi2EL'].copy(),
                              antennas['hfss_yagi2EL'].copy(),
                              antennas['hfss_yagi2EL'].copy(),
                         ])
target_antenna.antennas[0].set_position(x=0,y=0,z=0)
target_antenna.antennas[0].set_orientation(elevation=-90,azimuth=0)
target_antenna.antennas[1].set_position(x=0,y=0.5,z=0)
target_antenna.antennas[1].set_orientation(elevation=-90,azimuth=90)
target_antenna.antennas[2].set_position(x=0,y=1,z=0)
```

```
target_antenna.antennas[2].set_orientation(elevation=-90,azimuth=90)
target_antenna.evaluate()

print("Creating␣working␣array")

initial_array = Array.Array(name='Initial␣Array',
                        theta=theta.copy(),
                        phi=phi.copy(),
                        antennas=[
                            antennas['hfss_yagi2EL'].copy(),
                            antennas['hfss_yagi2EL'].copy(),
                            antennas['hfss_yagi2EL'].copy(),
                       ])
initial_array.antennas[0].set_position(x=0,y=0,z=0)
initial_array.antennas[0].set_orientation(elevation=-90,azimuth=0)
initial_array.antennas[1].set_position(x=0.3,y=0.3,z=0)
initial_array.antennas[1].set_orientation(elevation=-90,azimuth=120)
initial_array.antennas[2].set_position(x=-0.3,y=1.3,z=0)
initial_array.antennas[2].set_orientation(elevation=-90,azimuth=60)
initial_array.evaluate()

print("Exporting␣initial␣state")

import ExportResults

ExportResults.run([
    initial_array,
    ], export_directory,
    fields=[
        'F',
        'Fref',
        'Fcross',
        'Frhcp',
        'Flhcp'
        ])

print("Optimizing")

def cost_function(target, result):
    total_cost = (np.abs(np.abs(target.Frhcp) -
                      np.abs(result.Frhcp))*target.sin_mesh_theta).sum()
    total_cost += (np.abs(np.abs(target.Flhcp) -
                       np.abs(result.Flhcp))*target.sin_mesh_theta).sum()
    return total_cost

import Optimization

working_array = initial_array.copy()
optim = Optimization.Optimization(
    target_antenna=target_antenna,
    working_array=working_array,
    x_map = [
        dict(
            antenna=working_array.antennas[1],
            variables = [
                'x',
                'y',
```

```
                'roll',
                ],
            ),
        dict(
            antenna=working_array.antennas[2],
            variables = [
                'x',
                'y',
                'roll',
                ],
            ),
        ],
    cost_function=cost_function,
    # method = 'L-BFGS-B',
    )

result = optim.run()
final_array = optim.working_array

print(result)

final_array.name = 'Result Array'

print("Exporting results")

ExportResults.run([
    target_antenna,
    final_array,
    ], export_directory,
    fields=[
        'F',
        'Fref',
        'Fcross',
        'Frhcp',
        'Flhcp'
        ])

import ExportTable

initial_array.name = 'Arranjo inicial - otimização circular'
target_antenna.name = 'Arranjo alvo de polarização circular ' + \
    '- otimização circular'
final_array.name = 'Arranjo final otimizado - otimização circular'

ExportTable.export_table(
    export_directory,
    arrays=[
        initial_array,
        target_antenna,
        final_array
        ],
    captions = [
        "Arranjo inicial",
        "Arranjo alvo",
        "Resultado da otimização de polarização circular " +
            "com custo final de {:.2f}".format(optim.cost)
        ])
```

```
initial_cost = cost_function(initial_array, target_antenna)
final_cost = cost_function(final_array, target_antenna)
print("Custo␣inicial:␣" + str(round(initial_cost*100)/100))
print("Custo␣final:␣" + str(round(final_cost*100)/100))
print("Melhora␣de␣" + str(round((initial_cost-final_cost)/initial_cost*10000)/100) + "%")

if __name__=='__main__':

    print("Starting␣visualization")

    import App
    import ResultFrame
    import Result

    app = App.App(antennas=[target_antenna, final_array])
    try:
        Ntheta = 91
        Nphi = 181
        field = 'F'
        plot = '2d␣Polar␣Patch'

        tab = ResultFrame.ResultFrame(
            master=app.tabs,name=field,columns=2,rows=1)
        Result.Result(tab=tab,
                      title='Result',
                      name='Result␣F',
                      antenna=final_array,
                      field=field,
                      plot=plot,
                      ticks_flag=False,
                      in_dB=True,
                      column=1,row=1,
                      Ntheta=Ntheta,
                      Nphi=Nphi)
        Result.Result(tab=tab,
                      title='Target',
                      name='Target␣F',
                      antenna=target_antenna,
                      field=field,
                      plot=plot,
                      ticks_flag=False,
                      in_dB=True,
                      column=2,row=1,
                      Ntheta=Ntheta,
                      Nphi=Nphi)
        app.add_tab(tab)

        field = 'Frhcp'
        plot = '2d␣Polar␣Patch'

        tab = ResultFrame.ResultFrame(
            master=app.tabs,name=field,columns=2,rows=1)
        Result.Result(tab=tab,
                      title='Result',
                      name='Result␣RHCP',
                      antenna=final_array,
```

```
                    field=field,
                    plot=plot,
                    ticks_flag=False,
                    in_dB=True,
                    column=1,row=1,
                    Ntheta=Ntheta,
                    Nphi=Nphi)
        Result.Result(tab=tab,
                    title='Target',
                    name='Target␣RHCP',
                    antenna=target_antenna,
                    field=field,
                    plot=plot,
                    ticks_flag=False,
                    in_dB=True,
                    column=2,row=1,
                    Ntheta=Ntheta,
                    Nphi=Nphi)
        app.add_tab(tab)

        field = 'Flhcp'
        plot = '2d␣Polar␣Patch'

        tab = ResultFrame.ResultFrame(
            master=app.tabs,name=field,columns=2,rows=1)
        Result.Result(tab=tab,
                    title='Result',
                    name='Result␣LHCP',
                    antenna=final_array,
                    field=field,
                    plot=plot,
                    ticks_flag=False,
                    in_dB=True,
                    column=1,row=1,
                    Ntheta=Ntheta,
                    Nphi=Nphi)
        Result.Result(tab=tab,
                    title='Target',
                    name='Target␣LHCP',
                    antenna=target_antenna,
                    field=field,
                    plot=plot,
                    ticks_flag=False,
                    in_dB=True,
                    column=2,row=1,
                    Ntheta=Ntheta,
                    Nphi=Nphi)
        app.add_tab(tab)

        # Main application loop
        app.mainloop()
    except Exception as e:
        # If some error occur, destroy the application to close
        # the window, then show the error
        app.destroy()
        raise e
```

Apêndice 9.75 – exportOptimizationRef.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Aug 2 21:06:42 2023

@author: vitinho
"""

import sys
import os
import header

import Result
import ResultFrame
import App
import ExportResults
import Array
import numpy as np
import LoadHFSSYagis
import matplotlib.pyplot as plt

plt.close('all')

print("Loading␣antennas")

antennas = LoadHFSSYagis.run(
    Ntheta=91, Nphi=91)

print("Loading␣export␣results␣directory")

export_directory = os.path.join(
    header.results_dir,
    'Optimization',
    'LinearOptimization')
if not os.path.exists(export_directory):
    os.mkdir(export_directory)

print("Creating␣target␣distribution")


Ntheta = 91
Nphi = 181

theta = np.linspace(0, 180, Ntheta)
phi = np.linspace(-180, 180, Nphi)

target_antenna = Array.Array(
    name='Target␣Antenna',
    theta=theta.copy(),
    phi=phi.copy(),
    antennas=[
        antennas['hfss_yagi2EL'].copy(
        ),
        antennas['hfss_yagi2EL'].copy(
        ),
```

```
        antennas['hfss_yagi2EL'].copy(
        ),
    ])
target_antenna.antennas[0].set_position(
    x=0, y=0, z=0)
target_antenna.antennas[0].set_orientation(
    elevation=-90, azimuth=55)
target_antenna.antennas[1].set_position(
    x=0, y=0.5, z=0)
target_antenna.antennas[1].set_orientation(
    elevation=-90, azimuth=55)
target_antenna.antennas[2].set_position(
    x=0, y=1, z=0)
target_antenna.antennas[2].set_orientation(
    elevation=-90, azimuth=55)
target_antenna.evaluate()

print("Creating␣working␣array")

initial_array = Array.Array(
    name='Initial␣Array',
    theta=theta.copy(),
    phi=phi.copy(),
    antennas=[
        antennas['hfss_yagi2EL'].copy(
        ),
        antennas['hfss_yagi2EL'].copy(
        ),
        antennas['hfss_yagi2EL'].copy(
        ),
    ])
initial_array.antennas[0].set_position(
    x=0, y=0, z=0)
initial_array.antennas[0].set_orientation(
    elevation=-90, azimuth=60)
initial_array.antennas[1].set_position(
    x=0.0, y=0.5, z=0)
initial_array.antennas[1].set_orientation(
    elevation=-90, azimuth=30)
initial_array.antennas[2].set_position(
    x=0.0, y=1.0, z=0)
initial_array.antennas[2].set_orientation(
    elevation=-90, azimuth=60)
initial_array.evaluate()

print("Exporting␣initial␣state")


ExportResults.run([
    initial_array,
], export_directory)

print("Optimizing")


def cost_function(target, result):
    total_cost = (np.abs(
```

```
        np.abs(target.Fcross)-np.abs(result.Fcross))*target.sin_mesh_theta).sum()
    total_cost += (np.abs(np.abs(target.Fref) -
                        np.abs(result.Fref))*target.sin_mesh_theta).sum()
    return total_cost

import Optimization

working_array = initial_array.copy()
optim = Optimization.Optimization(
    target_antenna=target_antenna,
    working_array=working_array,
    x_map = [
        dict(
            antenna=working_array.antennas[0],
            variables = [
                # 'x',
                # 'y',
                'elevation',
                'azimuth',
                # 'roll',
                ],
            ),
        dict(
            antenna=working_array.antennas[1],
            variables = [
                # 'x',
                # 'y',
                'elevation',
                'azimuth',
                # 'roll',
                ],
            ),
        dict(
            antenna=working_array.antennas[2],
            variables = [
                # 'x',
                # 'y',
                'elevation',
                'azimuth',
                # 'roll',
                ],
            ),
        ],
    cost_function=cost_function,
    # method = 'L-BFGS-B',
    )

result = optim.run()
final_array = optim.working_array

print(result)

final_array.name = 'Result␣Array'

print("Exporting␣results")

ExportResults.run([
```

```
    target_antenna,
    final_array,
], export_directory)

import ExportTable

initial_array.name = 'Arranjo␣inicial␣-␣otimização␣linear'
target_antenna.name = 'Arranjo␣alvo␣de␣polarização␣linear␣' + \
    '-␣otimização␣linear'
final_array.name = 'Arranjo␣final␣otimizado␣-␣otimização␣linear'

ExportTable.export_table(
    export_directory,
    arrays=[
        initial_array,
        target_antenna,
        final_array
        ],
    captions = [
        "Arranjo␣inicial",
        "Arranjo␣alvo",
        "Resultados␣da␣otimização␣de␣polarização␣linear␣" +
            "com␣custo␣final␣de␣{:.2f}".format(optim.cost)
        ])

initial_cost = cost_function(initial_array, target_antenna)
final_cost = cost_function(final_array, target_antenna)
print("Custo␣inicial:␣" + str(round(initial_cost*100)/100))
print("Custo␣final:␣" + str(round(final_cost*100)/100))
print("Melhora␣de␣" + str(round((initial_cost-final_cost)/initial_cost*10000)/100) + "%")

if __name__=='__main__':

    print("Starting␣visualization")


    app = App.App(antennas=[target_antenna, final_array])
    try:
        Ntheta = 91
        Nphi = 181
        field = 'F'
        plot = '2d␣Polar␣Patch'

        tab = ResultFrame.ResultFrame(
            master=app.tabs,
            name=field,
            columns=2, rows=1)
        Result.Result(tab=tab,
                      title='Result',
                      name='Result',
                      antenna=final_array,
                      field=field,
                      plot=plot,
                      ticks_flag=False,
                      in_dB=True,
                      column=1, row=1,
                      Ntheta=Ntheta,
```

```
                Nphi=Nphi)
    Result.Result(tab=tab,
                title='Target',
                name='Target',
                antenna=target_antenna,
                field=field,
                plot=plot,
                ticks_flag=False,
                in_dB=True,
                column=2, row=1,
                Ntheta=Ntheta,
                Nphi=Nphi)
    app.add_tab(tab)

    field = 'Fref'
    plot = '2d␣Polar␣Patch'

    tab = ResultFrame.ResultFrame(
        master=app.tabs, name=field, columns=2, rows=1)
    Result.Result(tab=tab,
                title='Result',
                name='Result',
                antenna=final_array,
                field=field,
                plot=plot,
                ticks_flag=False,
                in_dB=True,
                column=1, row=1,
                Ntheta=Ntheta,
                Nphi=Nphi)
    Result.Result(tab=tab,
                title='Target',
                name='Target',
                antenna=target_antenna,
                field=field,
                plot=plot,
                ticks_flag=False,
                in_dB=True,
                column=2, row=1,
                Ntheta=Ntheta,
                Nphi=Nphi)
    app.add_tab(tab)

    field = 'Fcross'
    plot = '2d␣Polar␣Patch'

    tab = ResultFrame.ResultFrame(
        master=app.tabs, name=field, columns=2, rows=1)
    Result.Result(tab=tab,
                title='Result',
                name='Result',
                antenna=final_array,
                field=field,
                plot=plot,
                ticks_flag=False,
                in_dB=True,
                column=1, row=1,
```

```
                Ntheta=Ntheta,
                Nphi=Nphi)
    Result.Result(tab=tab,
                title='Target',
                name='Target',
                antenna=target_antenna,
                field=field,
                plot=plot,
                ticks_flag=False,
                in_dB=True,
                column=2, row=1,
                Ntheta=Ntheta,
                Nphi=Nphi)
    app.add_tab(tab)

    # Main application loop
    app.mainloop()
except Exception as e:
    # If some error occur, destroy the application to close
    # the window, then show the error
    app.destroy()
    raise e
```

Apêndice 9.76 – exportOptimizationRefNonConverged.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 11 23:48:20 2023

@author: vitinho
"""

import sys
import os
import header

import Result
import ResultFrame
import App
import ExportResults
import Array
import numpy as np
import LoadHFSSYagis
import matplotlib.pyplot as plt

plt.close('all')

print("Loading␣antennas")

antennas = LoadHFSSYagis.run(
    Ntheta=91, Nphi=91)

print("Loading␣export␣results␣directory")

export_directory = os.path.join(
    header.results_dir,
    'Optimization',
    'LinearOptimizationNonConverged')
if not os.path.exists(export_directory):
    os.mkdir(export_directory)

print("Creating␣target␣distribution")


Ntheta = 91
Nphi = 181

theta = np.linspace(0, 180, Ntheta)
phi = np.linspace(-180, 180, Nphi)

target_antenna = Array.Array(
    name='Target␣Antenna',
    theta=theta.copy(),
    phi=phi.copy(),
    antennas=[
        antennas['hfss_yagi2EL'].copy(
        ),
        antennas['hfss_yagi2EL'].copy(
        ),
```

```
        antennas['hfss_yagi2EL'].copy(
        ),
    ])
target_antenna.antennas[0].set_position(
    x=0, y=0, z=0)
target_antenna.antennas[0].set_orientation(
    elevation=-90, azimuth=55)
target_antenna.antennas[1].set_position(
    x=0, y=0.5, z=0)
target_antenna.antennas[1].set_orientation(
    elevation=-90, azimuth=55)
target_antenna.antennas[2].set_position(
    x=0, y=1, z=0)
target_antenna.antennas[2].set_orientation(
    elevation=-90, azimuth=55)
target_antenna.evaluate()

print("Creating␣working␣array")

initial_array = Array.Array(
    name='Initial␣Array',
    theta=theta.copy(),
    phi=phi.copy(),
    antennas=[
        antennas['hfss_yagi2EL'].copy(
        ),
        antennas['hfss_yagi2EL'].copy(
        ),
        antennas['hfss_yagi2EL'].copy(
        ),
    ])
initial_array.antennas[0].set_position(
    x=0, y=0, z=0)
initial_array.antennas[0].set_orientation(
    elevation=0, azimuth=90)
initial_array.antennas[1].set_position(
    x=0, y=0.5, z=0)
initial_array.antennas[1].set_orientation(
    elevation=0, azimuth=30)
initial_array.antennas[2].set_position(
    x=0, y=1.0, z=0)
initial_array.antennas[2].set_orientation(
    elevation=0, azimuth=110)
initial_array.evaluate()

print("Exporting␣initial␣state")


ExportResults.run([
    initial_array,
], export_directory)

print("Optimizing")


def cost_function(target, result):
    total_cost = (np.abs(
```

```
        np.abs(target.Fcross)-np.abs(result.Fcross))*target.sin_mesh_theta).sum()
    total_cost += (np.abs(np.abs(target.Fref) -
                        np.abs(result.Fref))*target.sin_mesh_theta).sum()
    return total_cost

import Optimization

working_array = initial_array.copy()
optim = Optimization.Optimization(
    target_antenna=target_antenna,
    working_array=working_array,
    x_map = [
        dict(
            antenna=working_array.antennas[0],
            variables = [
                # 'x',
                # 'y',
                'elevation',
                'azimuth',
                ],
            ),
        dict(
            antenna=working_array.antennas[1],
            variables = [
                # 'x',
                # 'y',
                'elevation',
                'azimuth',
                ],
            ),
        dict(
            antenna=working_array.antennas[2],
            variables = [
                # 'x',
                # 'y',
                'elevation',
                'azimuth',
                ],
            ),
        ],
    cost_function=cost_function,
    # method = 'L-BFGS-B',
    disp=True,
    )

result = optim.run()
final_array = optim.working_array

print(result)

final_array.name = 'Result␣Array'

print("Exporting␣results")

ExportResults.run([
    target_antenna,
    final_array,
```

```
], export_directory)

import ExportTable

initial_array.name = 'Arranjo␣inicial␣-␣otimização␣linear␣não␣' + \
    'convergida'
target_antenna.name = 'Arranjo␣alvo␣de␣polarização␣linear␣' + \
    '-␣otimização␣não␣convergida'
final_array.name = 'Arranjo␣final␣otimizado␣-␣otimização␣' + \
    'não␣convergida'

ExportTable.export_table(
    export_directory,
    arrays=[
        initial_array,
        target_antenna,
        final_array
        ],
    captions = [
        "Arranjo␣inicial",
        "Arranjo␣alvo",
        "Resultados␣da␣otimização␣de␣polarização␣linear␣" +
            "com␣custo␣final␣de␣{:.2f}".format(optim.cost)
        ])

initial_cost = cost_function(initial_array, target_antenna)
final_cost = cost_function(final_array, target_antenna)
print("Custo␣inicial:␣" + str(round(initial_cost*100)/100))
print("Custo␣final:␣" + str(round(final_cost*100)/100))
print("Melhora␣de␣" + str(round((initial_cost-final_cost)/initial_cost*10000)/100) + "%")

if __name__=='__main__':

    print("Starting␣visualization")


    app = App.App(antennas=[target_antenna, final_array])
    try:
        Ntheta = 91
        Nphi = 181
        field = 'F'
        plot = '2d␣Polar␣Patch'

        tab = ResultFrame.ResultFrame(
            master=app.tabs, name=field, columns=2, rows=1)
        Result.Result(tab=tab,
                      title='Result',
                      name='Result',
                      antenna=final_array,
                      field=field,
                      plot=plot,
                      ticks_flag=False,
                      in_dB=True,
                      column=1, row=1,
                      Ntheta=Ntheta,
                      Nphi=Nphi)
        Result.Result(tab=tab,
```

```
                title='Target',
                name='Target',
                antenna=target_antenna,
                field=field,
                plot=plot,
                ticks_flag=False,
                in_dB=True,
                column=2, row=1,
                Ntheta=Ntheta,
                Nphi=Nphi)
    app.add_tab(tab)

    field = 'Fref'
    plot = '2d␣Polar␣Patch'

    tab = ResultFrame.ResultFrame(
        master=app.tabs, name=field, columns=2, rows=1)
    Result.Result(tab=tab,
                title='Result',
                name='Result',
                antenna=final_array,
                field=field,
                plot=plot,
                ticks_flag=False,
                in_dB=True,
                column=1, row=1,
                Ntheta=Ntheta,
                Nphi=Nphi)
    Result.Result(tab=tab,
                title='Target',
                name='Target',
                antenna=target_antenna,
                field=field,
                plot=plot,
                ticks_flag=False,
                in_dB=True,
                column=2, row=1,
                Ntheta=Ntheta,
                Nphi=Nphi)
    app.add_tab(tab)

    field = 'Fcross'
    plot = '2d␣Polar␣Patch'

    tab = ResultFrame.ResultFrame(
        master=app.tabs, name=field, columns=2, rows=1)
    Result.Result(tab=tab,
                title='Result',
                name='Result',
                antenna=final_array,
                field=field,
                plot=plot,
                ticks_flag=False,
                in_dB=True,
                column=1, row=1,
                Ntheta=Ntheta,
                Nphi=Nphi)
```

```
        Result.Result(tab=tab,
                      title='Target',
                      name='Target',
                      antenna=target_antenna,
                      field=field,
                      plot=plot,
                      ticks_flag=False,
                      in_dB=True,
                      column=2, row=1,
                      Ntheta=Ntheta,
                      Nphi=Nphi)
        app.add_tab(tab)

        # Main application loop
        app.mainloop()
    except Exception as e:
        # If some error occur, destroy the application to close
        # the window, then show the error
        app.destroy()
        raise e
```

Apêndice 9.77 – ExportResults.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 13 16:32:59 2023

@author: 160047412
"""

import sys
import os
path = os.path.split(__file__)[0]
path = os.path.split(path)[0]
sys.path.insert(0, path)
path = os.path.split(path)[0]
home_directory = os.path.split(path)[0]
antennas_dir=os.path.join(home_directory, 'Antennas')

import matplotlib.pyplot as plt

import ResultFigure
import Result

def run(antennas,
       export_directory,
       fields = [
           'F',
           'Fref',
           'Fcross',
           'Ftheta',
           'Fphi'
           ],
       colors = [
           'Color by magnitude',
           'Color by phase'
           ],
       title='',
       Ntheta=91,
       Nphi=181,):

   if not os.path.exists(export_directory):
       os.mkdir(export_directory)

   for antenna in antennas:
       for field in fields:
           for color in colors:
               figure = ResultFigure.ResultFigure()
               plot = '2d Polar Patch'
               Result.Result(tab=figure,
                             title='',
                             antenna=antenna,
                             field=field,
                             color=color,
                             plot=plot,
                             ticks_flag=False,
                             in_dB=True,
                             Ntheta=Ntheta,
```

```
            Nphi=Nphi,)
figure.draw()
cur_export_directory = os.path.join(export_directory,
                                    color)
if not os.path.exists(cur_export_directory):
    os.mkdir(cur_export_directory)
fname = os.path.join(
    cur_export_directory, '␣'.join(
        [s for s \
         in [antenna.name,
             field,
             # color,
             plot,
              title
              ] \
             if s != '']) + '.png')
figure.figure.savefig(fname)
plt.close('all')
```

Apêndice 9.78 – ExportScript.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 13 18:37:32 2023

@author: 160047412
"""

import sys
import os
import header

import matplotlib.pyplot as plt
plt.close('all')

import LoadHFSSYagis
antennas = LoadHFSSYagis.run(
    Ntheta=91, Nphi=91)

import LoadHFSSValidationArrays
antennas.update(LoadHFSSValidationArrays.run(
    Ntheta=91, Nphi=91))

import LoadValidationArrays
antennas.update(LoadValidationArrays.run(
    Ntheta=91, Nphi=91))

import ExportResults
import ExportCompare
import ExportTable

export_directory = os.path.join(header.validation_results_dir,
                                'Fields1Y4EL')
if not os.path.exists(export_directory):
    os.mkdir(export_directory)
ExportResults.run([
    antennas['array_validation_1Y4EL'],
    antennas['HFSS_1Y4EL'],
    ], export_directory)
ExportTable.export_table(
    export_directory,
    arrays=[
        antennas['array_validation_1Y4EL']
        ],
    captions = [
        "Arranjo␣de␣validação␣do␣código␣com␣1␣antena"
        ])
export_directory = os.path.join(header.validation_results_dir,
                                'Fields2Y4EL')
if not os.path.exists(export_directory):
    os.mkdir(export_directory)
ExportResults.run([
    antennas['array_validation_2Y4EL'],
    antennas['HFSS_2Y4EL'],
    ], export_directory)
ExportTable.export_table(
```

```
    export_directory,
    arrays=[
        antennas['array_validation_2Y4EL']
        ],
    captions = [
        "Arranjo␣de␣validação␣do␣código␣com␣2␣antenas"
        ])

export_directory = os.path.join(header.validation_results_dir,
                            'Fields3Y4EL')
if not os.path.exists(export_directory):
    os.mkdir(export_directory)
ExportResults.run([
    antennas['array_validation_3Y4EL'],
    antennas['HFSS_3Y4EL'],
    ],export_directory)
ExportTable.export_table(
    export_directory,
    arrays=[
        antennas['array_validation_3Y4EL']
        ],
    captions = [
        "Arranjo␣de␣validação␣do␣código␣com␣3␣antenas"
        ])

export_directory = os.path.join(header.validation_results_dir,
                            'Fields4Y4EL')
if not os.path.exists(export_directory):
    os.mkdir(export_directory)
ExportResults.run([
    antennas['array_validation_4Y4EL'],
    antennas['HFSS_4Y4EL'],
    ], export_directory)
ExportTable.export_table(
    export_directory,
    arrays=[
        antennas['array_validation_4Y4EL']
        ],
    captions = [
        "Arranjo␣de␣validação␣do␣código␣com␣4␣antenas"
        ])

export_directory = os.path.join(header.validation_results_dir,
                            'Fields5Y4EL')
if not os.path.exists(export_directory):
    os.mkdir(export_directory)
ExportResults.run(
    [
    antennas['array_validation_5Y4EL'],
    antennas['HFSS_5Y4EL']
    ], export_directory)
ExportTable.export_table(
    export_directory,
    arrays=[
        antennas['array_validation_5Y4EL']
        ],
    captions = [
```

```
       "Arranjo␣de␣validação␣do␣código␣com␣5␣antenas"
       ])

export_directory = os.path.join(header.results_dir,
                            'Comparisons')
if not os.path.exists(export_directory):
    os.mkdir(export_directory)
ExportCompare.run([
                        (antennas['array_validation_1Y4EL'],
                        antennas['HFSS_1Y4EL']),
                        (antennas['array_validation_2Y4EL'],
                        antennas['HFSS_2Y4EL']),
                        (antennas['array_validation_3Y4EL'],
                        antennas['HFSS_3Y4EL']),
                        (antennas['array_validation_4Y4EL'],
                        antennas['HFSS_4Y4EL']),
                        (antennas['array_validation_5Y4EL'],
                        antennas['HFSS_5Y4EL'])
                        ], export_directory)

export_directory = os.path.join(header.results_dir,
                            'ReferenceSystemComparison')
if not os.path.exists(export_directory):
    os.mkdir(export_directory)
ExportResults.run(
    [
        antennas['array_validation_3Y4EL'],
        antennas['HFSS_3Y4EL'],
    ],
    export_directory,
    fields=['Fphi','Ftheta','Fref','Fcross', 'F', 'Fref-Fcross'],
    title='ReferenceSystemComparison')
```

Apêndice 9.79 – ExportTable.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 11 14:27:57 2023

@author: vitinho
"""

import os

def export_table(export_directory, arrays, captions):
    with open(os.path.join(export_directory, "table.tex"), 'w') as f:
        print('Exporting␣tables␣in␣' + f.name)
        for array, caption in zip(arrays, captions):
            f.write("\\begin{table}[H]\n")
            f.write("␣␣␣␣\\centering\n")
            f.write("␣␣␣␣\\tiny\n")
            f.write("␣␣␣␣\\begin{tabular}{c|c|c|c|c|c|c|c|c}\n")
            f.write("␣␣␣␣␣␣␣␣␣␣&␣\\multicolumn{3}{|c|}{Posição}␣&␣" +
                    "\\multicolumn{3}{|c|}{Orientação}␣&␣" +
                    "\\multicolumn{2}{|c}{Alimentação}␣\\\\\n")
            f.write("␣␣␣␣␣␣␣␣␣\\hline\n")
            f.write("␣␣␣␣␣␣␣␣␣Antenna␣&␣X␣&␣Y␣&␣Z␣&␣Elevation␣" +
                    "[$\circ$]␣&␣Azimuth␣[$\circ$]␣&␣Roll␣" +
                    "[$\circ$]␣&␣Magnitude␣&␣Phase␣[$\circ$]␣\\\\\n")
            f.write("␣␣␣␣␣␣␣␣␣\\hline\n")
            for i in range(len(array.antennas)):
                x = array.antennas[i].x
                y = array.antennas[i].y
                z = array.antennas[i].z
                e = array.antennas[i].elevation
                a = array.antennas[i].azimuth
                r = array.antennas[i].roll
                m = array.antennas[i].current_magnitude
                p = array.antennas[i].current_phase
                f.write("␣␣␣␣␣␣␣␣␣" + array.antennas[i].name +
                        "␣&␣{x:.2f}".format(x=x) +
                        "␣&␣{y:.2f}".format(y=y) +
                        "␣&␣{z:.2f}".format(z=z) +
                        "␣&␣{e:.2f}".format(e=e) +
                        "␣&␣{a:.2f}".format(a=a) +
                        "␣&␣{r:.2f}".format(r=r) +
                        "␣&␣{m:.2f}".format(m=m) +
                        "␣&␣{p:.2f}".format(p=p) +
                        "␣\\\\\n")
            f.write("␣␣␣␣\\end{tabular}\n")
            f.write("␣␣␣␣\\caption{{{caption}}}\n".format(
                caption=caption))
            f.write("␣␣␣␣\\label{tab:Arranjo␣" + array.name + "}\n")
            f.write("\\end{table}\n")

if __name__=="__main__":
    import sys
    path = os.path.split(os.path.split(__file__)[0])[0]
    sys.path.insert(0, path)
```

```
path = os.path.split(path)[0]
export_directory = os.path.join(
    path,
    'ExportedResults')

import Array
import numpy as np
import Scripts.AntennasLoaders.LoadHFSSYagis

print("Loading␣antennas")

antennas = Scripts.AntennasLoaders.LoadHFSSYagis.run(
    Ntheta=91, Nphi=91)

Ntheta = 91
Nphi = 181

theta = np.linspace(0, 180, Ntheta)
phi = np.linspace(-180, 180, Nphi)

ant = Array.Array(
    name='Target',
    theta=theta.copy(),
    phi=phi.copy(),
    antennas=[
        antennas['hfss_yagi2EL'].copy(
        ),
        antennas['hfss_yagi2EL'].copy(
        ),
        antennas['hfss_yagi2EL'].copy(
        ),
    ])
ant.antennas[0].set_position(x=0, y=0, z=0)
ant.antennas[0].set_orientation(
    elevation=-90, azimuth=90)
ant.antennas[1].set_position(
    x=0, y=0.5, z=0)
ant.antennas[1].set_orientation(
    elevation=-90, azimuth=0)
ant.antennas[2].set_position(x=0, y=1, z=0)
ant.antennas[2].set_orientation(
    elevation=-90, azimuth=120)
ant.evaluate()

export_table(export_directory=export_directory,
             arrays=[ant],
             captions=["Testing"])

import exportOptimizationRef
import exportOptimizationRefNonConverged
import exportOptimizationCircular
import CustomOptimization
```

Apêndice 9.80 – GenerateImages.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Jul 12 17:07:12 2023

@author: vitinho
"""

import sys
import os
import header

python_path = header.python_path
results_dir = os.path.join(header.results_dir, 'Validation')
home_dir = header.home_dir
antennas_dir = header.antennas_dir

import matplotlib.pyplot as plt
plt.close('all')

import Scripts.AntennasLoaders.LoadHFSSYagis
antennas = Scripts.AntennasLoaders.LoadHFSSYagis.run(Ntheta=91, Nphi=91,)
                                                # antennas_dir=antennas_dir)

import Scripts.AntennasLoaders.LoadHFSSValidationArrays
antennas.update(Scripts.AntennasLoaders.LoadHFSSValidationArrays.run(
    Ntheta=91, Nphi=91,))# antennas_dir=antennas_dir))

import Scripts.AntennasLoaders.LoadValidationArrays
antennas.update(Scripts.AntennasLoaders.LoadValidationArrays.run(
    Ntheta=91, Nphi=91))

import Scripts.ExportResults
import Scripts.ExportCompare

if not os.path.exists(results_dir):
    os.mkdir(results_dir)
cur_export_dir = os.path.join(results_dir, 'Fields1Y4EL')
if not os.path.exists(cur_export_dir):
    os.mkdir(cur_export_dir)
Scripts.ExportResults.run([
    antennas['array_validation_1Y4EL'],
    antennas['HFSS_1Y4EL'],
    ], cur_export_dir)
cur_export_dir = os.path.join(results_dir, 'Fields2Y4EL')
Scripts.ExportResults.run([
    antennas['array_validation_2Y4EL'],
    antennas['HFSS_2Y4EL'],
    ], cur_export_dir)
cur_export_dir = os.path.join(results_dir, 'Fields3Y4EL')
Scripts.ExportResults.run([
    antennas['array_validation_3Y4EL'],
    antennas['HFSS_3Y4EL'],
    ],cur_export_dir)
cur_export_dir = os.path.join(results_dir, 'Fields4Y4EL')
```

```
Scripts.ExportResults.run([
    antennas['array_validation_4Y4EL'],
    antennas['HFSS_4Y4EL'],
    ], cur_export_dir)
cur_export_dir = os.path.join(results_dir, 'Fields5Y4EL')
Scripts.ExportResults.run(
    [
    antennas['array_validation_5Y4EL'],
    antennas['HFSS_5Y4EL']
    ], cur_export_dir)

cur_export_dir = os.path.join(results_dir, 'Comparisons')
Scripts.ExportCompare.run([
                          (antennas['array_validation_1Y4EL'],
                          antennas['HFSS_1Y4EL']),
                          (antennas['array_validation_2Y4EL'],
                          antennas['HFSS_2Y4EL']),
                          (antennas['array_validation_3Y4EL'],
                          antennas['HFSS_3Y4EL']),
                          (antennas['array_validation_4Y4EL'],
                          antennas['HFSS_4Y4EL']),
                          (antennas['array_validation_5Y4EL'],
                          antennas['HFSS_5Y4EL'])
                          ], cur_export_dir)

cur_export_dir = os.path.join(results_dir, 'ReferenceSystemComparison')
Scripts.ExportResults.run(
    [
        antennas['array_validation_3Y4EL'],
        antennas['HFSS_3Y4EL'],
    ],
    cur_export_dir,
    fields=['Fphi','Ftheta','Fref','Fcross', 'F', 'Fref-Fcross'],
    title='ReferenceSystemComparison')

import Result
import ResultFigure

figure = ResultFigure.ResultFigure()
antenna = antennas['HFSS_3Y4EL']
field = 'Ftheta-Fphi'
plot = '2d␣Polar␣Patch'
Result.Result(tab=figure,
             title='',
             antenna=antenna,
             field=field,
             plot=plot,
             in_dB=True)
figure.draw()
fname = os.path.join(results_dir, antenna.name +
                  '␣RefSysCompare-Ftheta-Fphi' + '.png')
figure.figure.savefig(fname)

figure = ResultFigure.ResultFigure()
antenna = antennas['HFSS_3Y4EL']
field = 'Fref-Fcross'
plot = '2d␣Polar␣Patch'
```

```
Result.Result(tab=figure,
              title='',
              antenna=antenna,
              field=field,
              plot=plot,
              in_dB=True)
figure.draw()
fname = os.path.join(results_dir, antenna.name +
                   '␣RefSysCompare-Fref-Fcross' + '.png')
figure.figure.savefig(fname)

plt.close('all')
```

Apêndice 9.81 – Geometry.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon May 1 23:06:40 2023

@author: 160047412
"""

import numpy as np

epsilon = 1e-6

def cross_product(V1, V2):
    x = V1.y*V2.z - V1.z*V2.y
    y = V1.z*V2.x - V1.x*V2.z
    z = V1.x*V2.y - V1.y*V2.x
    return Vector(x=x,y=y,z=z)

def dot_product(V1, V2):
    return V1.x*V2.x+V1.y*V2.y+V1.z*V2.z

def norm(V):
    return np.sqrt(V.x*V.x + V.y*V.y + V.z*V.z)

def normalize(V):
    norm_V = norm(V)
    if norm_V > epsilon:
        V.x /= norm_V
        V.y /= norm_V
        V.z /= norm_V

class Geometry:
    pass

class Point(Geometry):
    def __init__(self, x=0.0, y=0.0, z=0.0):
        self.x=x
        self.y=y
        self.z=z

    def to_string(self):
        return str(self.x) + '␣' + str(self.y) + '␣' + str(self.z)

class Vector(Geometry):
    def __init__(self, x=0.0, y=0.0, z=0.0):
        self.x=x
        self.y=y
        self.z=z

    def to_string(self):
        return str(self.x) + '␣' + str(self.y) + '␣' + str(self.z)

class Plane(Geometry):
    def __init__(self, x=0.0, y=0.0, z=1.0):
        self.x = x
        self.y = y
```

```
        self.z = z

class Axis(Geometry):
    def __init__(self, x=1.0, y=0.0, z=0.0):
        self.x=x
        self.y=y
        self.z=z

    def to_string(self):
        return str(self.x) + ' ' + str(self.y) + ' ' + str(self.z)

class ReferenceSystem(Geometry):
    def __init__(self,
                 x_axis=Axis(),
                 z_axis=Axis(x=0.0,z=1.0),
                 origin=Point()):
        self.x_axis=x_axis
        self.z_axis=z_axis

        normalize(self.x_axis)

        y_vector = cross_product(z_axis, x_axis)
        normalize(y_vector)
        self.y_axis = Axis(x=y_vector.x,y=y_vector.y,z=y_vector.z)

        z_vector = cross_product(self.x_axis, self.y_axis)
        normalize(z_vector)
        self.z_axis = Axis(x=z_vector.x,y=z_vector.y,z=z_vector.z)

        self.R = np.array([[1,0,0],[0,1,0],[0,0,1]])
        self.elevation=0.0
        self.azimuth=0.0

        self.calculate_rotation_matrix()

    def calculate_rotation_matrix(self):
        xx = self.x_axis.x
        xy = self.x_axis.y
        xz = self.x_axis.z
        zx = self.z_axis.x
        zy = self.z_axis.y
        zz = self.z_axis.z

        self.elevation = np.arctan2(np.sqrt(zy*zy+zx*zx), zz)
        self.azimuth = np.arctan2(xy, xx)

        ct = np.cos(self.elevation)
        st = np.sin(self.elevation)
        Relevation = np.zeros((3, 3))
        Relevation[0,0] = ct
        Relevation[0,2] = -st
        Relevation[2,0] = st
        Relevation[1,1] = 1
        Relevation[2,2] = ct

        cp = np.cos(self.azimuth)
        sp = np.sin(self.azimuth)
```

```
        Razimuth = np.zeros((3, 3))
        Razimuth[0,0] = cp
        Razimuth[0,1] = -sp
        Razimuth[1,0] = sp
        Razimuth[1,1] = cp
        Razimuth[2,2] = 1

        self.R = Razimuth@Relevation

    # def calculate_transform_matrix(self):
    # s_mesh_theta = np.arctan2(np.sqrt(s_y*s_y+s_x*s_x), s_z)

    # # ids1 = s_mesh_theta>3*np.pi/4
    # # ids2 = s_mesh_theta<np.pi/4

    # # s_x[ids1] = s_hat_phi[ids1,1]
    # # s_y[ids1] = -s_hat_phi[ids1,0]
    # # s_x[ids2] = s_hat_phi[ids2,1]
    # # s_y[ids2] = s_hat_phi[ids2,0]
    # s_mesh_phi = np.arctan2(s_y, s_x)

if __name__=='__main__':
    sq2 = np.sqrt(2)/2
    sq3 = np.sqrt(3)/3
    x = Axis(x=sq2*sq2,y=sq2*sq2,z=sq2)
    z = Axis(x=0.0,z=1.0)
    ref = ReferenceSystem(x_axis=x,z_axis=z)

    y = cross_product(z, x)
    normalize(y)

    z = cross_product(x, y)
    normalize(z)

    angles = np.radians(np.linspace(-180,180,181))
    points = np.zeros((181,3))
    for i in range(181):
        points[i,0] = np.cos(angles[i])
        points[i,1] = np.sin(angles[i])
        points[i,:] = ref.R@points[i,:]

    print(z.to_string())
    print('')
    print(ref.x_axis.to_string())
    print(ref.y_axis.to_string())
    print(ref.z_axis.to_string())
    print('')
    print(np.degrees(ref.elevation))
    print(np.degrees(ref.azimuth))
    print('')
    print(norm(ref.x_axis))
    print(norm(ref.y_axis))
    print(norm(ref.z_axis))
    print('')
    print(points)

    import matplotlib.pyplot as plt
```

```
ax = plt.figure().add_subplot(projection='3d')

ax.quiver(0, 0, 0, x.x, x.y, x.z, length=0.1, normalize=True, color='b')
ax.quiver(0, 0, 0, y.x, y.y, y.z, length=0.1, normalize=True, color='b')
ax.quiver(0, 0, 0, z.x, z.y, z.z, length=0.1, normalize=True, color='b')

for i in range(181):
    ax.quiver(0,0,0, points[i,0], points[i,1], points[i,2], length=0.1, normalize=True, color
        ↪ ='r')

plt.show()
```

Apêndice 9.82 – header.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 24 00:44:10 2023

@author: vitinho
"""

import sys
import os
src_dir = os.path.split(__file__)[0]
results_dir = os.path.join(src_dir, 'Images')
sys.path.insert(0, src_dir)

path = os.path.split(src_dir)[0]
antennas_dir = os.path.join(path, 'Antennas')

if not os.path.exists(results_dir):
    os.mkdir(results_dir)

validation_results_dir = os.path.join(results_dir, "Validation")
if not os.path.exists(validation_results_dir):
    os.mkdir(validation_results_dir)

optimization_results_dir = os.path.join(results_dir, "Optimization")
if not os.path.exists(optimization_results_dir):
    os.mkdir(optimization_results_dir)

# comparisons_results_dir = os.path.join(results_dir, "Comparisons")
# if not os.path.exists(comparisons_results_dir):
# os.mkdir(comparisons_results_dir)

# reference_system_comparison_results_dir = os.path.join(results_dir, "ReferenceSystemComparison
    ↪ ")
# if not os.path.exists(comparisons_results_dir):
# os.mkdir(reference_system_comparison_results_dir)
```

Apêndice 9.83 – ImportAntennasPath.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 22 15:00:55 2023

@author: vitinho
"""

import sys
import os
path = os.path.split(__file__)[0]
path = os.path.split(path)[0]
path = os.path.split(path)[0]
sys.path.insert(0, path)
path = os.path.split(path)[0]
path = os.path.split(path)[0]
path = os.path.split(path)[0]
home_directory = os.path.split(path)[0]
antennas_dir=os.path.join(home_directory, 'Antennas')
```

Apêndice 9.84 – LoadBasicAntennas.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 8 21:19:07 2023

@author: 160047412
"""

import sys
import os
import ImportAntennasPath
home_directory = ImportAntennasPath.home_directory
antennas_dir = ImportAntennasPath.antennas_dir

import numpy as np

import Antenna

def run(elevation=0, azimuth=0, roll=0, Ntheta=91, Nphi=91):

    theta=np.linspace(0, 180, Ntheta)
    phi=np.linspace(-180, 180, Nphi)

    ideal_dipole = Antenna.Antenna(name='Ideal␣Dipole',
                       theta=theta.copy(),
                       phi=phi.copy(),
                       evaluate_as='ideal␣dipole')
    ideal_dipole.evaluate()

    ideal_loop_dipole = Antenna.Antenna(name='Ideal␣Loop␣Dipole',
                       theta=theta.copy(),
                       phi=phi.copy(),
                       evaluate_as='ideal␣loop␣dipole')
    ideal_loop_dipole.evaluate()

    antennas = dict(ideal_dipole=ideal_dipole,
                    ideal_loop_dipole=ideal_loop_dipole,)

    return antennas

if __name__=='__main__':
    antennas = run()
```

Apêndice 9.85 – LoadCompareAntennas.py

```
# -*- coding: utf-8 -*-
"""
Created on Fri May 12 15:08:15 2023

@author: 160047412
"""

import ResultFrame
import Result

def load_compare_graphs(app, name, antennas, titles):
    tab = ResultFrame.ResultFrame(
        master=app.tabs,
        name=name,
        columns=2,
        rows=2)
    for i,antenna,title in zip(range(2),antennas,titles):
        # Result.Result(tab=tab,
        # name='F ' + title,
        # antenna=antenna,field='F',
        # plot='3d Polar',
        # in_dB=True,
        # position=i+1)
        Result.Result(tab=tab,
                      name='Fref␣' + title,
                      title='Fref␣' + title,
                      antenna=antenna,field='Fref',
                      plot='2d␣Polar␣Patch',
                      in_dB=True,
                      position= i+1)
        Result.Result(tab=tab,
                      name='Fcross␣' + title,
                      title='Fcross␣' + title,
                      antenna=antenna,field='Fcross',
                      plot='2d␣Polar␣Patch',
                      in_dB=True,
                      position= i+1 + 2)
        # Result.Result(tab=tab,
        # name='Ftheta ' + title,
        # title='Ftheta ' + title,
        # antenna=antenna,field='Ftheta',
        # plot='2d Polar Patch',
        # in_dB=True,
        # position=i+1 + 4)
        # Result.Result(tab=tab,
        # name='Fphi ' + title,
        # title='Fphi ' + title,
        # antenna=antenna,field='Fphi',
        # plot='2d Polar Patch',
        # in_dB=True,
        # position=i+1 + 5)
        # Result.Result(tab=tab,
        # name='Frhcp ' + title,
        # title='Frhcp ' + title,
        # antenna=antenna,field='Frhcp',
```

```
        # plot='2d Polar Patch',
        # in_dB=True,
        # position=i+1 + 6)
        # Result.Result(tab=tab,
        # name='Flhcp ' + title,
        # title='Flhcp ' + title,
        # antenna=antenna,field='Flhcp',
        # plot='2d Polar Patch',
        # in_dB=True,
        # position=i+1 + 7)
    app.add_tab(tab)
```

Apêndice 9.86 – LoadCustomArrays.py

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 24 22:41:12 2023

@author: 160047412
"""

import sys
import os
import header

import numpy as np

import LoadHFSSYagis

import Array

def run(elevation=0, azimuth=0, roll=0,
        Ntheta=91, Nphi=91,
        antennas_dict=None):

    theta=np.linspace(0, 90, Ntheta)
    phi=np.linspace(-180, 180, Nphi)

    if antennas_dict==None:
        antennas_dict=LoadHFSSYagis.run(
            Ntheta=Ntheta,Nphi=Nphi)

    hfss_yagi3EL = antennas_dict['hfss_yagi3EL']
    hfss_yagi3EL_V = antennas_dict['hfss_yagi3EL_V']

    array_H = Array.Array(name='Optimized Array H',
                          theta=theta,
                          phi=phi,
                          antennas=[hfss_yagi3EL.copy() \
                                    for i in range(4)])
    array_H.antennas[0].set_current(magnitude=-1)
    array_H.antennas[0].set_orientation(azimuth=180)
    array_H.antennas[1].set_current(magnitude=-1)
    array_H.antennas[1].set_orientation(azimuth=180)
    array_H.evaluate()

    array_V = Array.Array(name='Optimized Array V',
                          theta=theta,
                          phi=phi,
                          antennas=[hfss_yagi3EL_V.copy() \
                                    for i in range(4)])
    array_V.antennas[0].set_current(magnitude=-1)
    array_V.antennas[0].set_orientation(azimuth=180)
    array_V.antennas[1].set_current(magnitude=-1)
    array_V.antennas[1].set_orientation(azimuth=180)
    array_V.evaluate()

    array_RHCP = Array.Array(name='Optimized Array RHCP',
                          theta=theta,
```

```
                    phi=phi,
                    antennas=[hfss_yagi3EL_V.copy(),
                              hfss_yagi3EL.copy(),
                              hfss_yagi3EL_V.copy(),
                              hfss_yagi3EL.copy(),])
array_RHCP.antennas[0].set_current(magnitude=-1)
array_RHCP.antennas[0].set_orientation(azimuth=180)
array_RHCP.antennas[1].set_current(magnitude=-1)
array_RHCP.antennas[1].set_orientation(azimuth=180)
array_RHCP.evaluate()

antennas_dict['array_H']=array_H
antennas_dict['array_V']=array_V
antennas_dict['array_RHCP']=array_RHCP

return antennas_dict
```

Apêndice 9.87 – LoadDefaultGraphs.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu May 11 00:58:16 2023

@author: 160047412
"""

import sys
import os
path = os.path.split(__file__)[0]
path = os.path.split(path)[0]
sys.path.insert(0, path)
path = os.path.split(path)[0]
home_directory = os.path.split(path)[0]
antennas_dir=os.path.join(home_directory, 'Antennas')

import ResultFrame
import Result

def load_default_graphs(app, name, antenna,
                        fields=['F',
                                'Fref',
                                'Fcross',
                                # 'Ftheta',
                                # 'Fphi',
                                # 'Frhcp',
                                # 'Flhcp',
                                ]):
    plots = dict(
        # F='3d Polar Surface',
        F='2d Polar Patch',
        Fref='2d Polar Patch',
        Fcross='2d Polar Patch',
        Ftheta='2d Polar Patch',
        Fphi='2d Polar Patch',
        Frhcp='2d Polar Patch',
        Flhcp='2d Polar Patch')
    pos = dict(F='i',
               Fref='i',
               Fcross='i',
               Ftheta='i',
               Fphi='i',
               Frhcp='i',
               Flhcp='i')
    tab = ResultFrame.ResultFrame(
        master=app.tabs,
        name=name,
        columns=3,
        rows=1)
    i = 1
    for field in fields:
        if pos[field]=='i':
            Result.Result(tab=tab,
                          name=field,
                          title=field,
```

```
                    antenna=antenna,field=field,
                    plot=plots[field],
                    in_dB=True,
                    position=i)
        i += 1
    else:
        Result.Result(tab=tab,
                    name=field,
                    title=field,
                    antenna=antenna,field=field,
                    plot=plots[field],
                    in_dB=True,
                    position=pos[field])
        i = pos[field][-1]+1
app.add_tab(tab)
```

Apêndice 9.88 – LoadHFSSValidationArrays.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 13 20:04:57 2023

@author: 160047412
"""

import sys
import os
import header

import numpy as np

import Antenna

def run(Ntheta=91, Nphi=91):

    theta=np.linspace(0, 180, Ntheta)
    phi=np.linspace(-180, 180, Nphi)

    antenna_path = os.path.join(header.antennas_dir, '1Y-4EL.csv')
    HFSS_1Y4EL = Antenna.Antenna(name='HFSS 1Y-4EL',
                                 theta=theta,
                                 phi=phi)
    HFSS_1Y4EL.set_evaluation_method('load file')
    HFSS_1Y4EL.evaluation_arguments['file path'] = antenna_path
    HFSS_1Y4EL.evaluation_arguments['load mesh from file'] = False
    HFSS_1Y4EL.set_orientation(elevation=0, azimuth=0, roll=0)
    HFSS_1Y4EL.evaluate()

    antenna_path = os.path.join(header.antennas_dir, '2Y-4EL.csv')
    HFSS_2Y4EL = Antenna.Antenna(name='HFSS 2Y-4EL',
                                 theta=theta,
                                 phi=phi)
    HFSS_2Y4EL.set_evaluation_method('load file')
    HFSS_2Y4EL.evaluation_arguments['file path'] = antenna_path
    HFSS_2Y4EL.evaluation_arguments['load mesh from file'] = False
    HFSS_2Y4EL.set_orientation(elevation=0, azimuth=0, roll=0)
    HFSS_2Y4EL.evaluate()

    antenna_path = os.path.join(header.antennas_dir, '3Y-4EL.csv')
    HFSS_3Y4EL = Antenna.Antenna(name='HFSS 3Y-4EL',
                                 theta=theta,
                                 phi=phi)
    HFSS_3Y4EL.set_evaluation_method('load file')
    HFSS_3Y4EL.evaluation_arguments['file path'] = antenna_path
    HFSS_3Y4EL.evaluation_arguments['load mesh from file'] = False
    HFSS_3Y4EL.set_orientation(elevation=0, azimuth=0, roll=0)
    HFSS_3Y4EL.evaluate()

    antenna_path = os.path.join(header.antennas_dir, '4Y-4EL.csv')
    HFSS_4Y4EL = Antenna.Antenna(name='HFSS 4Y-4EL',
                                 theta=theta,
                                 phi=phi)
    HFSS_4Y4EL.set_evaluation_method('load file')
```

```
    HFSS_4Y4EL.evaluation_arguments['file␣path'] = antenna_path
    HFSS_4Y4EL.evaluation_arguments['load␣mesh␣from␣file'] = False
    HFSS_4Y4EL.set_orientation(elevation=0, azimuth=0, roll=0)
    HFSS_4Y4EL.evaluate()

    antenna_path = os.path.join(header.antennas_dir, '5Y-4EL.csv')
    HFSS_5Y4EL = Antenna.Antenna(name='HFSS␣5Y-4EL',
                                 theta=theta,
                                 phi=phi)
    HFSS_5Y4EL.set_evaluation_method('load␣file')
    HFSS_5Y4EL.evaluation_arguments['file␣path'] = antenna_path
    HFSS_5Y4EL.evaluation_arguments['load␣mesh␣from␣file'] = False
    HFSS_5Y4EL.set_orientation(elevation=0, azimuth=0, roll=0)
    HFSS_5Y4EL.evaluate()

    antennas = dict(HFSS_1Y4EL=HFSS_1Y4EL,
                    HFSS_2Y4EL=HFSS_2Y4EL,
                    HFSS_3Y4EL=HFSS_3Y4EL,
                    HFSS_4Y4EL=HFSS_4Y4EL,
                    HFSS_5Y4EL=HFSS_5Y4EL,)

    return antennas

if __name__=='__main__':
    antennas = run()
```

Apêndice 9.89 – LoadHFSSYagis.py

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 24 22:43:33 2023

@author: 160047412
"""

import os
import header

import numpy as np

import Antenna

def run(elevation=0, azimuth=0, roll=0, Ntheta=91, Nphi=91):

    theta=np.linspace(0, 180, Ntheta)
    phi=np.linspace(-180, 180, Nphi)

    antennas_dict = dict()


    antenna_path = os.path.join(
        header.antennas_dir,
        'antenna-Dipole.csv')
    hfss_yagi1EL = Antenna.Antenna(name='Antena Dipolo',
                      theta=theta.copy(),
                      phi=phi.copy())
    hfss_yagi1EL.set_evaluation_method('load file')
    hfss_yagi1EL.evaluation_arguments['file path'] = antenna_path
    hfss_yagi1EL.evaluation_arguments['load mesh from file'] = False
    hfss_yagi1EL.set_orientation(
        elevation=elevation,
        azimuth=azimuth,
        roll=roll)
    hfss_yagi1EL.evaluate()
    hfss_yagi1EL_V = hfss_yagi1EL.copy()
    hfss_yagi1EL_V.set_orientation(roll=90);
    hfss_yagi1EL_V.set_current(phase=90);
    hfss_yagi1EL_V.name += " V"
    hfss_yagi1EL_V.evaluate()

    antenna_path = os.path.join(
        header.antennas_dir,
        'antenna-Yagi-2Elements.csv')
    hfss_yagi2EL = Antenna.Antenna(
        name='Yagi 2 elementos',
        theta=theta.copy(),
        phi=phi.copy())
    hfss_yagi2EL.set_evaluation_method('load file')
    hfss_yagi2EL.evaluation_arguments['file path'] = antenna_path
    hfss_yagi2EL.evaluation_arguments['load mesh from file'] = False
    hfss_yagi2EL.set_orientation(
        elevation=elevation,
        azimuth=azimuth,
```

```
    roll=roll)
hfss_yagi2EL.evaluate()
hfss_yagi2EL_V = hfss_yagi2EL.copy()
hfss_yagi2EL_V.set_orientation(roll=90);
hfss_yagi2EL_V.set_current(phase=90);
hfss_yagi2EL_V.name += "␣V"
hfss_yagi2EL_V.evaluate()

antenna_path = os.path.join(
    header.antennas_dir,
    'antenna-Yagi-3Elements.csv')
hfss_yagi3EL = Antenna.Antenna(name='Yagi␣3␣elementos',
                  theta=theta.copy(),
                  phi=phi.copy())
hfss_yagi3EL.set_evaluation_method('load␣file')
hfss_yagi3EL.evaluation_arguments['file␣path'] = antenna_path
hfss_yagi3EL.evaluation_arguments['load␣mesh␣from␣file'] = False
hfss_yagi3EL.set_orientation(
    elevation=elevation,
    azimuth=azimuth,
    roll=roll)
hfss_yagi3EL.evaluate()
hfss_yagi3EL_V = hfss_yagi3EL.copy()
hfss_yagi3EL_V.set_orientation(roll=90);
hfss_yagi3EL_V.set_current(phase=90);
hfss_yagi3EL_V.name += "␣V"
hfss_yagi3EL_V.evaluate()

antenna_path = os.path.join(
    header.antennas_dir,
    'antenna-Yagi-4Elements.csv')
hfss_yagi4EL = Antenna.Antenna(
    name='Yagi␣4␣elementos',
    theta=theta.copy(),
    phi=phi.copy())
hfss_yagi4EL.set_evaluation_method('load␣file')
hfss_yagi4EL.evaluation_arguments['file␣path'] = antenna_path
hfss_yagi4EL.evaluation_arguments['load␣mesh␣from␣file'] = False
hfss_yagi4EL.set_orientation(
    elevation=elevation,
    azimuth=azimuth,
    roll=roll)
hfss_yagi4EL.evaluate()
hfss_yagi4EL_V = hfss_yagi4EL.copy()
hfss_yagi4EL_V.set_orientation(roll=90);
hfss_yagi4EL_V.set_current(phase=90);
hfss_yagi4EL_V.name += "␣V"
hfss_yagi4EL_V.evaluate()

antennas_dict['hfss_yagi1EL']=hfss_yagi1EL
antennas_dict['hfss_yagi2EL']=hfss_yagi2EL
antennas_dict['hfss_yagi3EL']=hfss_yagi3EL
antennas_dict['hfss_yagi4EL']=hfss_yagi4EL
antennas_dict['hfss_yagi1EL_V']=hfss_yagi1EL_V
antennas_dict['hfss_yagi2EL_V']=hfss_yagi2EL_V
antennas_dict['hfss_yagi3EL_V']=hfss_yagi3EL_V
antennas_dict['hfss_yagi4EL_V']=hfss_yagi4EL_V
```

```
    return antennas_dict

if __name__=='__main__':
    antennas = run()
```

Apêndice 9.90 – LoadValidationArrays.py

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 24 22:46:38 2023

@author: 160047412
"""

import sys
import os
import header

import numpy as np

import LoadHFSSYagis

import Array

def run(elevation=0, azimuth=0, roll=0,
        Ntheta=91, Nphi=91,
        antennas_dict=None):

    theta=np.linspace(0, 180, Ntheta)
    phi=np.linspace(-180, 180, Nphi)

    if antennas_dict==None:
        antennas_dict=LoadHFSSYagis.run(
            Ntheta=Ntheta,
            Nphi=Nphi)

    hfss_yagi4EL = antennas_dict['hfss_yagi4EL']

    array_validation_1Y4EL = Array.Array(name='1Y-4EL',
                                         theta=theta,
                                         phi=phi,
                                         antennas=[
                                             hfss_yagi4EL.copy(),
                                             ])
    array_validation_1Y4EL.antennas[0].set_orientation(
        roll=0,
        elevation=0,
        azimuth=0)
    array_validation_1Y4EL.antennas[0].set_position(x=0, y=0, z=0)
    array_validation_1Y4EL.evaluate()

    array_validation_2Y4EL = Array.Array(name='2Y-4EL',
                                         theta=theta,
                                         phi=phi,
                                         antennas=[
                                             hfss_yagi4EL.copy(),
                                             hfss_yagi4EL.copy(),
                                             ])
    array_validation_2Y4EL.antennas[0].set_orientation(
        roll=0,
        elevation=0,
        azimuth=0)
```

```
array_validation_2Y4EL.antennas[0].set_position(x=0, y=0, z=0)
array_validation_2Y4EL.antennas[1].set_orientation(
    roll=0,
    elevation=0,
    azimuth=45)
array_validation_2Y4EL.antennas[1].set_position(x=0, y=1.5, z=0)
array_validation_2Y4EL.evaluate()

array_validation_3Y4EL = Array.Array(name='3Y-4EL',
                                     theta=theta,
                                     phi=phi,
                                     antennas=[
                                         hfss_yagi4EL.copy(),
                                         hfss_yagi4EL.copy(),
                                         hfss_yagi4EL.copy(),
                                         ])
array_validation_3Y4EL.antennas[0].set_orientation(
    roll=0,
    elevation=0,
    azimuth=0)
array_validation_3Y4EL.antennas[0].set_position(x=0, y=0, z=0)
array_validation_3Y4EL.antennas[1].set_orientation(
    roll=0,
    elevation=0,
    azimuth=45)
array_validation_3Y4EL.antennas[1].set_position(x=0, y=1.5, z=0)
array_validation_3Y4EL.antennas[2].set_orientation(
    roll=0,
    elevation=-45,
    azimuth=-45)
array_validation_3Y4EL.antennas[2].set_position(x=0, y=-1.5, z=0)
array_validation_3Y4EL.evaluate()

array_validation_4Y4EL = Array.Array(name='4Y-4EL',
                                     theta=theta,
                                     phi=phi,
                                     antennas=[
                                         hfss_yagi4EL.copy(),
                                         hfss_yagi4EL.copy(),
                                         hfss_yagi4EL.copy(),
                                         hfss_yagi4EL.copy(),
                                         ])
array_validation_4Y4EL.antennas[0].set_orientation(
    roll=0, elevation=0, azimuth=0)
array_validation_4Y4EL.antennas[0].set_position(x=0, y=0, z=0)
array_validation_4Y4EL.antennas[1].set_orientation(
    roll=0, elevation=0, azimuth=45)
array_validation_4Y4EL.antennas[1].set_position(x=0, y=1.5, z=0)
array_validation_4Y4EL.antennas[2].set_orientation(
    roll=0, elevation=-45, azimuth=-45)
array_validation_4Y4EL.antennas[2].set_position(x=0, y=-1.5, z=0)
array_validation_4Y4EL.antennas[3].set_orientation(
    roll=0, elevation=-45, azimuth=135)
array_validation_4Y4EL.antennas[3].set_position(
    x=0.34, y=-3.14, z=1.423)
array_validation_4Y4EL.evaluate()
```

```
    array_validation_5Y4EL = Array.Array(name='5Y-4EL',
                                         theta=theta,
                                         phi=phi,
                                         antennas=[
                                             hfss_yagi4EL.copy(),
                                             hfss_yagi4EL.copy(),
                                             hfss_yagi4EL.copy(),
                                             hfss_yagi4EL.copy(),
                                             hfss_yagi4EL.copy(),
                                             ])
    array_validation_5Y4EL.antennas[0].set_orientation(
        roll=0, elevation=0, azimuth=0)
    array_validation_5Y4EL.antennas[0].set_position(
        x=0, y=0, z=0)
    array_validation_5Y4EL.antennas[1].set_orientation(
        roll=0, elevation=0, azimuth=45)
    array_validation_5Y4EL.antennas[1].set_position(
        x=0, y=1.5, z=0)
    array_validation_5Y4EL.antennas[2].set_orientation(
        roll=0, elevation=-45, azimuth=-45)
    array_validation_5Y4EL.antennas[2].set_position(
        x=0, y=-1.5, z=0)
    array_validation_5Y4EL.antennas[3].set_orientation(
        roll=0, elevation=-45, azimuth=135)
    array_validation_5Y4EL.antennas[3].set_position(
        x=0.34, y=-3.14, z=1.423)
    array_validation_5Y4EL.antennas[4].set_orientation(
        roll=0, elevation=72, azimuth=14)
    array_validation_5Y4EL.antennas[4].set_position(
        x=0.83, y=1.19, z=-0.72)
    array_validation_5Y4EL.evaluate()

    antennas_dict['array_validation_1Y4EL']=array_validation_1Y4EL
    antennas_dict['array_validation_2Y4EL']=array_validation_2Y4EL
    antennas_dict['array_validation_3Y4EL']=array_validation_3Y4EL
    antennas_dict['array_validation_4Y4EL']=array_validation_4Y4EL
    antennas_dict['array_validation_5Y4EL']=array_validation_5Y4EL

    return antennas_dict

if __name__=='__main__':
    antennas = run()
```

Apêndice 9.91 – MyMath.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 1 18:16:37 2023

@author: 160047412
"""

import numpy as np

sqrt2 = np.sqrt(2)


def rotx(angle):
    c = np.cos(angle)
    s = np.sin(angle)
    R = np.zeros((angle.shape[0], angle.shape[1], 3, 3))
    R[:, :, 0, 0] = 1
    R[:, :, 0, 1] = 0
    R[:, :, 0, 2] = 0
    R[:, :, 1, 0] = 0
    R[:, :, 1, 1] = c
    R[:, :, 1, 2] = -s
    R[:, :, 2, 0] = 0
    R[:, :, 2, 1] = s
    R[:, :, 2, 2] = c

    return R


def roty(angle):
    c = np.cos(angle)
    s = np.sin(angle)
    R = np.zeros((angle.shape[0], angle.shape[1], 3, 3))
    R[:, :, 0, 0] = c
    R[:, :, 0, 1] = 0
    R[:, :, 0, 2] = s
    R[:, :, 1, 0] = 0
    R[:, :, 1, 1] = 1
    R[:, :, 1, 2] = 0
    R[:, :, 2, 0] = -s
    R[:, :, 2, 1] = 0
    R[:, :, 2, 2] = c

    return R


def rotz(angle):
    c = np.cos(angle)
    s = np.sin(angle)
    R = np.zeros((angle.shape[0], angle.shape[1], 3, 3))
    R[:, :, 0, 0] = c
    R[:, :, 0, 1] = -s
    R[:, :, 0, 2] = 0
    R[:, :, 1, 0] = s
    R[:, :, 1, 1] = c
```

```
    R[:, :, 1, 2] = 0
    R[:, :, 2, 0] = 0
    R[:, :, 2, 1] = 0
    R[:, :, 2, 2] = 1

    return R


def rotate(vec, R):
    new_vec = np.zeros_like(vec)
    R_swapped = R.swapaxes(2, 3)
    new_vec[:, :, 0] = (vec*R_swapped[:, :, :, 0]).sum(2)
    new_vec[:, :, 1] = (vec*R_swapped[:, :, :, 1]).sum(2)
    new_vec[:, :, 2] = (vec*R_swapped[:, :, :, 2]).sum(2)
    return new_vec
```

Apêndice 9.92 – NumpyExpressionParser.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 9 20:24:44 2023

@author: 160047412
"""

import pyparsing as pp
import numpy as np


class EvalTuple:
    "Class␣to␣evaluate␣a␣parsed␣tuple"

    def __init__(self, tokens):
        self.values = [i for j in tokens for i in j]

    def eval(self):
        return tuple([arg.eval() for arg in self.values])


class EvalList:
    "Class␣to␣evaluate␣a␣parsed␣list"

    def __init__(self, tokens):
        self.values = [i for j in tokens for i in j]

    def eval(self):
        return [arg.eval() for arg in self.values]


class EvalFunction:
    "Class␣to␣evaluate␣a␣parsed␣function"
    funs_ = {
        'array': np.array,
        'ndarray': np.ndarray,
        'linspace': np.linspace,
        'arange': np.arange,
        'meshgrid': np.meshgrid,
        'sin': np.sin,
        'cos': np.cos,
        'tan': np.tan,
        'asin': np.arcsin,
        'acos': np.arccos,
        'atan': np.arctan,
        'atan2': np.arctan2,
        'sinh': np.sinh,
        'cosh': np.cosh,
        'tanh': np.tanh,
        'sinc': np.sinc,
        'exp': np.exp,
        'degrees': np.degrees,
        'radians': np.radians,
        'abs': np.absolute,
        'angle': np.angle,
```

```
        'mod': np.mod,
        'real': np.real,
        'imag': np.imag,
        'conj': np.conj,
        'max': np.max,
        'min': np.min,
        'sqrt': np.sqrt,
        'cbrt': np.cbrt,
        'log': np.log,
        'log10': np.log10,
        'log2': np.log2,
        'in_db': lambda G: 20*np.log10(G),
        'sign': np.sign,
        'interp': np.interp,
        'floor': np.floor,
        'ceil': np.ceil,
        'sum': np.sum,
        'cumsum': np.cumsum,
        'prod': np.prod,
        'U': lambda a, t: 1*(np.array(t) >= np.array(a))
    }

    def __init__(self, tokens):
        self.name = tokens.pop(0)
        self.values = [i for j in tokens for i in j]

    def eval(self):
        if self.name in EvalFunction.funs_:
            args = [arg[0].eval() for arg in self.values]
            return EvalFunction.funs_[self.name](*args)
        else:
            raise Exception('Unknown function ' + str(self.name))


class EvalConstant:
    "Class to evaluate a parsed constant or variable"
    vars_ = {
        'pi': np.pi,
        'e': np.e
    }

    def __init__(self, tokens):
        self.value = tokens[0]

    def eval(self):
        if self.value in EvalConstant.vars_:
            return EvalConstant.vars_[self.value]
        else:
            try:
                return int(self.value)
            except ValueError:
                try:
                    return float(self.value)
                except ValueError:
                    return complex(self.value)
```

```
class EvalSignOp:
    "Class␣to␣evaluate␣expressions␣with␣a␣leading␣+␣or␣-␣sign"

    def __init__(self, tokens):
        self.sign, self.value = tokens[0]

    def eval(self):
        mult = {"+": 1, "-": -1}[self.sign]
        return mult * self.value.eval()


def operatorOperands(tokenlist):
    "generator␣to␣extract␣operators␣and␣operands␣in␣pairs"
    it = iter(tokenlist)
    while 1:
        try:
            yield (next(it), next(it))
        except StopIteration:
            break


class EvalPowerOp:
    "Class␣to␣evaluate␣power␣expressions"

    def __init__(self, tokens):
        self.value = tokens[0]

    def eval(self):
        res = self.value[-1].eval()
        for val in self.value[-3::-2]:
            res = val.eval() ** res
        return res


class EvalMultOp:
    "Class␣to␣evaluate␣multiplication␣and␣division␣expressions"

    def __init__(self, tokens):
        self.value = tokens[0]

    def eval(self):
        prod = self.value[0].eval()
        for op, val in operatorOperands(self.value[1:]):
            if op == "*":
                prod *= val.eval()
            if op == "/":
                prod /= val.eval()
        return prod


class EvalAddOp:
    "Class␣to␣evaluate␣addition␣and␣subtraction␣expressions"

    def __init__(self, tokens):
        self.value = tokens[0]

    def eval(self):
```

```
        value = self.value[0].eval()
        for op, val in operatorOperands(self.value[1:]):
            if op == "+":
                value = value + val.eval()
            if op == "-":
                value = value - val.eval()
        return value


class EvalFactOp:
    "Class␣to␣evaluate␣factorial␣expressions"

    def __init__(self, tokens):
        self.value = tokens[0]

    def eval(self):
        N = self.value[0].eval()
        prod = 1
        for i in range(1, N+1):
            prod *= i
        return prod


class EvalComparisonOp:
    "Class␣to␣evaluate␣comparison␣expressions"
    opMap = {
        "<": lambda a, b: a < b,
        "<=": lambda a, b: a <= b,
        ">": lambda a, b: a > b,
        ">=": lambda a, b: a >= b,
        "!=": lambda a, b: a != b,
        "=": lambda a, b: a == b,
        "LT": lambda a, b: a < b,
        "LE": lambda a, b: a <= b,
        "GT": lambda a, b: a > b,
        "GE": lambda a, b: a >= b,
        "NE": lambda a, b: a != b,
        "EQ": lambda a, b: a == b,
        "<>": lambda a, b: a != b,
    }

    def __init__(self, tokens):
        self.value = tokens[0]

    def eval(self):
        val1 = self.value[0].eval()
        for op, val in operatorOperands(self.value[1:]):
            fn = EvalComparisonOp.opMap[op]
            val2 = val.eval()
            val1 = fn(val1, val2)
        return val1


class NumpyExpressionParser:
    integer = pp.Word(pp.nums)
    real = pp.Combine(pp.Optional(
        integer) + '.' + integer) | pp.Combine(integer + '.' +
```

```
                                    pp.Optional(integer)) | integer
imaginary = pp.Combine(real + 'j')
cmplx = pp.Combine(real + '+' + imaginary)
variable = pp.Word(pp.alphanums + '_')
number = cmplx | imaginary | real | integer

signop = pp.oneOf("+ -")
multop = pp.oneOf("* /")
plusop = pp.oneOf("+ -")
expop = pp.Literal("**")

operand = pp.Forward()
expression = pp.Forward()
expression_list = pp.delimitedList(pp.Group(expression))
lpar = pp.Literal('(').suppress()
rpar = pp.Literal(')').suppress()
fn_call = pp.Word(pp.alphanums + '_') + lpar + \
    pp.Optional(pp.Group(expression_list)) + rpar

lpar = pp.Literal('(').suppress()
rpar = pp.Literal(')').suppress()
tpl = lpar + pp.Group(expression) + pp.Literal(',').suppress() + \
    pp.Optional(
    pp.delimitedList(pp.Group(expression), allow_trailing_delim=True)) + \
    rpar

lpar = pp.Literal('[').suppress()
rpar = pp.Literal(']').suppress()
lst = lpar + pp.Group(expression) + pp.Literal(',').suppress() + \
    pp.Optional(
    pp.delimitedList(pp.Group(expression), allow_trailing_delim=True)) + \
    rpar

operand <<= number | fn_call | lst | tpl | variable

number.setParseAction(EvalConstant)
variable.setParseAction(EvalConstant)
tpl.setParseAction(EvalTuple)
lst.setParseAction(EvalList)
fn_call.setParseAction(EvalFunction)

comparisonop = pp.oneOf("< <= > >= != = <> LT GT LE GE EQ NE")
arith_expr = pp.infixNotation(
    operand,
    [
        ("!", 1, pp.opAssoc.LEFT, EvalFactOp),
        (expop, 2, pp.opAssoc.RIGHT, EvalPowerOp),
        (signop, 1, pp.opAssoc.RIGHT, EvalSignOp),
        (multop, 2, pp.opAssoc.LEFT, EvalMultOp),
        (plusop, 2, pp.opAssoc.LEFT, EvalAddOp),
        (comparisonop, 2, pp.opAssoc.LEFT, EvalComparisonOp),
    ],
)

comp_expr = pp.infixNotation(
    operand,
    [
```

```
        ],
    )

    expression <<= arith_expr

    @staticmethod
    def eval(expression, variables={}, functions={}):
        for var_name in variables.keys():
            EvalConstant.vars_[var_name] = variables[var_name]

        for fun_name in functions.keys():
            EvalFunction.funs_[fun_name] = functions[fun_name]

        return NumpyExpressionParser.expression.parseString(expression)[0].\
            eval()


if __name__ == '__main__':
    theta = np.radians(np.linspace(0, 180, 21))
    phi = np.radians(np.linspace(-180, 180, 21))
    mesh_phi, mesh_theta = np.meshgrid(phi, theta)
    v = {
        'theta': mesh_theta,
        'phi': mesh_phi,
        'pi': np.pi
    }
    f = {

    }
    s = '(cos(pi/2*cos(theta))␣-␣cos(pi/2))/(sin(theta)+(sin(theta)==0))'
    print(s + '␣=␣' + str(NumpyExpressionParser.eval(expression=s,
                                                    variables=v,
                                                    functions=f)))
    s = '[1,1.0,.1,1.]'
    print(s + '␣=␣' + str(NumpyExpressionParser.eval(expression=s,
                                                    variables=v,
                                                    functions=f)))
```

Apêndice 9.93 – Optimization.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 6 15:08:48 2023

@author: Vitinho
"""

import time
import numpy as np
import scipy.optimize


class Optimization:
    methods = [
        'Nelder-Mead',
        'Powell',
        'CG',
        'BFGS',
        'Newton-CG',
        'L-BFGS-B',
        'TNC',
        'COBYLA',
        'SLSQP',
        'trust-ngc',
        'trust-exact',
        'trust-krylov'
    ]

    def __init__(self,
                 cost_function,
                 x_map=None,
                 name='new optimization',
                 method='L-BFGS-B',
                 working_array=None, target_antenna=None,
                 # analyses=None,weights=None,
                 weight_mask=1,
                 options={
                     # 'disp':True,
                     'eps': 0.01,
                     'gtol': 0.1,
                     'xrtol': 0.1,
                     'maxiter': 30
                 },
                 disp=False):
        self.cost_function = cost_function
        self.name = name
        self.method = method
        self.working_array = working_array
        self.target_antenna = target_antenna
        self.x_map = x_map
        self.options = options
        self.disp = disp

        self.sensitivity_x = 1
        self.sensitivity_y = 1
```

```
        self.sensitivity_z = 1

        self.result = None
        self.lowest_cost = None
        self.lowest_x = None

        self.state = 'up to date'
        self.listeners = []

    def assert_variables(self, x):
        for entry, x_val in zip(self.working_x_map, x):
            entry['v_cb'](entry['antenna'], x_val)
            entry['antenna'].ok = False

        self.working_array.local_field_flag = True
        self.working_array.ok = False
        self.working_array.evaluate()
        self.number_of_evaluations += 1

    def cost_function_helper(self, x, *args):
        self.assert_variables(x)

        self.cost = self.cost_function(self.target_antenna, self.working_array)
        if self.disp:
            print('evaluating with ' + str(x) + ' cost ' + str(self.cost))

        if self.lowest_cost is None:
            self.lowest_cost = self.cost
            self.lowest_x = x
        elif self.cost < self.lowest_cost:
            self.lowest_cost = self.cost
            self.lowest_x = x
        return self.cost

    def run(self):
        if not self.target_antenna.ok:
            self.target_antenna.evaluate()
        self.start_time = time.time()

        self.working_x_map = []
        x = []
        bounds = []
        for entry in self.x_map:
            antenna = entry['antenna']
            for variable in entry['variables']:
                if variable == 'elevation':
                    v_cb = self.set_elevation
                    x.append((antenna.elevation/180)+0.5)
                    bounds.append((0.0, 1.0))
                elif variable == 'azimuth':
                    v_cb = self.set_azimuth
                    x.append((antenna.azimuth/360)+0.5)
                    bounds.append((0.0, 1.0))
                elif variable == 'roll':
                    v_cb = self.set_roll
                    x.append((antenna.roll/360)+0.5)
                    bounds.append((0.0, 1.0))
```

```
            elif variable == 'x':
                v_cb = self.set_x
                x.append(antenna.x/self.sensitivity_x)
                bounds.append((None, None))
            elif variable == 'y':
                v_cb = self.set_y
                x.append(antenna.y/self.sensitivity_y)
                bounds.append((None, None))
            elif variable == 'z':
                v_cb = self.set_z
                x.append(antenna.z/self.sensitivity_z)
                bounds.append((None, None))
            elif variable == 'current magnitude':
                v_cb = self.set_current_magnitude
                x.append(antenna.current_magnitude)
                bounds.append((0.0, None))
            elif variable == 'current phase':
                v_cb = self.set_current_phase
                x.append((antenna.current_phase/360)+0.5)
                bounds.append((0.0, 1.0))
            self.working_x_map.append(dict(
                antenna=antenna,
                v_cb=v_cb
            ))
    self.number_of_evaluations = 0
    self.result = scipy.optimize.minimize(fun=self.cost_function_helper,
                                          x0=np.array(x),
                                          method=self.method,
                                          bounds=bounds,
                                          options=self.options)
    self.cost = self.result.fun
    self.x = self.result.x
    self.assert_variables(self.result.x)

    self.elapsed_time = time.time()-self.start_time

    if self.disp:
        print('elapsed time was ' + str(self.elapsed_time))
        print('number of evaluations was ' +
              str(self.number_of_evaluations))
        print('final cost: {}'.format(self.result.fun))
        print('final array have {N} antennas:'.format(
            N=len(self.working_array.antennas)))
        for i in range(len(self.working_array.antennas)):
            antenna = self.working_array.antennas[i]
            print('\tantenna {i}: '.format(i=i) + antenna.name)
            print('\t\televation: {e}'.format(e=antenna.elevation))
            print('\t\tazimuth: {a}'.format(a=antenna.azimuth))
            print('\t\troll: {a}'.format(a=antenna.roll))
            print('\t\tx: {x}'.format(x=antenna.x))
            print('\t\ty: {y}'.format(y=antenna.y))
            print('\t\tz: {z}'.format(z=antenna.z))
            print('\t\tcurrent magnitude: {magnitude}'.format(
                magnitude=antenna.current_magnitude))
            print('\t\tcurrent phase: {phase}'.format(
                phase=antenna.current_phase))
```

```
        return self.result

    def set_elevation(self, antenna, x):
        antenna.set_orientation(elevation=(x-0.5)*180)

    def set_azimuth(self, antenna, x):
        antenna.set_orientation(azimuth=(x-0.5)*360)

    def set_roll(self, antenna, x):
        antenna.set_orientation(roll=(x-0.5)*360)

    def set_x(self, antenna, x):
        antenna.set_position(x=self.sensitivity_x*x)

    def set_y(self, antenna, y):
        antenna.set_position(y=self.sensitivity_y*y)

    def set_z(self, antenna, z):
        antenna.set_position(z=self.sensitivity_z*z)

    def set_current_magnitude(self, antenna, magnitude):
        antenna.set_current(magnitude=magnitude)

    def set_current_phase(self, antenna, phase):
        antenna.set_current(phase=(phase-0.5)*180)
```

Apêndice 9.94 – OrientationDefinition.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 18 19:03:59 2023

@author: vitinho
"""

import sys
import os
import header
antennas_dir = header.antennas_dir
results_dir= header.results_dir
# path = os.path.split(__file__)[0]
# path = os.path.split(path)[0]
# sys.path.insert(0, path)
# import Path_To_Antennas
# antennas_dir = Path_To_Antennas.antennas_dir
# path = os.path.split(__file__)[0]
# path = os.path.split(path)[0]
# sys.path.insert(0, path)
# path = os.path.split(path)[0]
# home_directory = os.path.split(path)[0]
# antennas_dir=os.path.join(home_directory, 'Antennas')

import numpy as np

import Antenna
import Array

import matplotlib.pyplot as plt

plt.close('all')

theta=np.linspace(0, 180, 91)
phi=np.linspace(-180, 180, 91)

file_name = 'antenna-Yagi-4Elements.csv'
file_path = os.path.join(antennas_dir, file_name)
hfss_yagi4EL = Antenna.load_from_file(file_path,
                              name='Yagi 4EL',
                              theta=theta,
                              phi=phi,
                              load_mesh_from_file=False)

array_rotated = Array.Array(
    name='Array 1Y4El',
    theta=theta.copy(),
    phi=phi.copy(),
    antennas=[hfss_yagi4EL])
array_rotated.antennas[0].set_orientation(roll=60, elevation=-30,
                                      azimuth=-130)
array_rotated.evaluate()

plot='3d Polar'
```

```
field='F'
color='Color by magnitude'
in_dB = False

export_directory = os.path.join(results_dir,
                            'Orientation')
if not os.path.exists(export_directory):
    os.mkdir(export_directory)

import ResultFigure
import Result

figure = ResultFigure.ResultFigure()
Result.Result(tab=figure,
             title='',
             antenna=array_rotated,
             field=field,
             color=color,
             plot=plot,
             # ticks_flag=False,
             xtick_labels=[],
             ytick_labels=[],
             ztick_labels=[],
             xlabel='x',
             ylabel='y',
             zlabel='z',
             in_dB=True,
             Ntheta=91,
             Nphi=91,
             # axis_flag=False,
             view_camera=[30,-60,0],
             )
figure.draw()
a = array_rotated.antennas[0]
fname = os.path.join(
    export_directory, 'e={}, a={}, r={}'.format(a.elevation,
                                          a.azimuth,
                                          a.roll) + '.png')
figure.figure.savefig(fname)


array= Array.Array(
    name='Array 1Y4El',
    theta=theta.copy(),
    phi=phi.copy(),
    antennas=[hfss_yagi4EL])
array.antennas[0].set_orientation(roll=0, elevation=0,
                                  azimuth=0)
array.evaluate()

figure = ResultFigure.ResultFigure()
Result.Result(tab=figure,
             title='',
             antenna=array,
             field=field,
             color=color,
             plot=plot,
```

```
            # ticks_flag=False,
            xtick_labels=[],
            ytick_labels=[],
            ztick_labels=[],
            xlabel='x',
            ylabel='y',
            zlabel='z',
            in_dB=True,
            Ntheta=91,
            Nphi=91,
            # axis_flag=True,
            view_camera=[30,-60,0],
            )
figure.draw()
a = array.antennas[0]
fname = os.path.join(
    export_directory, 'e={}, a={}, r={}'.format(a.elevation,
                                        a.azimuth,
                                        a.roll) + '.png')

figure.figure.savefig(fname)

plt.close('all')
```

Apêndice 9.95 – Path_To_Antennas.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 11 21:14:00 2023

@author: vitinho
"""

import sys
import os
path = os.path.split(__file__)[0]
path = os.path.split(path)[0]
path = os.path.split(path)[0]
path = os.path.split(path)[0]
home_directory = os.path.split(path)[0]
antennas_dir=os.path.join(home_directory, 'Antennas')
```

Apêndice 9.96 – ProjectTreeview.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 26 19:01:06 2023

@author: 160047412
"""

import tkinter as tk
from tkinter import ttk
import pickle

import CreateMenu

import Antenna
import AntennaEditorFrame
import Array
import ArrayEditorFrame
import ResultFrame
import Result
import ResultEditorFrame


class ProjectTreeview(ttk.Treeview):
    def __init__(self, app, master=None, **kw):
        ttk.Treeview.__init__(self, master, **kw)
        self.app = app

        self.bind('<<TreeviewSelect>>', self.tree_view_select)

        self.antennas = dict()
        self.analyses = dict()
        # self.optims = dict()
        self.tabs = dict()

        self.antennas_iid = self.insert('', tk.END, text='Antennas')
        self.tabs_iid = self.insert('', tk.END, text='Tabs')

        self.object_iid_map = dict()
        self.iid_object_map = dict()

        self.menu = CreateMenu.CreateMenu(self)

    def tree_view_select(self, event):
        pass
        # print(event)
        # print(self.selection())

    def create_drop_menu(self, tw, event):
        self.selection = self.identify_row(event.y)
        self.selection_set(self.selection)
        if self.selection == '':
            return False

        self.obj = None
        if self.selection == self.antennas_iid:
```

```
            self.antennas_menu(tw)
        elif self.selection == self.tabs_iid:
            self.tabs_menu(tw)
        else:
            self.obj = self.iid_object_map[self.selection]
            obj_type = str(type(self.obj))
            if obj_type == "<class␣'Antenna.Antenna'>":
                self.antenna_menu(tw)
            elif obj_type == "<class␣'Array.Array'>":
                self.array_menu(tw)
            elif obj_type == "<class␣'ResultFrame.ResultFrame'>":
                self.tab_menu(tw)
            elif obj_type == "<class␣'Result.Result'>":
                self.result_menu(tw)
            else:
                print(obj_type)
                return False
        return True

    def antennas_menu(self, tw):
        new_menu = tk.Frame(master=tw)
        tk.Label(master=new_menu, text='Antennas', justify='left',
                relief='solid', borderwidth=0).pack(ipadx=1, fill=tk.BOTH)
        tk.Button(master=new_menu, text='new␣antenna',
                  command=self.on_antenna_ppp).pack(ipadx=1, fill=tk.BOTH)
        tk.Button(master=new_menu, text='new␣array',
                  command=self.on_array_ppp).pack(ipadx=1, fill=tk.BOTH)
        tk.Button(master=new_menu, text='load␣antenna',
                  command=self.on_load_antenna).pack(ipadx=1, fill=tk.BOTH)
        tk.Button(master=new_menu, text='update␣all',
                  command=self.on_update_all_antennas).pack(ipadx=1,
                                                            fill=tk.BOTH)
        new_menu.pack(ipadx=1)

    def tabs_menu(self, tw):
        new_menu = tk.Frame(master=tw)
        tk.Label(master=new_menu, text='Result␣tabs', justify='left',
                relief='solid', borderwidth=0).pack(ipadx=1, fill=tk.BOTH)
        tk.Button(master=new_menu, text='new␣tab',
                  command=self.on_new_tab).pack(ipadx=1, fill=tk.BOTH)
        tk.Button(master=new_menu, text='update␣all',
                  command=self.on_update_all_tabs).pack(ipadx=1, fill=tk.BOTH)
        new_menu.pack(ipadx=1)

    def antenna_menu(self, tw):
        new_menu = tk.Frame(master=tw)
        tk.Label(master=new_menu, text='Antenna', justify='left',
                relief='solid', borderwidth=0).pack(ipadx=1, fill=tk.BOTH)
        tk.Button(master=new_menu, text='edit',
                  command=self.on_antenna_ppp).pack(ipadx=1, fill=tk.BOTH)
        tk.Button(master=new_menu, text='update',
                  command=self.on_update_antenna).pack(ipadx=1, fill=tk.BOTH)
        tk.Button(master=new_menu, text='save',
                  command=self.on_save_antenna).pack(ipadx=1, fill=tk.BOTH)
        tk.Button(master=new_menu, text='delete',
                  command=self.on_delete_obj).pack(ipadx=1, fill=tk.BOTH)
        new_menu.pack(ipadx=1)
```

```
def array_menu(self, tw):
    new_menu = tk.Frame(master=tw)
    tk.Label(master=new_menu, text='Array', justify='left',
            relief='solid', borderwidth=0).pack(ipadx=1, fill=tk.BOTH)
    tk.Button(master=new_menu, text='edit',
              command=self.on_array_ppp).pack(ipadx=1, fill=tk.BOTH)
    tk.Button(master=new_menu, text='update',
              command=self.on_update_antenna).pack(ipadx=1, fill=tk.BOTH)
    tk.Button(master=new_menu, text='delete',
              command=self.on_delete_obj).pack(ipadx=1, fill=tk.BOTH)
    new_menu.pack(ipadx=1)

def tab_menu(self, tw):
    new_menu = tk.Frame(master=tw)
    tk.Label(master=new_menu, text='Results', justify='left',
            relief='solid', borderwidth=0).pack(ipadx=1, fill=tk.BOTH)
    # tk.Button(master=new_menu,
    # text='edit', command=self.on_tab_ppp).pack(
    # ipadx=1, fill=tk.BOTH)
    tk.Button(master=new_menu, text='rename',
              command=self.on_rename).pack(ipadx=1, fill=tk.BOTH)
    tk.Button(master=new_menu, text='new␣result',
              command=self.on_new_result).pack(ipadx=1, fill=tk.BOTH)
    tk.Button(master=new_menu, text='update',
              command=self.on_update_all_results).pack(
                  ipadx=1, fill=tk.BOTH)
    tk.Button(master=new_menu, text='delete',
              command=self.on_delete_obj).pack(ipadx=1, fill=tk.BOTH)
    new_menu.pack(ipadx=1)

def result_menu(self, tw):
    new_menu = tk.Frame(master=tw)
    tk.Label(master=new_menu, text='Results', justify='left',
            relief='solid', borderwidth=0).pack(ipadx=1, fill=tk.BOTH)
    tk.Button(master=new_menu, text='edit',
              command=self.on_new_result).pack(ipadx=1, fill=tk.BOTH)
    tk.Button(master=new_menu, text='update',
              command=self.on_update_result).pack(ipadx=1, fill=tk.BOTH)
    tk.Button(master=new_menu, text='delete',
              command=self.on_delete_obj).pack(ipadx=1, fill=tk.BOTH)
    new_menu.pack(ipadx=1)

def on_rename(self):
    self.menu.hidetip()
    root = tk.Toplevel()
    variable = tk.StringVar(value=self.obj.name)
    fr = tk.LabelFrame(master=root, text="Rename␣" + str(type(self.obj)))
    fr.pack(side=tk.TOP, fill=tk.BOTH)
    tk.Entry(master=fr, textvariable=variable).pack(
        side=tk.LEFT, fill=tk.BOTH)

    def on_done():
        root.destroy()
        self.obj.name = variable.get()
        self.item(self.selection, text=self.obj.name)
        self.app.tabs.add(self.obj, text=self.obj.name)
```

```
        tk.Button(master=fr, text="OK", command=on_done).pack(
            side=tk.RIGHT, fill=tk.BOTH)
        root.mainloop()

    def on_antenna_ppp(self):
        self.menu.hidetip()
        root = tk.Toplevel()
        if self.obj is None:
            antenna = Antenna.Antenna(constants=self.app.constants)
            editing = False
        else:
            antenna = self.obj
            editing = True

        def on_done():
            if not editing:
                self.app.add_antenna(antenna)
            else:
                self.item(self.selection, text=antenna.name)
            self.item(self.antennas_iid, open=True)
            root.destroy()

        def on_cancel():
            root.destroy()
        AntennaEditorFrame.AntennaEditorFrame(
            antenna=antenna, master=root, on_done=on_done,
            on_cancel=on_cancel).pack()
        root.mainloop()

    def on_load_antenna(self):
        file_path = tk.filedialog.askopenfilename()
        if file_path == '':
            return
        with open(file_path, mode='rb') as f:
            antenna = pickle.load(f)
            if str(type(antenna)) == "<class␣'Antenna.Antenna'>":
                self.app.add_antenna(antenna)
                self.item(self.antennas_iid, open=True)
            elif str(type(antenna)) == "<class␣'Array.Array'>":
                self.app.add_antenna(antenna)
                self.item(self.antennas_iid, open=True)

    def on_save_antenna(self):
        with tk.filedialog.asksaveasfile(mode='wb') as f:
            antenna = self.obj
            pickle.dump(antenna, f)

    def on_array_ppp(self):
        self.menu.hidetip()
        root = tk.Toplevel()
        if self.obj is None:
            array = Array.Array(constants=self.app.constants)
            editing = False
        else:
            array = self.obj
            editing = True
```

```
        def on_done():
            if not editing:
                self.app.add_antenna(array)
            else:
                self.item(self.selection, text=array.name)
            self.item(self.antennas_iid, open=True)
            root.destroy()

        def on_cancel():
            root.destroy()
        ArrayEditorFrame.ArrayEditorFrame(
            array=array, app=self.app, master=root, on_done=on_done,
            on_cancel=on_cancel).pack()
        root.mainloop()

    def on_new_tab(self):
        self.menu.hidetip()
        tab = ResultFrame.ResultFrame(master=self.app.tabs)
        self.app.add_tab(tab)
        self.item(self.tabs_iid, open=True)

    def on_tab_ppp(self):
        self.menu.hidetip()
        root = tk.Toplevel()
        if self.obj is None:
            tab = ResultFrame.ResultFrame(master=self.app.tabs)
            editing = False
        else:
            tab = self.obj
            editing = True

        def on_done():
            if not editing:
                self.app.add_tab(tab)
            else:
                self.item(self.selection, text=tab.name)
                self.app.tabs.add(tab, text=tab.name)
            self.item(self.tabs_iid, open=True)
            root.destroy()

        def on_cancel():
            if not editing:
                tab.destroy()
            root.destroy()
        ResultEditorFrame.ResultEditorFrame(
            app=self.app, tab=tab, master=root, on_done=on_done,
            on_cancel=on_cancel).pack()
        root.mainloop()

    def on_new_result(self):
        self.menu.hidetip()
        root = tk.Toplevel()
        if str(type(self.obj)) == "<class 'ResultFrame.ResultFrame'>":
            result = Result.Result(tab=self.obj)
            editing = False
        else:
            result = self.obj
```

```
            editing = True

        def on_done():
            if not editing:
                self.app.add_result(tab=self.obj, result=result)
            else:
                self.item(self.selection, text=result.name)
                # self.app.results.add(result,text=result.name)
            self.item(self.object_iid_map[self.obj], open=True)
            root.destroy()

        def on_cancel():
            root.destroy()
        ResultEditorFrame.ResultEditorFrame(
            app=self.app, result=result, master=root, on_done=on_done,
            on_cancel=on_cancel).pack()
        root.mainloop()

    def on_result_ppp(self):
        self.menu.hidetip()
        root = tk.Toplevel()
        if type(self.obj) is ResultFrame:
            result = Result()
            editing = False
        else:
            result = self.obj
            editing = True

        def on_done():
            if not editing:
                self.app.add_result(self.obj, result)
            self.item(self.tabs_iid, open=True)
            root.destroy()

        def on_cancel():
            root.destroy()
        ResultEditorFrame.ResultEditorFrame(
            result=result, app=self.app, master=root, on_done=on_done,
            on_cancel=on_cancel).pack()
        root.mainloop()

    def on_delete_obj(self):
        self.menu.hidetip()
        obj_type = str(type(self.obj))
        if obj_type == "<class␣'Antenna.Antenna'>":
            self.app.antennas.remove(self.obj)
            self.delete_antenna(self.obj)
        elif obj_type == "<class␣'Array.Array'>":
            self.app.antennas.remove(self.obj)
            self.delete_antenna(self.obj)
        elif obj_type == "<class␣'Analysis.Analysis'>":
            self.app.analyses.remove(self.obj)
            self.delete_analysis(self.obj)
        elif obj_type == "<class␣'Optimization.Optimization'>":
            self.app.optims.remove(self.obj)
            self.delete_optim(self.obj)
        elif obj_type == "<class␣'ResultFrame.ResultFrame'>":
```

```
            self.app.tabs.forget(self.obj)
            self.delete_tab(self.obj)
        elif obj_type == "<class␣'Result.Result'>":
            self.delete_result(self.obj)

    def add_antenna(self, antenna):
        iid = self.insert(self.antennas_iid, tk.END, text=antenna.name)
        self.iid_object_map[iid] = antenna
        self.object_iid_map[antenna] = iid

    def add_tab(self, tab):
        iid = self.insert(self.tabs_iid, tk.END, text=tab.name)
        self.iid_object_map[iid] = tab
        self.object_iid_map[tab] = iid

    def add_result(self, tab, result):
        tab_iid = self.object_iid_map[tab]
        result_iid = self.insert(tab_iid, tk.END, text=result.name)
        self.iid_object_map[result_iid] = result
        self.object_iid_map[result] = result_iid

    def delete_antenna(self, antenna):
        iid = self.object_iid_map[antenna]
        self.object_iid_map.pop(antenna)
        self.iid_object_map.pop(iid)
        self.delete(iid)

    def delete_tab(self, tab):
        iid = self.object_iid_map[tab]
        for result in tab.results:
            self.delete_result(result)
        self.object_iid_map.pop(tab)
        self.iid_object_map.pop(iid)
        self.delete(iid)

    def delete_result(self, result):
        result.tab.remove_result(result)

        iid = self.object_iid_map[result]
        self.object_iid_map.pop(result)
        self.iid_object_map.pop(iid)
        self.delete(iid)

    def on_update_antenna(self):
        self.menu.hidetip()
        self.obj.evaluate()

    def on_update_result(self):
        self.menu.hidetip()
        self.obj.ok = False
        self.obj.update()

    def on_update_all_antennas(self):
        self.menu.hidetip()
        for antenna in self.app.antennas:
            antenna.evaluate()
```

```
    def on_update_all_tabs(self):
        self.menu.hidetip()
        for tab in self.app.result_tabs:
            tab.ok = False
            tab.update()
            # tab.canvas.draw()

    def on_update_all_results(self):
        self.menu.hidetip()
        self.obj.ok = False
        self.obj.update()

    def on_run_optim(self):
        self.obj.run()
```

Apêndice 9.97 – Result.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 23 00:43:16 2023

@author: 160047412
"""

import tkinter as tk
import numpy as np
from scipy.interpolate import griddata

import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib import cm

import Geometry

# font = {
       # 'family':'normal',
       # 'weight':'bold',
# 'size':12}
# mpl.rc('font', **font)
degree_sign = u"\N{DEGREE␣SIGN}"


class Result():
    available_plots = [
        # '2d Graph',
        # '2d Contour',
        '3d␣Surface',
        # '2d Polar Graph',
        # '2d Polar Contour',
        '2d␣Polar␣Patch',
        '2d␣Polar␣Patch␣Type␣2',
        '3d␣Polar␣Surface',
        '3d␣Polar'
        ]
    available_fields = [
        'F',
        'Ftheta',
        'Fphi',
        'Frhcp',
        'Flhcp',
        'Fref',
        'Fcross',
        'Ftheta-phi',
        'Fref-cross'
        ]
    # plot_projections = ['2d', '2d', '2d', '3d', '2d', '2d', ]

    def __init__(self, tab, name='New␣result', title='Title',
                 antenna=None, analysis=None,
                 field='F', color='Color␣by␣magnitude',
                 plot='2d␣Polar␣Patch',
                 in_dB=True,
```

```
            dynamic_scaling_dB=-30,
            visible_flag=True,
            axis_flag=True,
            grid_flag=True,
            xaxis_flag=True,
            yaxis_flag=True,
            ticks_flag=True,
            xticks=None,
            yticks=None,
            zticks=None,
            rticks=None,
            xtick_labels=None,
            ytick_labels=None,
            ztick_labels=None,
            xlabel=None,
            ylabel=None,
            zlabel=None,
            add_colorbar=True,
            colorbar_min=0,
            colorbar_max=1,
            colorbar_dB_min=-30,
            colorbar_dB_max=0,
            position=None,
            column=1, row=1,
            compare_fields=None,
            view_camera=None,
            Ntheta=6,
            theta_i=0,
            theta_f=90,
            Nphi=6,
            phi_i=-180,
            phi_f=180,
            Nx=21,
            Ny=21,
            antialiased=True,
            reference_2d=Geometry.ReferenceSystem()):
        self.name = name
        self.title = title
        self.tab = tab
        self.antenna = antenna
        self.analysis = analysis
        self.plot = plot
        self.field = field
        self.color_by = color
        self.in_dB = in_dB
        self.colorbar_min = colorbar_min
        self.colorbar_max = colorbar_max
        self.colorbar_dB_min = colorbar_dB_min
        self.colorbar_dB_max = colorbar_dB_max
        self.visible_flag = visible_flag
        self.axis_flag = axis_flag
        self.grid_flag = grid_flag
        self.xaxis_flag = xaxis_flag
        self.yaxis_flag = yaxis_flag
        self.ticks_flag = ticks_flag
        self.xticks = xticks
        self.yticks = yticks
```

```
        self.zticks = zticks
        self.rticks = rticks
        self.xtick_labels = xtick_labels
        self.ytick_labels = ytick_labels
        self.ztick_labels = ztick_labels
        self.xlabel = xlabel
        self.ylabel = ylabel
        self.zlabel = zlabel
        self.add_colorbar = add_colorbar
        self.colorbar_min = colorbar_min
        self.colorbar_max = colorbar_max
        self.position = position
        self.column = column
        self.row = row
        self.compare_fields = compare_fields
        self.view_camera = view_camera
        self.reference_2d = reference_2d
        self.Ntheta = Ntheta
        self.theta_i = theta_i
        self.theta_f = theta_f
        self.Nphi = Nphi
        self.phi_i = phi_i
        self.phi_f = phi_f
        self.Nx = Nx
        self.Ny = Ny
        self.antialiased = antialiased
        self.labelsize = 20
        if not self.in_dB:
            self.colorbar_max = 1
            self.colorbar_min = 0

        self.colorbar_label = ''
        self.listeners = []
        self.ok = False

        self.domain = None

        if self.antenna is not None:
            self.antenna.listeners.append(self)

        self.axes = None
        self.projection = None
        self.properties = dict()
        self.graphical_objects = None
        self.translate = False

        self.custom_draw = None

        self.tab.add_result(self)

    def notify(self, caller, event):
        self.ok = False

    def mark_update(self, event):
        for l in self.listeners:
            l.notify(self, event)
```

```
def get_axes(self, **kw):
    self.properties = dict()

    if self.plot == '2d Graph':
        self.projection = '2dcartesian'
        self.axes = self.tab.request_axes(requester=self,
                                          projection=None,
                                          position=self.position,
                                          column=self.column,
                                          row=self.row,
                                          **kw)
        self.properties = {
            'x label': 'x',
            'y label': 'y',
        }
    elif self.plot == '2d Contour':
        self.projection = '2dcartesian'
        self.axes = self.tab.request_axes(requester=self,
                                          projection=None,
                                          position=self.position,
                                          column=self.column,
                                          row=self.row,
                                          **kw)
        self.properties = {
            'x label': 'x',
            'y label': 'y',
        }
    elif self.plot == '3d Surface':
        self.projection = '3dcartesian'
        self.axes = self.tab.request_axes(requester=self,
                                          projection='3d',
                                          position=self.position,
                                          column=self.column,
                                          row=self.row,
                                          **kw)
        self.properties = {
            'x label': 'x',
            'y label': 'y',
            'z label': 'z',
        }
    elif self.plot == '2d Polar Graph':
        self.projection = '2dpolar'
        self.axes = self.tab.request_axes(requester=self,
                                          projection='polar',
                                          position=self.position,
                                          column=self.column,
                                          row=self.row,
                                          **kw)
    elif self.plot == '2d Polar Contour':
        self.projection = '2dpolar'
        self.axes = self.tab.request_axes(requester=self,
                                          projection='polar',
                                          position=self.position,
                                          column=self.column,
                                          row=self.row,
                                          **kw)
    elif self.plot == '2d Polar Patch' or \
```

```
            self.plot == '2d␣Polar␣Patch␣Type␣1':
            self.projection = '2dpolar'
            self.axes = self.tab.request_axes(requester=self,
                                              projection='polar',
                                              position=self.position,
                                              column=self.column,
                                              row=self.row,
                                              **kw)
        elif self.plot == '2d␣Polar␣Patch␣Type␣2':
            self.projection = '2dpolar'
            self.axes = self.tab.request_axes(requester=self,
                                              projection='polar',
                                              position=self.position,
                                              column=self.column,
                                              row=self.row,
                                              **kw)
        elif self.plot == '3d␣Polar␣Surface':
            self.projection = '3dpolar'
            self.axes = self.tab.request_axes(requester=self,
                                              projection='3d',
                                              position=self.position,
                                              column=self.column,
                                              row=self.row,
                                              **kw)
            self.properties = {
                'x␣label': 'x',
                'y␣label': 'y',
                'z␣label': 'z',
            }
        elif self.plot == '3d␣Polar':
            self.projection = '3dpolar'
            self.axes = self.tab.request_axes(requester=self,
                                              projection='3d',
                                              position=self.position,
                                              column=self.column,
                                              row=self.row,
                                              **kw)
            self.properties = {
                'x␣label': 'x',
                'y␣label': 'y',
                'z␣label': 'z',
            }

    def update_axes(self):
        # pass
        # if self.axes is not None:
        # self.axes.autoscale(enable=True, tight=True)
        # for k, v in zip(self.properties.keys(),
        # self.properties.values()):
        # if k == 'x label':
        # self.axes.set_xlabel(v)
        # elif k == 'y label':
        # self.axes.set_ylabel(v)
        # elif k == 'z label':
        # self.axes.set_zlabel(v)
        # elif k == 'axis':
        # self.axes.axis(v)
```

```
        # self.axes.set_visible(self.visible_flag)
        # self.axes.grid(self.grid_flag)
        # self.axes.get_xaxis().set_visible(self.xaxis_flag)
        # self.axes.get_yaxis().set_visible(self.yaxis_flag)
        # if not self.ticks_flag:
        # self.axes.xaxis.set_ticklabels([])
        # self.axes.yaxis.set_ticklabels([])
        if self.view_camera is not None:
            self.axes.view_init(elev=self.view_camera[0],azim=self.view_camera[1],roll=self.
                ↪ view_camera[2])
        self.axes.axis(self.axis_flag)
        if self.xticks is not None:
            self.axes.set_xticks(self.xticks)
        if self.yticks is not None:
            self.axes.set_yticks(self.yticks)
        if self.zticks is not None:
            self.axes.set_zticks(self.zticks)
        if self.xtick_labels is not None:
            self.axes.set_xticklabels(self.xtick_labels)
        if self.ytick_labels is not None:
            self.axes.set_yticklabels(self.ytick_labels)
        if self.ztick_labels is not None:
            self.axes.set_zticklabels(self.ztick_labels)
        if self.xlabel is not None:
            self.axes.set_xlabel(self.xlabel)
        if self.ylabel is not None:
            self.axes.set_ylabel(self.ylabel)
        if self.zlabel is not None:
            self.axes.set_zlabel(self.zlabel)
        if self.projection == '2dpolar':
            self.axes.grid(False)
            self.axis = []
            r = np.array([0, np.max(self.rticks)])
            for angle in [0,
                  np.pi/4,
                  np.pi/2,
                  3*np.pi/4,
                  np.pi,
                  -3*np.pi/4,
                  -np.pi/2,
                  -np.pi/4]:
                self.axis.append(self.axes.plot(np.array([angle, angle]), r, 'w', linewidth=0.3))
            a = np.linspace(0, 2*np.pi, 360)
            if self.rticks is not None:
                for radius in self.rticks:
                    self.axis.append(self.axes.plot(a, radius*np.ones_like(a), 'w', linewidth=0.3)
                        ↪ )
                self.axes.set_rticks(self.rticks)
                self.axes.set_yticklabels([str(i) + degree_sign for i in self.rticks])

            # self.axes.set_yticklabels(['$22.5^\circ$', '$45^\circ$', '$66.5^\circ$', '$90^\
                ↪ circ$'])
            # self.axes.set_ylabel(r"$\theta$")
            # self.axes.set_xlabel(r"$\phi$")

    def reference_polarization(self, theta, phi):
        sp = np.sin(phi)
```

```
        cp = np.cos(phi)
        st = np.sin(theta)
        ct = np.cos(theta)
        hat_i_ref_x = - (1 - ct)*sp*cp
        hat_i_ref_y = (1 - sp*sp*(1 - ct))
        hat_i_ref_z = - st*sp
        hat_i_ref = np.array([hat_i_ref_x,
                              hat_i_ref_y,
                              hat_i_ref_z]).swapaxes(
            0, 1).swapaxes(1, 2)

        hat_i_cross_x = (1 - cp*cp*(1-ct))
        hat_i_cross_y = - (1-ct)*sp*cp
        hat_i_cross_z = - st*cp
        hat_i_cross = np.array([hat_i_cross_x,
                                hat_i_cross_y,
                                hat_i_cross_z]
                               ).swapaxes(
                                  0, 1).swapaxes(1, 2)

        return hat_i_ref, hat_i_cross

    def polarization_matrix(self, theta, phi):
        sp = np.sin(phi)
        cp = np.cos(phi)
        st = np.sin(theta)
        ct = np.cos(theta)

        a_11 = 1 - st*st*cp*cp
        a_12 = - st*st*sp*cp
        a_13 = - st*ct*cp
        a_21 = - st*st*sp*cp
        a_22 = 1 - st*st*sp*sp
        a_23 = - st*ct*sp
        a_31 = - st*ct*cp
        a_32 = - st*ct*sp
        a_33 = 1 - ct*ct
        matrix = np.array([[a_11, a_12, a_13],
                           [a_21, a_22, a_23],
                           [a_31, a_32, a_33]])
        return matrix.swapaxes(0, 2).swapaxes(1, 3)

    def get_field(self, antenna=None, field=None, interp=False):
        if antenna is None:
            antenna = self.antenna
        if field is None:
            field = self.field

        if field == 'F':
            field = antenna.F
        elif field == 'Ftheta':
            field = antenna.Ftheta
        elif field == 'Fphi':
            field = antenna.Fphi
        elif field == 'Frhcp':
            field = antenna.Frhcp
        elif field == 'Flhcp':
```

```
            field = antenna.Flhcp
        else:
            F = np.array([antenna.Fx, antenna.Fy, antenna.Fz]
                         ).swapaxes(0, 1).swapaxes(1, 2)
            polarization_matrix = self.polarization_matrix(
                antenna.mesh_theta, antenna.mesh_phi)
            E = np.squeeze(polarization_matrix@F[:, :, :, np.newaxis])

            hat_i_ref, hat_i_cross = self.reference_polarization(
                antenna.mesh_theta, antenna.mesh_phi)

            if field == 'Fref':
                field = np.multiply(E, hat_i_ref).sum(2)
            elif field == 'Fcross':
                field = np.multiply(E, hat_i_cross).sum(2)
            elif field == 'Fref-Fcross':
                Fref = np.abs(np.multiply(E, hat_i_ref).sum(2))
                Fcross = np.abs(np.multiply(E, hat_i_cross).sum(2))
                field = np.sqrt(Fref*Fref + Fcross*Fcross)
            elif field == 'Ftheta-Fphi':
                Ftheta = np.abs(antenna.Ftheta)
                Fphi = np.abs(antenna.Fphi)
                field = np.sqrt(Ftheta*Ftheta + Fphi*Fphi)

        # field_phase = np.degrees(np.angle(field))
        field_magnitude = np.absolute(field)
        field_phase = np.angle(field)
        if self.in_dB:
            field_magnitude = 20*np.log10(field_magnitude)
            # self.vmin = self.colorbar_dB_min
            # self.vmax = self.colorbar_dB_max
        # else:
        # self.vmin = self.colorbar_min
        # self.vmax = self.colorbar_max
        # field_mag[field_mag <
        # self.dynamic_scaling_dB] = self.dynamic_scaling_dB
        # self.colorbar_label = '[dB]'

        if self.color_by == 'Color␣by␣magnitude':
            # color = field_mag
            if self.in_dB:
                self.colorbar_label = '[dB]'
            else:
                self.colorbar_label = ''
        elif self.color_by == 'Color␣by␣phase':
            # color = field_phase
            self.colorbar_label = '[deg]'
        # color = self.analysis.evaluate_color(antenna)
        # if str(type(color))=="<class 'NoneType'>":
        # color = field

        if self.color_by=='Color␣by␣phase':
            self.vmin = -180
            self.vmax = 180
        elif self.color_by=='Color␣by␣magnitude':
            if self.in_dB:
                self.vmin = self.colorbar_dB_min
```

```
            self.vmax = self.colorbar_dB_max
        else:
            self.vmin = self.colorbar_min
            self.vmax = self.colorbar_max

    if interp:
        interp_theta_deg = np.linspace(self.theta_i, self.theta_f, self.Ntheta)
        interp_phi_deg = np.linspace(self.phi_i, self.phi_f, self.Nphi)

        self.interp_mesh_phi_deg, self.interp_mesh_theta_deg = np.meshgrid(
            interp_phi_deg, interp_theta_deg)

        field_magnitude = self.antenna.interpolate_at(
            self.interp_mesh_theta_deg,
            self.interp_mesh_phi_deg,
            field_magnitude)

        field_phase = self.antenna.interpolate_at(
            self.interp_mesh_theta_deg,
            self.interp_mesh_phi_deg,
            field_phase)

        # normalized_field_magnitude = self.antenna.interpolate_at(
        # self.interp_mesh_theta_deg,
        # self.interp_mesh_phi_deg,
        # normalized_field_magnitude)

        # normalized_field_phase = self.antenna.interpolate_at(
        # self.interp_mesh_theta_deg,
        # self.interp_mesh_phi_deg,
        # normalized_field_phase)

    field_magnitude_max = field_magnitude.max()
    field_magnitude_min = field_magnitude.min()
    if field_magnitude_max != field_magnitude_min:
        normalized_field_magnitude = (field_magnitude-field_magnitude_min)/(
            ↪ field_magnitude_max-field_magnitude_min)
    else:
        normalized_field_magnitude = np.zeros_like(field_magnitude)

    field_magnitude_max = field_phase.max()
    field_magnitude_min = field_phase.min()
    if field_magnitude_max != field_magnitude_min:
        normalized_field_phase = (field_phase-field_magnitude_min)/(field_magnitude_max-
            ↪ field_magnitude_min)
    else:
        normalized_field_phase = np.zeros_like(field_phase)

    return field_magnitude, field_phase, normalized_field_magnitude, normalized_field_phase

def set_antenna(self, antenna):
    # self.undraw()
    if self.antenna is not None:
        self.antenna.listeners.remove(self)
    self.antenna = antenna
    if self.antenna is not None:
        self.antenna.listeners.append(self)
```

```
        self.ok = False

    # def set_analysis(self, analysis):
    # # self.undraw()
    # if self.analysis is not None:
    # self.analysis.listeners.append(self)
    # self.analysis = analysis
    # if self.analysis is not None:
    # self.analysis.listeners.append(self)
    # self.ok = False

    def set_field(self, field):
        self.field = field
        self.ok = False

    def set_color(self, color):
        self.color_by = color
        self.ok = False

    def set_plot(self, plot):
        self.plot = plot
        if self.plot == '2d Graph':
            self.projection = '2dcartesian'
        elif self.plot == '2d Contour':
            self.projection = '2dcartesian'
        elif self.plot == '3d Surface':
            self.projection = '3dcartesian'
        elif self.plot == '2d Polar Graph':
            self.projection = '2dpolar'
        elif self.plot == '2d Polar Contour':
            self.projection = '2dpolar'
        elif self.plot == '2d Polar Patch':
            self.projection = '2dpolar'
        elif self.plot == '2d Polar Patch Type 2':
            self.projection = '2dpolar'
        elif self.plot == '3d Polar Surface':
            self.projection = '3dpolar'
        elif self.plot == '3d Polar':
            self.projection = '3dpolar'
        elif self.plot == 'custom':
            self.projection = self.custom_projection
        self.ok = False

    def set_position(self, position=None, row=None, column=None):
        self.position = position
        self.row = row
        self.column = column
        self.ok = False

    def update(self):
        if self.ok:
            return

        # self.draw()

        self.tab.request_repaint()
        self.mark_update('Draw')
```

```
        self.ok = True

    def draw(self):
        # self.undraw()

        if self.plot == 'custom':
            self.custom_draw(self, self.tab)
            return

        if self.antenna is None and self.plot != 'custom':
            return
        if not self.antenna.ok:
            return

        self.get_axes()

        if self.axes is not None:
            if self.plot == '2d Graph':
                self.draw_graph()
            elif self.plot == '2d Contour':
                self.draw_contourf()
            elif self.plot == '3d Surface':
                self.draw_surface()
            elif self.plot == '2d Polar Graph':
                self.draw_polar_graph()
            elif self.plot == '2d Polar Contour':
                self.draw_polar_contourf()
            elif self.plot == '2d Polar Patch' or self.plot == \
                '2d Polar Patch Type 1':
                self.draw_polar_patch()
            elif self.plot == '2d Polar Patch Type 2':
                self.draw_polar_patch_type_2()
            elif self.plot == '3d Polar':
                self.draw_polar3d()
            elif self.plot == '3d Polar Surface':
                self.draw_polar_surface()

        self.update_axes()
        # print("Updating result " + str(self) + " " + self.plot)

    def get_2d_field(self, points):
        field, color = self.get_field()

        thetas = np.zeros((181))
        phis = np.zeros((181))
        for i in range(points.shape[0]):
            point = self.reference_2d.R@points[i, :]
            x = point[0]
            y = point[1]
            z = point[2]
            thetas[i] = np.arctan2(np.sqrt(y*y+x*x), z)
            phis[i] = np.arctan2(y, x)

        thetas = np.degrees(thetas)
        phis = np.degrees(phis)
        fit_points = np.ndarray((len(thetas), 2))
```

```
        fit_points[:, 0] = thetas
        fit_points[:, 1] = phis

        interp_thetas = np.degrees(self.antenna.mesh_theta).flatten()
        interp_phis = np.degrees(self.antenna.mesh_phi).flatten()
        interp_points = np.ndarray((len(interp_thetas), 2))
        interp_points[:, 0] = interp_thetas
        interp_points[:, 1] = interp_phis

        values = field.flatten()
        if values.dtype == np.dtype('complex64'):
            values = np.array(values, dtype=np.dtype('complex128'))
        field = griddata(interp_points,
                         values,
                         fit_points,
                         method='linear')

        return field

    def draw_graph(self):
        angles = np.radians(np.linspace(-180, 180, 181))
        points = np.zeros((len(angles), 3))
        points[:, 0] = np.cos(angles)
        points[:, 1] = np.sin(angles)

        field = self.get_2d_field(points)

        self.graphical_objects = self.axes.plot(angles, field)

    def draw_polar_graph(self):
        angles = np.radians(np.linspace(-180, 180, 181))
        points = np.zeros((len(angles), 3))
        points[:, 0] = np.cos(angles)
        points[:, 1] = np.sin(angles)

        field = self.get_2d_field(points)

        self.graphical_objects = self.axes.plot(angles, field)

    def draw_polar_contourf(self):
        field_magnitude, field_phase, normalized_field_magnitude, normalized_field_phase = self.
            ↪ get_field()

        self.graphical_objects = self.axes.contourf(
            self.antenna.mesh_phi,
            np.degrees(self.antenna.mesh_theta),
            field_magnitude, 10, cmap='jet')

    def draw_contourf(self):
        field_magnitude, field_phase, normalized_field_magnitude, normalized_field_phase = self.
            ↪ get_field()

        x = self.antenna.mesh_phi
        y = self.antenna.mesh_theta

        self.graphical_objects = self.axes.contourf(
            x, y, field_magnitude, 10, cmap='jet')
```

```
def draw_polar3d(self):
    field_magnitude, field_phase, normalized_field_magnitude, normalized_field_phase = self.
        ↪ get_field()

    position = np.array([self.antenna.x,
                         self.antenna.y,
                         self.antenna.z])
    if self.in_dB:
        min_field = np.min(field_magnitude)
        R = (field_magnitude[:, :, np.newaxis]-min_field)*self.antenna.hat_k
    else:
        R = field_magnitude[:, :, np.newaxis]*self.antenna.hat_k

    field_max = field_magnitude.max()
    field_min = field_magnitude.min()
    if field_max != field_min:
        normalized_field_magnitude = (field_magnitude-field_min)/(field_max-field_min)
    else:
        normalized_field_magnitude = np.zeros_like(field_magnitude)

    if self.color_by=='Color␣by␣magnitude':
        jet = plt.colormaps['jet']
        # if self.in_dB:
        # if color_max != color_min:
        # C = (field_magnitude-color_min)/(color_max-color_min)
        # rgb = jet(C)
        # else:
        # rgb = list(field_magnitude.shape)
        # rgb.append(4)
        # rgb = np.zeros(tuple(rgb))
        # rgb[:, :, 3] = 1
        # rgb[:, :, 2] = 1
        # else:
        rgb = jet(normalized_field_magnitude)
    elif self.color_by=='Color␣by␣phase':
        jet = plt.colormaps['jet']
        rgb = jet(normalized_field_phase)
    if self.translate:
        R += position
    self.graphical_objects = self.axes.plot_surface(
        R[:, :, 0], R[:, :, 1], R[:, :, 2],
        rstride=1, cstride=1, facecolors=rgb,
        linewidth=0, antialiased=False)
    self.properties['axis'] = 'equal'

def draw_surface(self):
    field_magnitude, field_phase, normalized_field_magnitude, normalized_field_phase = self.
        ↪ get_field(interp=True)

    # interp_x = np.linspace(self.theta_i, self.theta_f, self.Ntheta)
    # interp_y = np.linspace(self.phi_i, self.phi_f, self.Nphi)

    # interp_mesh_x, interp_mesh_y = np.meshgrid(interp_x, interp_y)

    # field_magnitude = self.antenna.interpolate_at(
    # interp_mesh_x, interp_mesh_y, field_magnitude)
```

```
        if self.color_by==’Color␣by␣magnitude’:
            cmap = plt.colormaps[’jet’]
            color = field_magnitude
            # rgb = cmap(normalized_field_magnitude)
            # if self.in_dB:
            # color_max = field_magnitude.max()
            # color_min = field_magnitude.min()
            # if color_max != color_min:
            # C = (field_magnitude-color_min)/(color_max-color_min)
            # rgb = jet(C)
            # else:
            # rgb = list(field_magnitude.shape)
            # rgb.append(4)
            # rgb = np.zeros(tuple(rgb))
            # rgb[:, :, 3] = 1
            # rgb[:, :, 2] = 1
            # else:
        elif self.color_by==’Color␣by␣phase’:
            cmap = plt.colormaps[’twilight’]
            # field_phase = self.antenna.interpolate_at(
            # self.interp_mesh_theta_deg, self.interp_mesh_phi_deg, field_phase)
            # rgb = cmap(normalized_field_phase)
            color = 180*field_phase/np.pi

        norm=mpl.colors.Normalize(vmin=self.vmin,vmax=self.vmax)
        m = cm.ScalarMappable(cmap=cmap, norm=norm)
        self.graphical_objects = self.axes.plot_surface(
            self.interp_mesh_theta_deg,
            self.interp_mesh_phi_deg,
            field_magnitude,
            rstride=1, cstride=1, facecolors=cmap(norm(color)),
            linewidth=0, antialiased=self.antialiased)
        plt.colorbar(m,
                     ax=self.axes,
                     label=self.colorbar_label, extend=’both’,
                     # anchor=(0.0, 0.7),
                     pad=0.1)

    def draw_polar_surface(self):
        field_magnitude, field_phase, normalized_field_magnitude, normalized_field_phase = self.
            ↪ get_field(interp=True)

        if self.color_by==’Color␣by␣magnitude’:
            cmap = plt.colormaps[’jet’]
            # rgb = cmap(normalized_field_magnitude)
            color = field_magnitude
        elif self.color_by==’Color␣by␣phase’:
            cmap = plt.colormaps[’twilight’]
            # rgb = cmap(normalized_field_phase)
            color = 180*field_phase/np.pi
        # self.graphical_objects = self.axes.pcolormesh(
        # np.radians(interp_mesh_phi_deg),
        # 90*np.sin(np.radians(self.interp_mesh_theta_deg)),
        # color,
        # rstride=1, cstride=1, facecolors=rgb,
        # linewidth=0, antialiased=False,
```

```
        # # norm=norm,
        # shading='gouraud',
        # cmap=cmap,
        # vmin=self.vmin,
        # vmax=self.vmax)

        # field = self.antenna.interpolate_at(
        # self.interp_mesh_theta_deg, interp_mesh_phi_deg, field)

        # jet = plt.colormaps['jet']
        # color_max = color.max()
        # color_min = color.min()
        # if color_max != color_min:
        # C = (color-color_min)/(color_max-color_min)
        # rgb = jet(C)
        # else:
        # rgb = list(color.shape)
        # rgb.append(4)
        # rgb = np.zeros(tuple(rgb))
        # rgb[:, :, 3] = 1
        # rgb[:, :, 2] = 1

        norm=mpl.colors.Normalize(vmin=self.vmin,vmax=self.vmax)
        m = cm.ScalarMappable(cmap=cmap, norm=norm)
        self.graphical_objects = self.axes.plot_surface(
            np.cos(np.radians(self.interp_mesh_phi_deg))* \
                self.interp_mesh_theta_deg,
            np.sin(np.radians(self.interp_mesh_phi_deg))* \
                self.interp_mesh_theta_deg,
            field_magnitude,
            rstride=1, cstride=1, facecolors=cmap(norm(color)),
            linewidth=0, antialiased=self.antialiased)
        plt.colorbar(m,
                     ax=self.axes,
                     label=self.colorbar_label, extend='both',
                     # anchor=(0.0, 0.7),
                     pad=0.1)

    def draw_polar_patch(self):
        field_magnitude, field_phase, normalized_field_magnitude, normalized_field_phase = self.
            ↪ get_field(interp=True)

        # interp_theta_deg = np.linspace(self.theta_i, self.theta_f, self.Ntheta)
        # interp_phi_deg = np.linspace(self.phi_i, self.phi_f, self.Nphi)

        # interp_mesh_phi_deg, interp_mesh_theta_deg = np.meshgrid(
        # interp_phi_deg, interp_theta_deg)

        if self.color_by == 'Color␣by␣magnitude':
            # field_magnitude = self.antenna.interpolate_at(
            # self.interp_mesh_theta_deg,
            # interp_mesh_phi_deg,
            # field_magnitude)
            cmap = 'jet'
            color = field_magnitude
            # self.graphical_objects = self.axes.pcolormesh(
            # np.radians(self.interp_mesh_phi_deg),
```

```
            # 90*np.sin(np.radians(self.interp_mesh_theta_deg)),
            # field_magnitude,
            # # norm=norm,
            # shading='gouraud',
            # cmap=cmap,
            # vmin=self.vmin,
            # vmax=self.vmax
            # )
            # plt.colorbar(self.graphical_objects, ax=self.axes,
            # label=self.colorbar_label, extend='both',
            # pad=0.1)
            # # boundaries=[self.colorbar_min, self.colorbar_max])
        elif self.color_by == 'Color by phase':
            # field_phase = self.antenna.interpolate_at(
            # self.interp_mesh_theta_deg,
            # self.interp_mesh_phi_deg,
            # field_phase)
            cmap = 'twilight'
            color = 180*field_phase/np.pi
        self.graphical_objects = self.axes.pcolormesh(
            np.radians(self.interp_mesh_phi_deg),
            90*np.sin(np.radians(self.interp_mesh_theta_deg)),
            color,
            # norm=norm,
            shading='gouraud',
            cmap=cmap,
            vmin=self.vmin,
            vmax=self.vmax)
        plt.colorbar(self.graphical_objects, ax=self.axes,
                     label=self.colorbar_label, extend='both',
                     # anchor=(0.0, 0.7),
                     pad=0.1)
                      # boundaries=[self.colorbar_min, self.colorbar_max])

        # color_max = color.max()
        # color_min = color.min()
        # if color_max != color_min:
        # C = (color-color_min)/(color_max-color_min)
        # rgb = plt.colormaps[cmap](C)
        # else:
        # rgb = list(color.shape)
        # rgb.append(4)
        # rgb = np.zeros(tuple(rgb))
        # rgb[:, :, 3] = 1
        # rgb[:, :, 2] = 1

        # norm = mpl.colors.BoundaryNorm(boundaries=[self.colorbar_min, self.colorbar_max],
            ↪ ncolors=256)
        self.axes.set_title(self.title)
        self.rticks = [22.5, 45, 66.5, 90]

    def draw_polar_patch_type_2(self):
        field_magnitude, field_phase, normalized_field_magnitude, normalized_field_phase = self.
            ↪ get_field()

        if self.color_by == 'Color by magnitude':
            cmap = 'jet'
```

```
        color = field_magnitude
        # self.graphical_objects = self.axes.pcolormesh(
        # self.antenna.mesh_phi,
        # 180*self.antenna.mesh_theta/np.pi,
        # field_magnitude,
        # shading='gouraud',
        # cmap=cmap,
        # vmin=self.vmin,
        # vmax=self.vmax)
    elif self.color_by == 'Color␣by␣phase':
        cmap = 'twilight'
        color = 180*field_phase/np.pi
    self.graphical_objects = self.axes.pcolormesh(
        self.antenna.mesh_phi,
        180*self.antenna.mesh_theta/np.pi,
        color,
        shading='gouraud',
        cmap=cmap,
        vmin=self.vmin,
        vmax=self.vmax)
    plt.colorbar(self.graphical_objects, ax=self.axes,
                 label=self.colorbar_label, pad=0.1)

    # color_max = color.max()
    # color_min = color.min()
    # if color_max != color_min:
    # C = (color-color_min)/(color_max-color_min)
    # rgb = plt.colormaps[cmap](C)
    # else:
    # rgb = list(color.shape)
    # rgb.append(4)
    # rgb = np.zeros(tuple(rgb))
    # rgb[:, :, 3] = 1
    # rgb[:, :, 2] = 1

    self.axes.set_title(self.title)
    self.rticks = [45, \
                   90, \
                   135, \
                   180]
    self.rtick_labels = ['45' + degree_sign, \
                         '90' + degree_sign, \
                         '135' + degree_sign, \
                         '180' + degree_sign]

def result_menu(self, tw):
    new_menu = tk.Frame(master=tw)
    tk.Label(master=new_menu, text='Results', justify='left',
             relief='solid', borderwidth=0).pack(
                 ipadx=1, fill=tk.BOTH)
    tk.Button(master=new_menu, text='edit',
              command=self.on_result_ppp).pack(
                  ipadx=1, fill=tk.BOTH)
    tk.Button(master=new_menu, text='update',
              command=self.on_update_result).pack(
                  ipadx=1, fill=tk.BOTH)
    tk.Button(master=new_menu, text='delete',
```

```
                command=self.on_delete_obj).pack(
                    ipadx=1, fill=tk.BOTH)
        new_menu.pack(ipadx=1)
```

Apêndice 9.98 – ResultEditorFrame.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 14 17:34:44 2023

@author: 160047412
"""

import tkinter as tk
from tkinter import ttk

# from AxesEditorFrame import AxesEditorFrame
import Geometry


class ResultEditorFrame(ttk.Frame):
    def __init__(self,
                 app,
                 result,
                 on_finish=True,
                 on_done=None,
                 on_cancel=None,
                 master=None,
                 **kw):
        ttk.Frame.__init__(self, master=master, width=300, height=200, **kw)
        self.app = app
        self.result = result
        self.on_finish = on_finish
        self.on_done = on_done
        self.on_cancel = on_cancel

        self.current_plot_frame = None

        self.init_variables()
        self.init_layout()

        self._on_plot_change()
        # self._on_builtin_field_change()

        # self.select_axes_lstbx.bind('<<ListboxSelect>>',
        # self.on_axes_selection)

    def init_variables(self):
        self.name = tk.StringVar(value=self.result.name)
        self.projection_variable = tk.StringVar(value=self.result.projection)
        self.plot_variable = tk.StringVar(value=self.result.plot)
        self.field_variable = tk.StringVar(value=self.result.field)
        self.color_by_variable = tk.StringVar(value=self.result.color_by)
        self.color_by_textvariable = tk.StringVar(value=self.result.color_by)
        self.row_variable = tk.IntVar(value=self.result.row)
        self.column_variable = tk.IntVar(value=self.result.column)

        self.X_X_variable = tk.DoubleVar(
            value=self.result.reference_2d.x_axis.x)
        self.X_Y_variable = tk.DoubleVar(
            value=self.result.reference_2d.x_axis.y)
```

```
        self.X_Z_variable = tk.DoubleVar(
            value=self.result.reference_2d.x_axis.z)
        self.Z_X_variable = tk.DoubleVar(
            value=self.result.reference_2d.z_axis.x)
        self.Z_Y_variable = tk.DoubleVar(
            value=self.result.reference_2d.z_axis.y)
        self.Z_Z_variable = tk.DoubleVar(
            value=self.result.reference_2d.z_axis.z)

        self.in_dB_variable = tk.IntVar(value=self.result.in_dB)
        self.colorbar_dB_min_variable = tk.DoubleVar(
            value=self.result.colorbar_dB_min)
        self.visible_flag_variable = tk.IntVar(value=self.result.visible_flag)
        self.axis_flag_variable = tk.IntVar(value=self.result.axis_flag)
        self.grid_flag_variable = tk.IntVar(value=self.result.grid_flag)
        self.antialiased_variable = tk.IntVar(value=self.result.antialiased)
        self.add_colorbar_variable = tk.IntVar(value=self.result.add_colorbar)
        self.colorbar_dB_min_variable = tk.DoubleVar(value=self.result.colorbar_dB_min)
        self.colorbar_dB_max_variable = tk.DoubleVar(value=self.result.colorbar_dB_max)
        self.colorbar_min_variable = tk.DoubleVar(value=self.result.colorbar_min)
        self.colorbar_max_variable = tk.DoubleVar(value=self.result.colorbar_max)

        # self.edit_axes_frame_label = tk.StringVar()

    def init_layout(self):
        fr_left = ttk.Frame(master=self)
        fr_left.pack(side='left', fill='both', padx=7, pady=7)

        fr_left_top = ttk.Frame(master=fr_left)
        fr_left_top.pack(side='top', fill='both', padx=3, pady=3)

        fr_left_top_left = ttk.LabelFrame(master=fr_left_top, text='Tab␣name:')
        fr_left_top_left.pack(side='left', fill='both', padx=3, pady=3)

        ttk.Entry(master=fr_left_top_left, textvariable=self.name).pack(
            side='left', fill='both')

        ttk.Button(master=fr_left_top, text='update',
                   command=self._on_update).pack(side='left', fill='both')

        fr_left_top = ttk.LabelFrame(master=fr_left, text='Axes␣position:')
        fr_left_top.pack(side='top', fill='both', padx=3, pady=3)

        fr_left_top_left = ttk.LabelFrame(master=fr_left_top, text='Row:')
        fr_left_top_left.pack(side='left', fill='both', padx=3, pady=3)
        ttk.Entry(master=fr_left_top_left, textvariable=self.row_variable).pack(
            side='left', fill='both')
        fr_left_top_left = ttk.LabelFrame(master=fr_left_top, text='Column:')
        fr_left_top_left.pack(side='left', fill='both', padx=3, pady=3)
        ttk.Entry(master=fr_left_top_left, textvariable=self.column_variable).pack(
            side='left', fill='both')

        # ttk.Button(master=fr_left_top, text='update',
        # command=self._on_update).pack(side='left', fill='both')

        # fr_left_top_left_left = ttk.LabelFrame(master=fr_left_top_left, text='Row:')
        # fr_left_top_left.pack(side='left', fill='both', padx=3, pady=3)
```

```
        # ttk.Entry(master=fr_left_top_left_left, textvariable=self.row_variable).pack(
        # side='left', fill='both')
        # fr_left_top_left_left = ttk.LabelFrame(master=fr_left_top_left, text='Column:')
        # fr_left_top_left.pack(side='left', fill='both', padx=3, pady=3)
        # ttk.Entry(master=fr_left_top_left_left, textvariable=self.column_variable).pack(
        # side='left', fill='both')

        fr_left_top = ttk.Frame(master=fr_left)
        fr_left_top.pack(side='top', fill='both', padx=3, pady=3)

        fr_left_top_left = ttk.LabelFrame(master=fr_left_top, text='Antennas')
        fr_left_top_left.pack(side='left', fill='both', padx=3, pady=3)

        self.antenna_cbbx = ttk.Combobox(
            master=fr_left_top_left,
            state='readonly',
            values=[
                antenna.name
                for antenna
                in self.app.antennas])
        self.antenna_cbbx.pack(side='left', fill='both')
        if len(self.app.antennas) > 0:
            if self.result.antenna is not None:
                self.antenna_cbbx.current(
                    self.app.antennas.index(self.result.antenna))
            else:
                self.antenna_cbbx.current(0)

        fr_left_top = ttk.Frame(master=fr_left)
        fr_left_top.pack(side='top', fill='both', padx=3, pady=3)

        fr_left_top_left = ttk.LabelFrame(master=fr_left_top, text='Plot')
        fr_left_top_left.pack(side='left', fill='both', padx=3, pady=3)

        cbbx = ttk.Combobox(
            master=fr_left_top_left,
            state='readonly',
            values=self.result.available_plots,
            textvariable=self.plot_variable,
            validate='all',
            validatecommand=self._on_plot_change)
        cbbx.pack(side='left', fill='both')
        cbbx.bind('<<ComboboxSelected>>', self._on_plot_change)

        fr_left_top_left = ttk.LabelFrame(master=fr_left_top, text='Field')
        fr_left_top_left.pack(side='left', fill='both', padx=3, pady=3)

        field_cbbx = ttk.Combobox(
            master=fr_left_top_left,
            state='readonly',
            textvariable=self.field_variable,
            values=self.result.available_fields)
        field_cbbx.pack(side='left', fill='both')

        fr_left_top = ttk.LabelFrame(master=fr_left, text='Options')
        fr_left_top.pack(side='top', fill='both')
```

```
        fr_left_top_left = ttk.Frame(master=fr_left_top)
        fr_left_top_left.pack(side='left', fill='both', padx=3, pady=3)

        ttk.Checkbutton(
            master=fr_left_top_left, text='visible',
            variable=self.visible_flag_variable).pack(side='top', fill='both')
        ttk.Checkbutton(
            master=fr_left_top_left, text='axis',
            variable=self.axis_flag_variable).pack(side='top', fill='both')
        ttk.Checkbutton(
            master=fr_left_top_left, text='grid',
            variable=self.grid_flag_variable).pack(side='top', fill='both')
        ttk.Checkbutton(
            master=fr_left_top_left, text='in␣dB',
            variable=self.in_dB_variable).pack(side='top', fill='both')
        ttk.Checkbutton(
            master=fr_left_top_left, text='antialiased',
            variable=self.antialiased_variable).pack(side='top', fill='both')

        fr_left_top_left = ttk.Frame(master=fr_left_top)
        fr_left_top_left.pack(side='left', fill='both', padx=3, pady=3)

        ttk.Checkbutton(
            master=fr_left_top_left, textvariable=self.color_by_textvariable,
            variable=self.color_by_variable,
            command=self._on_color_cbt_change,
            onvalue='Color␣by␣magnitude',
            offvalue='Color␣by␣phase').pack(side='left', fill='both')

        fr_left_top_left = ttk.Frame(master=fr_left_top)
        fr_left_top_left.pack(side='top', fill='both', padx=3, pady=3)

        fr_left_top_left_left = ttk.LabelFrame(
            master=fr_left_top_left, text='Dynamic␣Scaling')
        fr_left_top_left_left.pack(side='left')

        fr_left_top_left_left_top = ttk.Frame(master=fr_left_top_left_left)
        ttk.Label(master=fr_left_top_left_left_top, text="min␣dB").pack(side='left')
        ttk.Entry(
            master=fr_left_top_left_left_top,
            textvariable=self.colorbar_dB_min_variable).pack(
            side='left', fill='both')
        fr_left_top_left_left_top.pack(side='top', fill='both', padx=3, pady=3)

        fr_left_top_left_left_top = ttk.Frame(master=fr_left_top_left_left)
        ttk.Label(master=fr_left_top_left_left_top, text="max␣dB").pack(side='left')
        ttk.Entry(
            master=fr_left_top_left_left_top,
            textvariable=self.colorbar_dB_max_variable).pack(
            side='left', fill='both')
        fr_left_top_left_left_top.pack(side='top', fill='both', padx=3, pady=3)

        fr_left_top_left_left_top = ttk.Frame(master=fr_left_top_left_left)
        ttk.Label(master=fr_left_top_left_left_top, text="min").pack(side='left')
        ttk.Entry(
            master=fr_left_top_left_left_top,
            textvariable=self.colorbar_min_variable).pack(
```

```
    side='left', fill='both')
fr_left_top_left_left_top.pack(side='top', fill='both', padx=3, pady=3)

fr_left_top_left_left_top = ttk.Frame(master=fr_left_top_left_left)
ttk.Label(master=fr_left_top_left_left_top, text="max").pack(side='left')
ttk.Entry(
    master=fr_left_top_left_left_top,
    textvariable=self.colorbar_max_variable).pack(
    side='left', fill='both')
fr_left_top_left_left_top.pack(side='top', fill='both', padx=3, pady=3)

self.plot_frame = ttk.LabelFrame(master=self, text='Plot␣options')

self.plot_2d_frame = ttk.LabelFrame(
    master=self.plot_frame, text='2d␣plot')
self.plot_3d_frame = ttk.LabelFrame(
    master=self.plot_frame, text='3d␣plot')

fr_left_top_left = ttk.LabelFrame(
    master=self.plot_2d_frame, text='X␣Axis')
fr_left_top_left.pack(side='top', fill='both')
fr_left_top_left_left = ttk.LabelFrame(
    master=fr_left_top_left, text='X')
fr_left_top_left_left.pack(side='left', fill='both')
ttk.Entry(master=fr_left_top_left_left,
         textvariable=self.X_X_variable).pack(
            side='left', fill='both')
fr_left_top_left_left = ttk.LabelFrame(
    master=fr_left_top_left, text='Y')
fr_left_top_left_left.pack(side='left', fill='both')
ttk.Entry(master=fr_left_top_left_left,
         textvariable=self.X_Y_variable).pack(
            side='left', fill='both')
fr_left_top_left_left = ttk.LabelFrame(
    master=fr_left_top_left, text='Z')
fr_left_top_left_left.pack(side='left', fill='both')
ttk.Entry(master=fr_left_top_left_left,
         textvariable=self.X_Z_variable).pack(
            side='left', fill='both')

fr_left_top_left = ttk.LabelFrame(
    master=self.plot_2d_frame, text='Z␣Plane')
fr_left_top_left.pack(side='top', fill='both')
fr_left_top_left_left = ttk.LabelFrame(
    master=fr_left_top_left, text='X')
fr_left_top_left_left.pack(side='left', fill='both')
ttk.Entry(master=fr_left_top_left_left,
         textvariable=self.Z_X_variable).pack(
            side='left', fill='both')
fr_left_top_left_left = ttk.LabelFrame(
    master=fr_left_top_left, text='Y')
fr_left_top_left_left.pack(side='left', fill='both')
ttk.Entry(master=fr_left_top_left_left,
         textvariable=self.Z_Y_variable).pack(
            side='left', fill='both')
fr_left_top_left_left = ttk.LabelFrame(
    master=fr_left_top_left, text='Z')
```

```
        fr_left_top_left_left.pack(side='left', fill='both')
        ttk.Entry(master=fr_left_top_left_left,
                  textvariable=self.Z_Z_variable).pack(
                      side='left', fill='both')

        if self.on_finish:
            fr_left_top = ttk.LabelFrame(master=fr_left, text='Finish')
            fr_left_top.pack(side='top', fill='both')
            ttk.Button(master=fr_left_top, text='Done',
                       command=self._on_done).pack(
                           side=tk.LEFT, fill=tk.BOTH)
            ttk.Button(master=fr_left_top, text='Cancel',
                       command=self.on_cancel).pack(side=tk.LEFT, fill=tk.BOTH)

    def _on_update(self):
        # self.result.ok = False
        # self.result.update()
        # if True:
        # return
        self.result.name = self.name.get()
        self.result.set_antenna(self.app.antennas[self.antenna_cbbx.current()])
        self.result.set_field(self.field_variable.get())
        self.result.set_plot(self.plot_variable.get())
        self.result.set_color(self.color_by_textvariable.get())
        self.result.set_position(row=self.row_variable.get(), column=self.column_variable.get())

        x_axis = Geometry.Axis()
        z_axis = Geometry.Axis()

        x_axis.x = self.X_X_variable.get()
        x_axis.y = self.X_Y_variable.get()
        x_axis.z = self.X_Z_variable.get()

        z_axis.x = self.Z_X_variable.get()
        z_axis.y = self.Z_Y_variable.get()
        z_axis.z = self.Z_Z_variable.get()

        self.result.reference_2d = Geometry.ReferenceSystem(
            x_axis=x_axis, z_axis=z_axis)

        # print(self.visible_flag_variable.get())
        self.result.in_dB = self.in_dB_variable.get() == 1
        self.result.colorbar_dB_min = self.colorbar_dB_min_variable.get()
        self.result.visible_flag = self.visible_flag_variable.get() == 1
        self.result.axis_flag = self.axis_flag_variable.get() == 1
        self.result.grid_flag = self.grid_flag_variable.get() == 1
        self.result.add_colorbar = self.add_colorbar_variable.get() == 1
        self.result.colorbar_min = self.colorbar_min_variable.get()
        self.result.colorbar_max = self.colorbar_max_variable.get()

        self.result.ok = False
        self.result.update()

    def _on_done(self):
        self._on_update()
        self.on_done()
```

```
    def _on_color_cbt_change(self):
        self.color_by_textvariable.set(self.color_by_variable.get())

    def _on_plot_change(self, event=None):
        if self.current_plot_frame is not None:
            self.current_plot_frame.pack_forget()
            self.plot_frame.pack_forget()

        if self.plot_variable.get() in [
                '2d␣Graph',
                '2d␣Polar␣Graph',
                '2d␣Contour',
                '2d␣Polar␣Contour',
                '2d␣Polar␣Patch']:
            self.current_plot_frame = self.plot_2d_frame
            self.plot_2d_frame.pack(side='top', fill='both')
            self.plot_frame.pack(side='top', fill='both')
        elif self.plot_variable.get() == [
                '3d␣Surface',
                '3d␣Polar␣Surface',
                '3d␣Polar']:
            self.current_plot_frame = self.plot_3d_frame
            self.plot_3d_frame.pack(side='top', fill='both')
            self.plot_frame.pack(side='top', fill='both')
```

Apêndice 9.99 – ResultFigure.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 13 18:10:10 2023

@author: 160047412
"""

import matplotlib.pyplot as plt


class ResultFigure:
    def __init__(self, columns=1, rows=1):
        self.columns = columns
        self.rows = rows

        self.listeners = []
        self.ok = True

        bg_color = 'white'
        # bg_color = '#505050'
        self.figure = plt.figure(figsize=(5, 4), dpi=100,
                                 facecolor=bg_color)
        self.axes_dict = dict()
        self.results = []

    def notify(self, caller, event):
        self.ok = False
        self.mark_update('"' + str(caller) +
                         '" called notify with event "' + event + '"')

    def mark_update(self, event):
        for l in self.listeners:
            print(str(self) +
                  ' notifying ' + str(l) + ' for ' + event)
            l.notify(self)

    def add_result(self, result):
        self.results.append(result)
        result.listeners.append(self)
        self.ok = False

    def remove_result(self, result):
        self.results.remove(result)
        result.listeners.remove(self)
        self.ok = False

    def request_axes(self, requester, projection,
                     position=None, column=None, row=None, **kw):
        if position is None:
            position = (row - 1)*self.columns + column
        self.axes_dict[requester] = self.figure.add_subplot(
            self.rows, self.columns, position, projection=projection)
        return self.axes_dict[requester]

    def request_axes_delete(self, requester):
```

```
        if requester in self.axes_dict.get_keys():
            self.axes_dict[requester].remove()
            self.axes_dict.pop(requester)

    def request_repaint(self):
        self.canvas_flag = True

    def update(self):
        if self.ok:
            return

        self.draw()

        self.ok = True
        self.mark_update('Draw')

    def draw(self):
        self.undraw()

        self.draw_background()
        for result in self.results:
            result.draw()

    def undraw(self):
        self.figure.clear()

    def draw_background(self):
        pass
        # self.background_axes = self.figure.axes(rect=(0,0,1,1))
        # self.background_axes.set_facecolor('#505050')
```

Apêndice 9.100 – ResultFrame.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 13 18:19:19 2023

@author: 160047412
"""

import tkinter as tk

import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import (
    FigureCanvasTkAgg, NavigationToolbar2Tk)
from matplotlib.backend_bases import key_press_handler


class ResultFrame(tk.Frame):
    def __init__(self, master=None, name='New tab',
                 columns=1, rows=1, **kw):
        tk.Frame.__init__(self,
                          master,
                          width=300, height=200,
                          **kw)
        self.name = name
        self.columns = columns
        self.rows = rows
        self.subplot_spaces = [None]

        self.listeners = []
        self.ok = True

        bg_color = 'white'
        # bg_color = '#505050'
        self.figure = plt.Figure(figsize=(5, 4),
                                 dpi=100,
                                 facecolor=bg_color)
        self.axes_dict = dict()
        self.results = []

        # A tk.DrawingArea.
        self.canvas = FigureCanvasTkAgg(self.figure, master=self)
        self.canvas.draw()

        # pack_toolbar=False will make it easier to use a
        # layout manager later on.
        self.toolbar = NavigationToolbar2Tk(self.canvas,
                                            self,
                                            pack_toolbar=False)
        self.toolbar.update()

        self.canvas.mpl_connect(
            "key_press_event",
            lambda event: print(f"you pressed {event.key}"))
        self.canvas.mpl_connect("key_press_event",
                                key_press_handler)
        self.toolbar.pack(side=tk.BOTTOM, fill=tk.X)
```

```
    self.canvas.get_tk_widget().pack(side=tk.TOP,
                                     fill=tk.BOTH,
                                     expand=True)

def notify(self, caller, event):
    self.ok = False
    self.mark_update('"' + str(caller) +
                     '"␣called␣notify␣with␣event␣"' +
                     event + '"')

def mark_update(self, event):
    for l in self.listeners:
        print(str(self) + '␣notifying␣' + str(l) + '␣for␣' +
              event)
        l.notify(self)

def add_result(self, result):
    self.results.append(result)
    result.listeners.append(self)
    self.ok = False
    # if len(self.results)>len(self.subplot_spaces):
    # self.determine_subplots_layout()

def remove_result(self, result):
    self.results.remove(result)
    result.listeners.remove(self)
    self.ok = False

def request_axes(self, requester, projection,
                 position=None, column=None, row=None, **kw):
    if position is None:
        position = (row - 1)*self.columns + column
    self.axes_dict[requester] = \
        self.figure.add_subplot(
            self.rows, self.columns,
            position, projection=projection, **kw)
    # print(position)
    # print(self.axes_dict[requester])
    return self.axes_dict[requester]

def request_axes_delete(self, requester):
    if requester in self.axes_dict.get_keys():
        self.axes_dict[requester].remove()
        self.axes_dict.pop(requester)

def request_repaint(self):
    self.ok = False

def update(self):
    if self.ok:
        return

    self.draw()

    self.ok = True
    self.mark_update('Draw')
```

```
    def draw(self):
        self.undraw()
        # print("Drawing " + str(self))

        # self.draw_background()
        for result in self.results:
            result.draw()

        self.canvas.draw()
        # self.configure(width=300*self.iy, height=200*self.ix)

    def undraw(self):
        self.figure.clear()

    def draw_background(self):
        pass
        # self.background_axes = self.figure.axes(rect=(0,0,1,1))
        # self.background_axes.set_facecolor('#505050')
```

Apêndice 9.101 – script_antenas.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 29 18:31:27 2023

@author: vitinho
"""
import os
import sys
import header

python_path = header.python_path
results_dir = os.path.join(header.results_dir, 'Circular␣Polarization')
home_dir = header.home_dir
antennas_dir = header.antennas_dir

import numpy as np

import matplotlib.pyplot as plt
plt.close('all')

import AntennasLoaders.LoadHFSSYagis
# import Scripts.AntennasLoaders.LoadHFSSValidationArrays
# import Scripts.AntennasLoaders.LoadValidationArrays

Ntheta = 91
Nphi = 91

antennas = AntennasLoaders.LoadHFSSYagis.run(
    Ntheta=Ntheta, Nphi=Nphi, elevation=-90)

import Array

Ntheta=91
Nphi=181

theta=np.linspace(0, 180, Ntheta)
phi=np.linspace(-180, 180, Nphi)

array = Array.Array(name='Custom',
                    theta=theta.copy(),
                    phi=phi.copy(),
                    antennas=[
                        antennas['hfss_yagi2EL'].copy(),
                        antennas['hfss_yagi2EL'].copy(),
                        ])
array.antennas[0].set_position(x=0,y=0,z=0)
array.antennas[0].set_orientation(
    elevation=-90,azimuth=0,roll=0)
array.antennas[0].set_current(
    magnitude=1.0,
    phase=0)
array.antennas[1].set_position(x=0,y=0,z=0)
array.antennas[1].set_orientation(
    elevation=-90,azimuth=0,roll=90)
```

```
array.antennas[1].set_current(
    magnitude=1.0,
    phase=90)
array.evaluate()

import Result
import ResultFigure

plot = '2d Polar Patch'

field = 'F'
figure = ResultFigure.ResultFigure(
    columns=1,rows=1)
result = Result.Result(
    tab=figure,
    title='Custom Array',
    name='Custom Array',
    antenna=array,
    field=field,
    plot=plot,
    ticks_flag=False,
    in_dB=True,
    column=1,row=1,
    Ntheta=Ntheta,
    Nphi=Nphi
    )
figure.draw()
fname = os.path.join(
    results_dir, array.name +
        ' F' + '.png')
figure.figure.savefig(fname)
plt.close('all')
```

Apêndice 9.102 – ValidationHFSS.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 13 14:49:56 2023

@author: 160047412
"""

import sys
import os
path = os.path.split(__file__)[0]
path = os.path.split(path)[0]
sys.path.insert(0, path)
path = os.path.split(path)[0]
home_directory = os.path.split(path)[0]
antennas_dir=os.path.join(home_directory, 'Antennas')

import Scripts.LoadDefaultGraphs
import Scripts.LoadCompareAntennas

def run(app, antennas):

    # Scripts.LoadDefaultGraphs.load_default_graphs(
    # app=app,
    # name='H',
    # antenna=antennas['array_H'])
    # Scripts.LoadDefaultGraphs.load_default_graphs(
    # app=app,
    # name='V',
    # antenna=antennas['array_V'])
    # Scripts.LoadDefaultGraphs.load_default_graphs(
    # app=app,
    # name='RHCP',
    # antenna=antennas['array_RHCP'])

    Scripts.LoadDefaultGraphs.load_default_graphs(
        app=app,
        name='Validation␣1Y-4EL',
        antenna=antennas['array_validation_1Y4EL'])
    Scripts.LoadDefaultGraphs.load_default_graphs(
        app=app,
        name='Validation␣2Y-4EL',
        antenna=antennas['array_validation_2Y4EL'],
        fields=['F', 'Fref', 'Fcross'])
    Scripts.LoadDefaultGraphs.load_default_graphs(
        app=app,
        name='Validation␣3Y-4EL',
        antenna=antennas['array_validation_3Y4EL'],
        fields=['F', 'Fref', 'Fcross'])
    Scripts.LoadDefaultGraphs.load_default_graphs(
        app=app,
        name='Validation␣4Y-4EL',
        antenna=antennas['array_validation_4Y4EL'])
    Scripts.LoadDefaultGraphs.load_default_graphs(
        app=app,
        name='Validation␣5Y-4EL',
```

```
    antenna=antennas['array_validation_5Y4EL'])

Scripts.LoadDefaultGraphs.load_default_graphs(
    app=app,
    name='HFSS 1Y-4EL',
    antenna=antennas['HFSS_1Y4EL'])
Scripts.LoadDefaultGraphs.load_default_graphs(
    app=app,
    name='HFSS 2Y-4EL',
    antenna=antennas['HFSS_2Y4EL'],
    fields=['F', 'Fref', 'Fcross'])
Scripts.LoadDefaultGraphs.load_default_graphs(
    app=app,
    name='HFSS 3Y-4EL',
    antenna=antennas['HFSS_3Y4EL'],
    fields=['F', 'Fref', 'Fcross'])
Scripts.LoadDefaultGraphs.load_default_graphs(
    app=app,
    name='HFSS 4Y-4EL',
    antenna=antennas['HFSS_4Y4EL'])
Scripts.LoadDefaultGraphs.load_default_graphs(
    app=app,
    name='HFSS 5Y-4EL',
    antenna=antennas['HFSS_5Y4EL'])

Scripts.LoadCompareAntennas.load_compare_graphs(
    app=app,
    name='1Y-4EL',
    antennas=[antennas['HFSS_1Y4EL'],
              antennas['array_validation_1Y4EL']],
    titles=['HFSS', 'Validation'])
Scripts.LoadCompareAntennas.load_compare_graphs(
    app=app,
    name='2Y-4EL',
    antennas=[antennas['HFSS_2Y4EL'],
              antennas['array_validation_2Y4EL']],
    titles=['HFSS', 'Validation'])
Scripts.LoadCompareAntennas.load_compare_graphs(
    app=app,
    name='3Y-4EL',
    antennas=[antennas['HFSS_3Y4EL'],
              antennas['array_validation_3Y4EL']],
    titles=['HFSS', 'Validation'])
Scripts.LoadCompareAntennas.load_compare_graphs(
    app=app,
    name='4Y-4EL',
    antennas=[antennas['HFSS_4Y4EL'],
              antennas['array_validation_4Y4EL']],
    titles=['HFSS', 'Validation'])
Scripts.LoadCompareAntennas.load_compare_graphs(
    app=app,
    name='5Y-4EL',
    antennas=[antennas['HFSS_5Y4EL'],
              antennas['array_validation_5Y4EL']],
    titles=['HFSS', 'Validation'])
```