UNIVERSIDADE FEDERAL DO PARANÁ

MARINOEL JOAQUIM

MÉTODO DAS DIFERENÇAS FINITAS NO DOMÍNIO DO TEMPO APLICADO À GRADE TRIDIMENSIONAL FORMADA POR PRISMAS HEXAGONAIS

CURITIBA

2016

MARINOEL JOAQUIM

MÉTODO DAS DIFERENÇAS FINITAS NO DOMÍNIO DO TEMPO APLICADO À GRADE TRIDIMENSIONAL FORMADA POR PRISMAS HEXAGONAIS

Tese apresentada ao Programa de Pós-Graduação em Métodos Numéricos em Engenharia, Universidade Federal do Paraná, como requisito parcial à obtenção do título de Doutor em Métodos Numéricos em Engenharia.

Orientador: Prof. Dr. Sergio Scheer

CURITIBA

2016

Joaquim, Marinoel
Método das diferenças finitas no domínio do tempo aplicado à grade tridimensional formada por prismas hexagonais / Marinoel Joaquim. – Curitiba, 2016.
521 f. : il.

Tese (doutorado) – Universidade Federal do Paraná, Setor de Tecnologia, Programa de Pós-Graduação em Métodos Numéricos em Engenharia.
Orientador: Sérgio Scheer
Bibliografia: p.211-215

1. Anisotropia. 2. Prismas. 3. Sistemas de comunicação sem fio. I. Scheer, Sérgio. II. Título.

CDD 671.823

Ministério da Educação
Universidade Federal do Paraná
Setor de Tecnologia / Setor de Ciências Exatas
Departamento de Construção Civil / Departamento de Matemática/ Departamento de Engenharia de Produção.
Programa de Pós-Graduação em Métodos Numéricos em Engenharia – PPGMNE/UFPR.

PPGMNE

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em MÉTODOS NUMÉRICOS EM ENGENHARIA da Universidade Federal do Paraná foram convocados para realizar a arguição da Tese de Doutorado de **MARINOEL JOAQUIM**, intitulada: *"Método das Diferenças Finitas no Domínio do Tempo Aplicado à Grade Tridimensional Formada por Prismas Hexagonais"*, após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua APROVAÇÃO

Curitiba, 02 de Junho de 2016.

Prof SÉRGIO SCHEER
Presidente da Banca Examinadora (UFPR)

Prof JOSÉ ANTONIO MARQUES CARRER
Avaliador Interno (UFPR)

Prof KLAUS DE GEUS
Avaliador Interno (UFPR)

Prof WILSON ARNALDO ARTUZI JUNIOR
Avaliador Externo (UFPR)

Prof ISMAEL CHIAMENTI
Avaliador Externo (UTFPR)

Dedico este trabalho a Deus pela sua providência.

À minha esposa Marli por seu apoio e amor.

Aos meus filhos Lucas e Ana que têm sido motivo de alegria na minha vida.

AGRADECIMENTOS

A Deus pela sua graça e bênçãos que se renovam a cada manhã.

Ao professor Sergio Scheer pela confiança depositada em mim, ajuda e orientação na realização deste trabalho de pesquisa.

À banca examinadora composta pelos professores Klaus de Geus, José Antônio Marques Carrer, Wilson Arnaldo Artuzi Junior e Ismael Chiamenti; suas sugestões e correções aperfeiçoaram significativamente o texto final desta tese.

Ao corpo docente do Programa de Pós-Graduação em Métodos Numéricos em Engenharia da UFPR pelo aprimoramento profissional e pessoal que me propiciou.

Ao secretário Jair B. dos Anjos Silva do PPGMNE pela ajuda e profissionalismo.

À direção da UTFPR por me liberar em tempo integral para a realização desta pós-graduação.

Aos meus pais José e Teresa pelo amor e apoio constante; e especialmente a minha mãe que incultiu nas minhas duas irmãs e em mim o amor aos estudos.

Aos meus tios Valmir (*in memoriam*) e Emília pela amizade, as muitas conversas e incentivo para continuar progredindo na vida.

RESUMO

O método das diferenças finitas no domínio do tempo, mais conhecido em inglês como *Finite-Difference Time-Domain* (FDTD), foi aplicado em uma grade tridimensional de prismas hexagonais, tendo como objetivo produzir menos anisotropia de velocidade de fase numérica que o método FDTD Yee (com células hexaédricas). Comparações de propagação de onda são feitas entre o método FDTD com grade de prismas hexagonais e o método FDTD Yee, com o objetivo de validar esse novo método FDTD. As análises teóricas da anisotropia, dispersão e condição de estabilidade numéricas são realizadas usando a análise de Fourier no método FDTD com grade de prismas hexagonais e comparadas com aquelas realizadas para o método FDTD Yee. Medidas de anisotropia numérica neste novo método FDTD são comparadas com os resultados da análise de Fourier. Uma formulação para camadas de absorção casadas perfeitamente, que emulam um espaço infinito, é desenvolvida para a grade de prismas hexagonais, e medidas de reflexão dessas camadas de absorção são comparadas com aquelas das camadas de absorção da grade Yee. Uma onda plana é aplicada na grade de prismas hexagonais usando a formulação de regiões de campos total e espalhado; e medidas de espalhamento de campo de placa retangular e esfera (ambas usando condutor elétrico perfeito) são realizadas na grade de prismas hexagonais, e comparadas com as medidas de espalhamento, para os mesmos objetos, na grade Yee e soluções analíticas. Um algoritmo de compensação de dispersão numérica é desenvolvido para o método FDTD com grade de prismas hexagonais, com a condição de que as dimensões da grade no plano x-y são muito maiores que a dimensão z (altura). Medidas de dispersão numérica na grade de prismas hexagonais e comparações com aquelas da grade Yee demonstram a eficácia do algoritmo proposto, que permite o uso de grade de prismas hexagonais com menor densidade de malha que a grade Yee, para uma certa precisão desejada. Os resultados obtidos são promissores para o uso do método FDTD com grade de prismas hexagonais em simulações de propagação de ondas eletromagnéticas irradiadas por redes *Wireless* em edificações, onde, normalmente, as dimensões no plano horizontal são muito maiores que a altura.

Palavras-chave: FDTD. Dispersão numérica. Anisotropia numérica. Prisma hexagonal. Célula hexagonal. Propagação de onda. Redes sem fio.

ABSTRACT

The finite-difference time-domain (FDTD) method was applied at a grid of hexagonal prisms, with the objective to yield less numerical anisotropy of phase velocity than the Yee FDTD method (with hexahedral cells). Comparisons of wave propagation are made between the FDTD method with grid of hexagonal prisms and the Yee FDTD method, with the aim of checking the validity of this new FDTD method. The theoretical analyses of the numerical anisotropy, dispersion and stability condition are obtained using the Fourier analysis in the FDTD method with grid of hexagonal prisms, and they are compared with those accomplished for Yee FDTD method. Measurements of numerical anisotropy are accomplished in this new FDTD method, and then they are compared with the results of the Fourier analysis. A formulation for perfectly matched layer absorbing boundary conditions, which emulates an infinite space, is developed for a grid of hexagonal prisms, and reflection measurements of these perfectly matched layers (PMLs) are compared with those of the Yee grid PMLs. A plane wave is applied in the grid of hexagonal prisms using the total-field/scattered-field formulation, and scattered-field measurements of rectangular plate and sphere (both using perfect electric conductor) are made in the grid of hexagonal prisms, and the next ones are compared with the scattered-field measurements, for the same objects, in the Yee grid and theoretical solutions. An algorithm of numerical dispersion compensation is developed for the FDTD method with grid of hexagonal prisms, with the condition that the grid dimensions in the x-y plane are considerably larger than the z-dimension (height). Measurements of numerical dispersion in the grid of hexagonal prisms and comparisons with those of the Yee grid, it proves the effectiveness of the proposed algorithm, which allows using grid of hexagonal prisms with less density of mesh than the Yee grid, for a certain desirable accuracy. The obtained results are promising to use the FDTD method with grid of hexagonal prisms in simulations of electromagnetic wave propagation yielded for wireless networks in indoor buildings, where the dimensions in the horizontal plane are usually much larger than the height.

Keywords: FDTD. Numerical dispersion. Numerical anisotropy. Hexagonal prisms. Hexagonal cells. Wave propagation, Wireless networks.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

1D	-	Unidimensional
2D	-	Bidimensional
3D	-	Tridimensional
1G	-	1ª geração de telefonia celular
2G	-	2ª geração de telefonia celular
2.5G	-	Transição entre a 2ª e 3ª gerações de telefonia celular
3G	-	3ª geração de telefonia celular
4G	-	4ª geração de telefonia celular
ABCs	-	Analytical Absorbing Boundary Conditions
ADI-FDTD	-	Alternating-Direction-Implicit Finite-Difference Time-Domain
AM	-	Amplitude Modulada
CPML	-	Convolutional Perfectly Matched Layer
CPUs	-	Central Processing Units
DGTD	-	Discontinuous Galerkin Time-Domain
FCC-FDTD	-	Face-Centered-Cubic Finite-Difference Time-Domain
FDTD	-	Finite-Difference Time-Domain
FEM	-	Finite-Element Method
FETD	-	Finite-Element Time-Domain
FM	-	Frequência Modulada
FVTD	-	Finite-Volume Time-Domain
GPUs	-	Graphics Processing Units
HIE-FDTD	-	Hybrid Implicit-Explicit Finite-Difference Time-Domain
IEEE	-	Institute of Electrical and Electronics Engineers
IRLA	-	Intelligent Ray Launching
LOD-FDTD	-	Locally One-Dimensional Finite-Difference Time-Domain
MRTD	-	Multiresolution Time-Domain
PML	-	Perfectly Matched Layer
RCS	-	Radar Cross Section
RT	-	Ray Tracing
SFDTD	-	Symplectic Finite-Difference Time-Domain
SS-FDTD	-	Split-Step Finite-Difference Time-Domain

TE - Transversal Elétrico

TF/SF - Total-Field / Scattered-Field

TM - Transversal Magnético

WiMAX - Worldwide Interoperability for Microwave Acess

WLANs - Wireless Local Area Networks

LISTA DE SÍMBOLOS

A - Ampere

a - raio da esfera; largura na direção y da placa retangular

A_H - área do hexágono menor

A_R - área da face retangular do prisma com hexágono menor

$\boldsymbol{a}_{n1}$ - vetor unitário na direção n_1

$\boldsymbol{a}_{n2}$ - vetor unitário na direção n_2

$\boldsymbol{a}_{n3}$ - vetor unitário na direção n_3

A_{s0} - amplitude inicial da função senoidal analítica

b - largura na direção z da placa retangular

B - densidade de fluxo magnético

$\boldsymbol{B}$ - vetor densidade de fluxo magnético

C - coulomb

c - velocidade de fase física (real); largura na direção x da placa retangular

C - contorno fechado

c^* - velocidade de fase numérica aproximada

D - densidade de fluxo elétrico

$\boldsymbol{D}$ - vetor densidade de fluxo elétrico

dBsm - decibel square meter

E - intensidade de campo elétrico

$\boldsymbol{E}$ - vetor intensidade de campo elétrico

f - frequência

F - faraday

$\boldsymbol{F}$ - vetor campo

f_p - frequência de pico

F_D - densidade de malha da grade Yee em relação a grade de prismas hexagonais

GB giga bytes

H - henry

H - intensidade de campo magnético

$\boldsymbol{H}$ - vetor Intensidade de campo magnético

Hz - hertz

i - índice na direção x

$\boldsymbol{i}$ - vetor unitário na direção x

j - índice na direção y; unidade imaginária

J - unidade imaginária

$\boldsymbol{j}$ - vetor unitário na direção y

$\boldsymbol{J}$ - vetor densidade de corrente elétrica

k - índice na direção z; módulo do vetor número de onda

$\boldsymbol{k}$ - vetor unitário na direção z; vetor número de onda

L_X - comprimento (em metros) na direção x

L_Y - comprimento (em metros) na direção y

L_Z - comprimento (em metros) na direção z

m - metro

m - ordem do polinômio

ma - ordem do polinômio

$\boldsymbol{M_i}$ - vetor densidade de corrente magnética

n - número de passos de tempo

N_X - número de pontos na direção x

N_Y - número de pontos na direção y

N_Z - número de pontos na direção z

N_λ - número de pontos por comprimento de onda

O - ordem do erro nas equações de diferença finita

r - distância; raio

$\boldsymbol{r}$ - vetor distância

R - razão; coeficiente de reflexão

$\boldsymbol{r_i}$ - vetor distância de incidência

$\boldsymbol{r_r}$ - vetor distância de reflexão

S - siemens

S - superfície aberta ou fechada

S_{CFL} - número de Courant

S_w - condutividade normalizada na direção w

t - tempo

T - tesla

T - período da senoide; coeficiente de transmissão

$u(t)$ - função passo no tempo

V - volt

V - volume

v_g - velocidade de grupo

v_{med} - velocidade de fase média numérica

v_{pn} - velocidade de fase numérica exata

Wb - weber

Z - impedância característica do meio

α_T - fator de atenuação

β_0 - fase ou ângulo inicial

X_e - susceptibilidade elétrica

$\delta(t)$ - função impulso no tempo

Δ - incremento de espaço

Δb - tamanho do lado do hexágono menor

Δd - tamanho do lado do hexágono maior

Δt - incremento de tempo

Δx - incremento de espaço na direção x

Δy - incremento de espaço na direção y

Δz - incremento de espaço na direção z

ε - permissividade elétrica

ε_0 - permissividade elétrica no vácuo

ε_r - permissividade elétrica relativa

Φ - ângulo

φ - função escalar

η - impedância característica do meio

λ - comprimento de onda

μ - permeabilidade magnética

μ_0 - permeabilidade magnética no vácuo

μ_r - permeabilidade magnética relativa

θ - ângulo; ângulo de espalhamento

θ_i - ângulo de incidência

θ_r - ângulo de reflexão

θ_s - ângulo de espalhamento

θ_t - ângulo de transmissão ou refração

ρ_{mv} - densidade volumétrica de cargas magnéticas

ρ_v - densidade volumétrica de cargas elétricas

ς - constante de tempo na função senoidal numérica

ς_0 - constante de tempo na função senoidal analítica

σ - condutividade elétrica

σ_m - condutividade magnética

σ_{3D} - seção transversal de radar (para espaço tridimensional)

ω - frequência angular

Ω - ohm

ξ - ângulo

∇ - operador diferencial

∇_s - operador diferencial usando coordenadas estiradas

SUMÁRIO

1 INTRODUÇÃO

As comunicações sem fio (“*wireless*”) têm sido usadas desde 1899 quando Guglielmo Marconi inventou um sistema prático com transmissores de “faísca” e antenas para transmissão de ondas de rádio ou ondas hertzianas como ficaram conhecidas na época, em homenagem ao físico Heinrich Rudolf Hertz que em 1887 comprovou experimentalmente a existência destas ondas eletromagnéticas, semelhantes à luz, que foram postuladas teoricamente por James Clerk Maxwell em 1864. Este sistema, conhecido como telegrafia sem fio, foi utilizado com sucesso para comunicação com navios. Com a invenção da válvula tríodo (o primeiro dispositivo amplificador eletrônico inventado), no início do século XX, foi aberto o caminho para o rápido desenvolvimento da radiodifusão de som (AM e FM) e de imagens (TV analógica), o uso de rádio fixo e móvel pelas forças armadas, órgãos governamentais, radioamadores e cidadãos comuns. Com a invenção do transistor, circuitos integrados, microprocessadores e avanços na microeletrônica, os dispositivos de radiocomunicação se tornaram ao mesmo tempo muito sofisticados e de tamanho reduzido.

No entanto, uma revolução extraordinária nas comunicações sem fio só ocorreu quando foram implantados os primeiros sistemas de telefonia celular na década de 1980. Utilizando o conceito de reutilização de frequência em diferentes células separadas por uma distância suficiente para se evitar a interferência de co-canal (RAPPAPORT, 2009), esses sistemas foram capazes de fornecer serviço de telefonia celular a baixo custo para a população em geral. A segunda geração (2G) de telefonia celular (com tecnologia digital ao invés da 1G que era analógica) foi introduzida no início dos anos 1990 com aumento da capacidade de dados trafegados e maior eficiência, mas ainda projetada quase que especificamente para comunicação de voz. Com o grande crescimento mundial da internet e proliferação de redes de computadores a partir, também, da década de 1990, surgiram no início do século XXI as gerações 2.5G e 3G, onde a geração 2.5G faz uma transição tecnológica gradativa e viável economicamente para a 3G. Nestas novas gerações de telefonia celular, principalmente a 3G, o objetivo principal é fornecer de forma flexível maior largura de banda aos usuários, para suportar outros tipos de informação, além da voz, como textos, e-mails, *Web*, imagens, vídeos, e outros serviços, como por exemplo, conexão a redes locais sem fio, *Bluetooth* e redes pessoais. Recentemente entraram em

operação os primeiros sistemas da 4G, tais como o WiMAX (*Worldwide Inter-operability for Microwave Acess*).

Outra tecnologia "*wireless*" que tem se tornado muito importante são as redes locais sem fio [*Wireless Local Area Networks* (WLANs)] que permite que usuários móveis usando *notebooks*, *laptops*, telefones celulares e outros dispositivos portáteis possam se conectar à internet a partir de uma rede local fixa interligada, normalmente por meio com fio (fios metálicos ou fibras ópticas), a um provedor de serviços de telecomunicações. As WLANs têm como vantagem trabalhar com espectro de frequência não-licenciado que torna seu uso mais flexível, já que seu alcance é normalmente limitado a um prédio ou ao campus de uma universidade. Isto não ocorre no caso da telefonia celular, em que o espectro é licenciado às operadoras de uma região ou cidade pelos governos desses locais, que exercem uma fiscalização e controle bem mais rígidos, para se evitarem abusos e interferências eletromagnéticas indesejáveis entre sistemas.

1.1 MODELOS DE RADIOPROPAGAÇÃO

As WLANs seguem padrões propostos internacionalmente, como os padrões IEEE 802.11. Os canais alocados por estes padrões estão em torno das frequências de 2,45 GHz e 5,86 GHz. Muitos usuários de WLANs no mundo têm ainda uma experiência não satisfatória com estes padrões, devido, por exemplo, às taxas de transferência de dados inadequadas e insuficientes, taxas de erros de bits altas, quedas de conexão, etc. (NEVE *et al.*, 2010). A causa principal destes problemas está na radiopropagação dos canais de rádio no ambiente de uso. Diferentemente da tecnologia com fio (fios metálicos ou fibras ópticas), a propagação da onda eletromagnética do canal de rádio é mais complexa e aleatória, dependendo da estrutura geométrica e de materiais do ambiente físico real, das condições do tempo (umidade, temperatura, chuva, neve, etc.), da localização física das antenas, tipos de antenas, níveis de potência, espectro de frequência alocado e das técnicas utilizadas de modulação, codificação e processamento de sinal. Assim, historicamente, a modelagem do canal de rádio, principalmente o canal de rádio móvel, tem sido a parte mais difícil e empírica no projeto de um sistema de comunicações sem fio (RAPPAPORT, 2009).

Há atualmente uma demanda por serviços de comunicação sem fio de alta qualidade, banda larga, confiáveis, robustos e fáceis de implantar. É importante lembrar que os engenheiros e tecnólogos responsáveis pela implantação rotineira de WLANs têm pouco controle sobre a tecnologia em si, pois esta é definida por padrões internacionais (por exemplo, os padrões IEEE 802.11), que especificam os tipos de modulação, codificação, acesso múltiplo, etc. No entanto, há alguns fatores que podem ser controlados por quem implanta WLANs: ajuste dinâmico, embora limitado, de camada de protocolo (especificamente nos padrões IEEE 802.11); localização dos pontos de acesso (que depende de análise e modelagem de radiopropagação); tipos de antenas e suas orientações; seleção do canal de rádio; e ajustes no próprio ambiente físico (NEVE *et al.*, 2010).

A análise e modelagem de radiopropagação de canais de rádio têm sido estudadas por décadas. Modelos de radiopropagação dividem-se em três principais: empíricos, determinísticos e quase-determinísticos. Modelos empíricos podem ser razoavelmente precisos e eficientes (custo computacional baixo). A sua precisão é garantida unicamente no ambiente em que foram coletadas medidas para ajustes do modelo matemático. Aqui se entende custo computacional baixo como reduzido tempo de processamento e uso de limitada quantidade de memória (a rigor, isto depende da tecnologia disponível e de sua relação custo/benefício). Uma desvantagem do modelo empírico é que os resultados obtidos em um ambiente normalmente não podem ser transportados para outro ambiente. Modelos determinísticos são obtidos a partir das leis da física e, portanto, são muito precisos; mas essa precisão tem um custo computacional alto, principalmente quando não se utiliza processamento paralelo. Várias tentativas têm sido feitas para desenvolver modelos quase-determinísticos, que procuram fazer aproximações mais simples, mas retendo fundamentos das leis da física. Tais modelos são mais precisos que modelos empíricos e mais eficientes que modelos determinísticos e, portanto, normalmente mais práticos de serem utilizados no planejamento diário de sistemas de comunicação sem fio. É bom lembrar que embora modelos determinísticos possam fazer cálculos rigorosos e precisos baseados nas leis da física, frequentemente o ambiente físico poderá unicamente ser especificado com um grau limitado de precisão (NEVE *et al.*, 2010). Assim permanecerá o desafio de se instalarem sistemas de comunicação sem fio em um ambiente físico relativamente incerto e sujeito a constantes mudanças.

1.2 PESQUISAS COM MODELOS DE RADIOPROPAGAÇÃO DETERMINÍSTICOS

As atividades de pesquisa para radiopropagação em WLANs se dividem em três áreas (NEVE *et al*., 2010): modelos de radiopropagação em ambientes internos para implantação de redes locais sem fio; aperfeiçoamento de comunicação sem fio de uso interno via ajustes da camada de protocolo (padrões IEEE 802.11); e otimização de comunicação sem fio por modificação no ambiente físico (por exemplo, uso de superfícies seletivas em frequência).

O objetivo geral de pesquisa, neste texto, é a modelagem de radiopropagação em ambientes, principalmente internos, para implantação de WLANs. Como já visto, os modelos determinísticos são os mais precisos, porém exigem um maior custo computacional em termos de tempo de execução e memória. Dois modelos determinísticos bastante usados atualmente são o traçado de raios [*Ray Tracing* (RT)] e o método das diferenças finitas no domínio do tempo [*Finite-Difference Time-Domain* (FDTD)]. Ambos podem ser usados em simulações bidimensionais (2D) e tridimensionais (3D). O método RT é de simples implementação e tem um custo computacional menor que o FDTD.

A rigor os fenômenos físicos relevantes na propagação das ondas de rádio são: atenuação no espaço livre (espalhamento da energia eletromagnética nas direções de propagação), reflexão, refração, transmissão, absorção, difração e dispersão (por exemplo, pela folhagem de árvores ou superfícies rugosas). Além destes, existe a propagação por múltiplos caminhos (desvanecimento de pequena escala), que pode provocar interferência intersimbólica e aumentar a taxa de erros de bits (RAPPAPORT, 2009).

O método RT é baseado no comportamento óptico dos raios de luz. Deve-se lembrar que o comprimento de onda de uma onda eletromagnética que se propaga num meio é definido como a velocidade da onda eletromagnética neste meio dividido pela frequência desta onda. Assim considerando o meio como sendo o ar, com velocidade de propagação de aproximadamente $3\ 10^8$ m/s, a luz visível terá comprimentos de onda entre 0,4 e 0,8 μm, o que é muito menor que as dimensões de objetos encontrados em ambientes internos de edificações. Esta aproximação permite utilizar a teoria da óptica geométrica (ou teoria dos raios de luz), ao invés da mais rigorosa teoria eletromagnética, sem perda significativa de precisão. Na teoria óptica dos raios são apenas levadas em conta a reflexão e a refração.

No entanto, as frequências de rádio por não serem tão altas como as da luz visível apresentam comprimentos de ondas maiores; por exemplo, uma onda eletromagnética se propagando no ar à velocidade da luz, numa frequência de 3 GHz, terá um comprimento de onda de 10 cm. Na propagação em um ambiente de escritório haverá objetos de dimensões semelhantes à desse comprimento de onda, que não poderão ser analisados apenas pela teoria geométrica da óptica. Assim modelos mais precisos do método RT incluem efeitos de difração e de atenuação, como é o caso do modelo IRLA (*Intelligent Ray Launching*) para ambientes internos (LAI *et al.*, 2010).

O segundo método, o método determinístico (FDTD), é baseado nas equações de Maxwell e permite uma análise mais completa e precisa de ambientes físicos com dimensões elétricas moderadas. Ele é mais poderoso que o método RT, pois incorpora, implicitamente nas suas equações, todas as reflexões e difrações possíveis. Outra vantagem é a sua simplicidade matemática (sem uso de matrizes e operações matriciais), que torna simples codificá-lo em uma linguagem de programação. Ele é assim simples e robusto com uma base teórica bem estabelecida (TAFLOVE; HAGNESS, 2005), o que o torna muito popular. No entanto, simulações realistas usando o método FDTD envolvem uma divisão discreta fina dos domínios espacial e temporal. Não é incomum usar grades espaciais de mais de 10^9 células e grades espaços-temporais de mais de 10^{13} células (SHAMS; SADEGHI, 2010). Como resultado, as simulações com FDTD exigem grandes fontes de recursos computacionais em termos de memória e velocidade de processamento. Felizmente o método FDTD é inerentemente paralelo na sua formulação matemática, o que o torna bem adaptado ao processamento computacional paralelo. Assim, como o método RT, o método FDTD pode ser implementado em unidades de processamento gráfico [*Graphics Processing Units* (GPUs)] e também em "*clusters*" de GPUs ou de CPUs (*Central Processing Units*) como demonstrado em (SHAMS; SADEGHI, 2010).

Métodos determinísticos híbridos (THIEL; SARABANDI, 2009; ROCHE *et al.*, 2010) também têm sido pesquisados, principalmente os que utilizam o método RT ou suas variantes acoplados a métodos de análise de onda total, como por exemplo, o método FDTD. A vantagem desses métodos híbridos é que eles buscam reter o menor custo computacional do método RT, e manter uma precisão razoável com o método FDTD. Os recentes avanços em computação paralela de baixo custo têm tornado o uso de métodos numéricos de onda total, como o FDTD, viáveis para cenários de propagação interna.

1.3 OBJETIVOS ESPECÍFICOS DE PESQUISA

Como visto na seção anterior, o método FDTD parece ser bastante promissor para análise de radiopropagação em ambientes internos eletricamente grandes (edifícios empresariais, fábricas, universidades, aeroportos, etc.), na medida em que os recursos de memória, velocidade e capacidade de processamento paralelo dos computadores atuais tendem a aumentar e os custos a diminuir. Apesar disso, existe uma limitação intrínseca do método FDTD denominada dispersão numérica. A dispersão numérica pode ser definida como o erro entre a velocidade de propagação numérica da onda eletromagnética na grade e sua velocidade física (real). Este erro dependerá da direção em que a frente de onda se propaga na grade. Assim, uma outra medida muito importante será o erro relativo entre as direções de maior e menor velocidade, ou seja, a anisotropia de velocidade de fase numérica. Quando analisando estruturas eletricamente grandes (como, por exemplo, ambientes onde se usam WLANs) com o método FDTD é muito importante ter baixa dispersão numérica na onda eletromagnética, para que se possam minimizar os erros de fase e, consequentemente, aumentar a precisão da simulação. A dispersão numérica é mais facilmente minimizada quando a anisotropia numérica é também baixa. Esta anisotropia é uma característica intrínseca da grade, dependendo principalmente de forma geométrica; a minimização dessa anisotropia numérica tem sido objeto de muita pesquisa, havendo muitas formulações diferentes (LIU, 1996; SHLAGER; SCHNEIDER, 2003; TAFLOVE; HAGNESS, 2005; PANARETOS; ABERLE; DÍAZ, 2006; SMITH *et al.*, 2012; SESCU; HIXON, 2014).

Nesta tese, os objetivos específicos de pesquisa são os seguintes:

1) Desenvolver um novo método FDTD com grade tridimensional, que utiliza células com formato de prismas hexagonais a partir da formulação bidimensional de hexágonos (LIU, 1996), com o objetivo de reduzir a anisotropia de velocidade de fase numérica, principalmente no plano x-y.
2) A redução de anisotropia numérica em centenas de vezes no plano x-y da grade de prismas hexagonais permitirá desenvolver um algoritmo eficiente e simples de compensação de dispersão numérica para o método FDTD com grade de prismas hexagonais. Consequentemente, este novo método FDTD poderá simular a propagação de ondas eletromagnéticas em andares de edificações, onde normalmente as dimensões no plano horizontal (plano x-y)

são muito maiores que a altura, com uma malha menos densa que o método FDTD convencional (com células hexaédricas). Os resultados práticos serão o aumento de precisão nas simulações e a redução do uso de recursos computacionais, como memória e tempo de processamento.

3) Desenvolver a formulação de camadas de absorção para o método FDTD com grade de prismas hexagonais de forma a emular um espaço infinito.
4) Desenvolver a formulação com regiões de campos total e espalhado adaptada a grade de prismas hexagonais.

No capítulo 2 é feita uma breve revisão de análise vetorial, teoria eletromagnética e do método FDTD Yee.

No capítulo 3 é realizada a dedução da formulação matemática do método FDTD aplicado à grade formada por prismas hexagonais, e sua validação comparando-o com o método FDTD Yee, que utiliza células hexaédricas (ou cúbicas).

No capítulo 4 são analisadas a anisotropia e a dispersão de velocidade de fase numéricas e a condição de estabilidade da grade formada por prismas hexagonais, utilizando a análise de Fourier. Medidas de anisotropia são realizadas na grade de prismas hexagonais e comparadas com os resultados da análise de Fourier.

No capítulo 5 é desenvolvida uma formulação de camada de absorção casada perfeitamente para a grade de prismas hexagonais, que emula propagação num espaço infinito. Medidas de reflexão das camadas de absorção são realizadas nesta grade e comparadas com aquelas das camadas de absorção da grade Yee.

No capítulo 6 é aplicada uma fonte externa com onda plana na grade de prismas hexagonais a partir da formulação com regiões de campos total e espalhado. Comparações de espalhamentos teóricos e numéricos (para ambos os métodos FDTD) de placa retangular metálica e esfera metálica são realizadas.

No capítulo 7 é desenvolvida uma compensação de dispersão mais simples para a grade de prismas hexagonais, aplicável em edificações grandes, principalmente no plano x-y, que permitirá utilizar malhas menos densas que o método FDTD Yee.

No capítulo 8 são feitas algumas considerações importantes na implementação computacional do método FDTD com grade de prismas hexagonais.

No capítulo 9 são feitas as considerações finais e conclusões desta pesquisa.

2 REVISÃO DE LITERATURA

O método das diferenças finitas no domínio do tempo, mais conhecido em inglês como FDTD (*Finite-Difference Time-Domain*), tem se tornado na atualidade um dos mais importantes métodos numéricos aplicados em problemas eletromagnéticos. O método FDTD encontra aplicações por todo o espectro de frequências: prospecção geológica (com frequências muito baixas), radiodifusão de rádio e televisão, telefonia celular, redes *Wireless*, satélites, micro-ondas, óptica, fotônica, microeletrônica, astronomia, raios-x, raios gama (em medicina ou pesquisa científica), bioengenharia, aplicações na indústria, agricultura, meteorologia, forças armadas, etc. No entanto, o uso desse método tornou-se prático e viável apenas a partir dos anos 90, com o crescimento do poder de processamento dos computadores, tanto em termos de velocidade de processamento, como em capacidade de armazenamento de dados.

O método FDTD tem como vantagens (INAN; MARSHALL, 2011) em relação a outros métodos numéricos frequentemente usados em computação eletromagnética (como, por exemplo, o método dos elementos finitos ou o método dos momentos usado na solução de equações integrais ou integro-diferenciais em eletromagnetismo (JIN, 2010)):

- Tempo de desenvolvimento mais curto de programas computacionais.
- Fácil entendimento, pois ele gera equações de diferença finita diretamente a partir das equações de Maxwell na forma diferencial ou integral.
- O método FDTD Yee (tradicional) é de natureza explícita, ou seja, sem uso de álgebra linear ou inversão de matrizes e, portanto, com processamento mais simples e tendo o tempo de simulação como o principal limite.
- O método FDTD adapta-se bem com meios físicos complexos: não-homogêneos, anisotrópicos ou dispersivos.
- Sendo um método no domínio do tempo permite o uso de fontes com alto conteúdo espectral (pulsos no tempo), o que pode reduzir bastante o tempo de simulação, quando se deseja a solução do problema para uma faixa grande de frequências.

Tem, como todo método numérico, algumas desvantagens importantes:

- Bordas aproximadas por passos em forma de escada. O método FDTD Yee usa uma estrutura de grade ortogonal (retangular em simulações

bidimensionais, ou hexaedros em simulações tridimensionais); portanto, contornos ou superfícies curvas são aproximados por passos em escada, e tornam-se um problema, quando precisão grande é exigida na simulação. Existem alguns métodos desenvolvidos para FDTD como, por exemplo, estruturas com subcélulas que buscam superar esta limitação; mas outros métodos numéricos como, por exemplo, o método dos elementos finitos ou o método dos volumes finitos, são atualmente mais adequados para essas geometrias complexas.

- Tempo computacional alto. O método FDTD utiliza um incremento de tempo que depende do menor incremento de espaço utilizado. Este incremento de tempo tem um valor máximo para que a simulação seja estável e convirja para um resultado correto. Assim, para espaços de simulação eletricamente grandes ou variação grande dos comprimentos de onda da fonte utilizada (por exemplo, pulsos no tempo) pode ser necessário um longo tempo de simulação. Logo, outros métodos podem ser mais adequados para estes tipos de problemas.
- Dispersão numérica alta. Quando simulando um espaço físico eletricamente grande (referido ao menor comprimento de onda da fonte de sinal utilizado) um outro problema que surge com o método FDTD Yee é a dispersão numérica. A dispersão numérica é a diferença entre a velocidade de fase numérica da onda eletromagnética na grade e sua velocidade de fase real (física). Outra medida importante associada à dispersão numérica é a anisotropia numérica da velocidade de fase da onda, ou seja, a velocidade de fase numérica depende da direção de propagação da onda na grade. Quanto maior for essa anisotropia máxima (diferença entre a maior e a menor velocidade da onda numérica), mais difícil será reduzir a dispersão numérica. A consequência da dispersão numérica e a anisotropia associada à onda propagando-se na grade é a introdução de erros de fase que aumentam à medida que a onda numérica se distancia da fonte que a gerou. A partir de uma certa distância estes erros de fase aumentam a ponto de comprometer a precisão desejada e tornar os resultados obtidos sem utilidade.

Uma das formas de reduzir a dispersão numérica no método Yee (explícito no tempo) é utilizar mais pontos por comprimento de onda, ou seja, reduzir o incremento do espaço, porém isso também reduz o incremento máximo de tempo que garante

estabilidade da simulação. A consequência é a necessidade de mais memória e mais tempo de simulação. Vários métodos para reduzir a dispersão e anisotropia numéricas têm sido desenvolvidos para o método FDTD Yee explícito no tempo (SHLAGER; SCHNEIDER, 2003), normalmente, utilizando aproximações de alta ordem (SMITH *et al.*, 2012; CHEN *et al*., 2013; ZHU; CAO; ZHAO, 2014). Uma desvantagem do método Yee convencional (explícito no tempo) é que o tamanho máximo do passo de tempo é limitado pela condição de estabilidade (INAN; MARSHALL, 2011), o que aumenta o custo computacional no modelamento de estruturas geometricamente finas ou eletricamente grandes. Uma forma de contornar esta limitação é usar esquemas incondicionalmente estáveis (implícitos no tempo), atualmente muito pesquisados, que permitem usar maiores tamanhos de passo de tempo, o que reduz o tempo de simulação. Em métodos FDTD implícitos é importante aumentar a precisão, o que é conseguido reduzindo também a dispersão e anisotropia numéricas. Assim, variantes pesquisadas do método *Alternating-Direction-Implicit Finite-Difference Time-Domain* (ADI-FDTD) tem como um dos objetivos importantes reduzir a dispersão numérica (LIANG; YUAN, 2013; KONG; CHU; LI, 2014b). Um outro método FDTD implícito é o método *Locally One-Dimensional Finite-Difference Time-Domain* (LOD-FDTD), muito pesquisado atualmente (RANA; MOHAN, 2013; SAXENA; SRIVASTAVA, 2013; GRANDE; PEREDA, 2014; SAXENA; SRIVASTAVA, 2014; ZYGIRIDIS *et al*., 2015), e que normalmente tem um custo computacional menor que o método ADI-FDTD. Uma importante classe de métodos FDTD implícitos é denominada genericamente de *Split-Step Finite-Difference Time-Domain* (SS-FDTD); estes métodos utilizam vários subpassos e aproximações de alta ordem com muitas variações (KONG; CHU, 2011; GRANDE *et al*., 2013; HEH; TAN, 2014; KONG; CHU; LI, 2014a) com o objetivo de reduzir o custo computacional e a dispersão numérica. Métodos híbridos conhecidos como *Hybrid Implicit-Explicit Finite-Difference Time-Domain* (HIE-FDTD) são aplicados, por exemplo, em estruturas que exigem geometria fina em uma direção (método implícito), mas podem usar o método explícito para as outras duas direções (WANG *et al.*, 2014; ZHANG; ZHOU, 2015). Métodos FDTD, atualmente mais sofisticados, como os métodos *Multiresolution Time Domain* (MRTD) e o *Symplectic Finite-Difference Time-Domain* (SFDTD) apresentam melhores características de dispersão numérica que o método FDTD Yee convencional (SONG; YANG, 2012; SU *et al.*, 2013). Um método FDTD explícito muito interessante, como uma alternativa ao método FDTD Yee convencional (cartesiano), é denominado *Face-Centered-Cubic*

Finite-Difference Time-Domain (FCC-FDTD), o qual é considerado uma extensão lógica da grade bidimensional de hexágonos (LIU, 1996); esse método FCC-FDTD (POTTER; LAMOUREUX; NAUTA, 2011; POTTER; NAUTA, 2013) produz um pouco menos anisotropia e dispersão numéricas que o método FDTD Yee. A pesquisa já realizada por este autor, como será descrita nesta tese, propõe reduzir a anisotropia de velocidade numérica da grade tridimensional formada por hexaedros (ou células cúbicas) do método FDTD Yee, utilizando uma nova geometria tridimensional formada por prismas hexagonais, derivada a partir da grade bidimensional de hexágonos (LIU, 1996).

É importante, também, mencionar que, atualmente, o método DGTD (*Discontinuous Galerkin Time-Domain*) tem emergido como uma alternativa viável (DESCOMBES *et al.*, 2013) para o método FDTD ou o método FETD (*Finite-Element Time-Domain*). Ele utiliza ideias do método FVTD (*Finite-Volume Time-Domain*) para conectar elementos juntos em seus contornos (INAN; MARSHALL, 2011), e mantém quase todas as vantagens do método FEM (*Finite-Element Method*) como espectro grande de aplicações, geometrias complexas, etc. (JIN, 2010), produzindo um esquema estritamente local, explícito no tempo e com precisão de alta ordem (HESTHAVEN; WARBURTON, 2002; KABAKIAN; SHANKAR; HALL, 2004). No entanto, o método FDTD tem como vantagens em relação ao método DGTD a simplicidade na implementação computacional e maior velocidade de processamento (GARCIA *et al.*, 2008).

A seguir far-se-á uma breve revisão da análise vetorial, da teoria eletromagnética e do método FDTD Yee, que servirá como base teórica para o desenvolvimento da pesquisa com o método FDTD aplicado à grade formada por prismas hexagonais.

2.1 REVISÃO DE ANÁLISE VETORIAL

Antes de analisar as equações de Maxwell, introduzir-se-ão alguns conceitos importantes da análise vetorial (REITZ; MILFORD; CHRISTY, 1982). Por simplicidade, são utilizadas coordenadas cartesianas (x, y, z) no sistema dextrogiro, que é o mais usado em engenharia. Uma função vetorial ou vetor campo $\boldsymbol{F}$ é definida em coordenadas cartesianas como:

$$\boldsymbol{F} = F_X\boldsymbol{i} + F_Y\boldsymbol{j} + F_Z\boldsymbol{k} \tag{2.1}$$

onde ***i***, ***j*** e ***k*** são vetores unitários nas direções coordenadas x, y e z, respectivamente. Os valores F_X, F_Y, e F_Z são as amplitudes do vetor ***F*** nas direções coordenadas x, y e z, respectivamente. O módulo (*F*) do vetor campo ***F*** é definido por:

$$F = |\boldsymbol{F}| = \sqrt{{F_X}^2 + {F_Y}^2 + {F_Z}^2} \tag{2.2}$$

Apesar de não definido explicitamente, deve ser entendido que a função vetorial ***F*** pode ser variável no espaço e no tempo (*t*), ou seja, ***F*** = ***F***(*x, y, z, t*). Existem outros tipos de vetores como vetor distância, vetor número de onda, etc. Por exemplo, um vetor distância ***r*** é definido como:

$$\boldsymbol{r} = r_x\boldsymbol{i} + r_y\boldsymbol{j} + r_z\boldsymbol{k} \tag{2.3}$$

onde os valores r_x, r_y, e r_z são as distâncias do vetor distância ***r*** nas direções coordenadas x, y e z, respectivamente. E o módulo (*r*) do vetor distância ***r*** é definido por:

$$r = |\boldsymbol{r}| = \sqrt{{r_x}^2 + {r_y}^2 + {r_z}^2} \tag{2.4}$$

É possível, também, definir um vetor unitário na direção do vetor distância ***r***, como sendo:

$$\boldsymbol{a_r} = \frac{\boldsymbol{r}}{|\boldsymbol{r}|} = \frac{r_x}{|\boldsymbol{r}|}\boldsymbol{i} + \frac{r_y}{|\boldsymbol{r}|}\boldsymbol{j} + \frac{r_z}{|\boldsymbol{r}|}\boldsymbol{k} \tag{2.5}$$

Para vetores ***A***, ***B***, e ***C*** tem-se algumas operações básicas. A primeira delas é o produto escalar (ou ponto), que resulta num valor escalar, e é definido como:

$$\boldsymbol{A}.\boldsymbol{B} = (A_X\boldsymbol{i} + A_Y\boldsymbol{j} + A_Z\boldsymbol{k}).(B_X\boldsymbol{i} + B_Y\boldsymbol{j} + B_Z\boldsymbol{k})$$

e:

$$\boldsymbol{A}.\boldsymbol{B} = A_X B_X + A_Y B_Y + A_Z B_Z \tag{2.6}$$

Outra operação básica é o produto vetorial (ou cruz), que resulta num vetor, definido como:

$$\boldsymbol{C} = \boldsymbol{A} \times \boldsymbol{B} = \begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ A_X & A_Y & A_Z \\ B_X & B_Y & B_Z \end{bmatrix}$$

ou:

$$\boldsymbol{C} = (A_Y B_Z - A_Z B_Y)\boldsymbol{i} + (A_Z B_X - A_X B_Z)\boldsymbol{j} + (A_X B_Y - A_Y B_X)\boldsymbol{k} \tag{2.7}$$

onde ***C*** é um vetor perpendicular ao plano que contém os vetores ***A*** e ***B***, e cuja direção é dada pela regra da mão direita: os dedos da mão direita giram do vetor ***A*** para o vetor ***B***, e o polegar aponta na direção do vetor ***C***.

Existem três operações muito importantes em cálculo vetorial denominadas gradiente, divergente e rotacional. Uma notação comum para estas três operações utiliza um operador diferencial ∇ (chamado del ou nabla) definido, por simplicidade, em coordenadas cartesianas como:

$$\nabla = \frac{\partial}{\partial x}\boldsymbol{i} + \frac{\partial}{\partial y}\boldsymbol{j} + \frac{\partial}{\partial z}\boldsymbol{k} \tag{2.8}$$

Del é um operador diferencial, que é usado apenas em frente a uma função escalar ou vetorial de (*x, y, z, t*) que ele diferencia; é um vetor já que obedece às leis da álgebra vetorial.

Usando o operador diferencial, o gradiente de uma função escalar φ(*x, y, z, t*) é definido como:

$$\nabla\varphi = \frac{\partial\varphi}{\partial x}\boldsymbol{i} + \frac{\partial\varphi}{\partial y}\boldsymbol{j} + \frac{\partial\varphi}{\partial z}\boldsymbol{k} \tag{2.9}$$

O divergente de uma função vetorial ***F***(*x, y, z, t*) pode ser definido, no sistema cartesiano, como o produto escalar dela com o operador diferencial:

$$\nabla . \boldsymbol{F} = \frac{\partial F_X}{\partial x} + \frac{\partial F_Y}{\partial y} + \frac{\partial F_Z}{\partial z} \tag{2.10}$$

O rotacional de uma função vetorial ***F***(*x, y, z, t*) pode ser definido, no sistema cartesiano, como o produto vetorial dela com o operador diferencial:

$$\nabla \times \boldsymbol{F} = \begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ F_X & F_Y & F_Z \end{bmatrix}$$

$$\nabla \times \boldsymbol{F} = \left(\frac{\partial F_Z}{\partial y} - \frac{\partial F_Y}{\partial z}\right)\boldsymbol{i} + \left(\frac{\partial F_X}{\partial z} - \frac{\partial F_Z}{\partial x}\right)\boldsymbol{j} + \left(\frac{\partial F_Y}{\partial x} - \frac{\partial F_X}{\partial y}\right)\boldsymbol{k} \tag{2.11}$$

Dois importantes teoremas em análise vetorial são o teorema da divergência e o teorema de Stokes. O teorema da divergência é definido pela equação (2.12):

$$\oint_S \boldsymbol{F} . \boldsymbol{ds} = \int_V (\nabla . \boldsymbol{F})\, dv \tag{2.12}$$

O teorema da divergência da equação (2.12) declara que a divergência de um campo vetorial ***F*** sobre o volume *V* é igual ao fluxo do campo ***F*** que atravessa a superfície fechada *S*, que engloba todo o volume *V*.

O teorema de Stokes é definido pela equação (2.13):

$$\oint_C \boldsymbol{F} . \boldsymbol{dl} = \int_S (\nabla \times \boldsymbol{F}) . \boldsymbol{ds} \tag{2.13}$$

O teorema de Stokes da equação (2.13) declara que o fluxo do rotacional de um campo vetorial ***F*** através de uma superfície *S* (que é aberta) é igual à integral de linha do campo ***F*** ao longo do contorno fechado *C*, que engloba a superfície *S*.

Tem-se ainda algumas identidades vetoriais importantes para campo escalar φ ou campo vetorial ***F*** definidas pelas equações (2.14), (2.15), (2.16) e (2.17):

$$\nabla . \nabla \varphi = \nabla^2 \varphi \tag{2.14}$$

$$\nabla \times \nabla \varphi = 0 \tag{2.15}$$

$$\nabla . \nabla \times \boldsymbol{F} \; = \; 0 \tag{2.16}$$

$$\nabla \times (\nabla \times \boldsymbol{F}) \; = \; \nabla(\nabla . \boldsymbol{F}) - \nabla^2 \boldsymbol{F} \tag{2.17}$$

Em análise vetorial existem dois tipos de vetores especiais. O primeiro é o vetor irrotacional $\boldsymbol{F_i}$ cujo rotacional é zero, definido como:

$$\nabla \times \boldsymbol{F_i} = 0 \quad \text{e} \quad \nabla . \boldsymbol{F_i} \; \neq 0 \tag{2.18}$$

Um exemplo de vetor irrotacional é o campo elétrico entre cargas elétricas na eletrostática. Um outro vetor especial é o vetor solenoidal $\boldsymbol{F_s}$ cujo divergente é zero, definido como:

$$\nabla . \boldsymbol{F_s} = 0 \quad \text{e} \quad \nabla \times \boldsymbol{F_s} \; \neq 0 \tag{2.19}$$

Um exemplo de vetor solenoidal é o campo magnético em torno de um fio condutor em que circula uma corrente, que pode ser estacionária (contínua) ou variável no tempo. Em um campo solenoidal as linhas de campo sempre se fecham sobre si mesmas.

Embora uma função vetorial possa ter uma variação complicada no espaço e no tempo, pode ser mostrado que qualquer função vetorial $\boldsymbol{F}$ suave e que tende à amplitude zero no infinito (que é o caso de uma onda eletromagnética) pode ser decomposta em um vetor irrotacional $\boldsymbol{F_i}$ e um vetor solenoidal $\boldsymbol{F_s}$, tal que:

$$\boldsymbol{F} = \; \boldsymbol{F_i} + \; \boldsymbol{F_s} \tag{2.20}$$

Tomando a divergência e o rotacional da equação (2.20) obtêm-se:

$$\nabla . \boldsymbol{F} \; = \nabla . \boldsymbol{F_i} \quad \text{e} \quad \nabla \times \boldsymbol{F} \; = \; \nabla \times \boldsymbol{F_s} \tag{2.21}$$

As equações (2.21) demonstram que uma função vetorial $\boldsymbol{F}$ é totalmente determinada quando os valores de divergência e rotacional dessa função são especificados.

2.2 REVISÃO DA TEORIA ELETROMAGNÉTICA

Todos os fenômenos eletromagnéticos clássicos são governados por um conjunto compacto e abrangente de regras fundamentais conhecidas como equações de Maxwell (BALANIS, 1989; JIN, 2010). Foi o matemático e físico britânico James Clerk Maxwell quem, entre 1856 e 1865, escreveu uma série de artigos analisando e elucidando matematicamente o conhecimento experimental, existente na época, dos fenômenos elétricos e magnéticos. Uma das maiores contribuições de Maxwell foi introduzir o conceito de corrente de deslocamento na equação da lei de Ampère e inferir que a luz é uma onda eletromagnética. Tal hipótese foi confirmada 23 anos depois (em 1887) nos experimentos do físico alemão Heinrich Rudolf Hertz. Até os dias de hoje as equações de Maxwell permanecem válidas e consistentes com o conhecimento experimental observado em todo o espectro eletromagnético, estendendo-se para os raios cósmicos de frequência superior a 10^{22} Hz e para as assim chamadas micropulsações de frequências em torno de 10^{-3} Hz.

2.2.1 Equações de Maxwell nas formas integral e diferencial

As equações de Maxwell são baseadas em fatos experimentais. Da lei de Faraday sabe-se que um fluxo magnético variável no tempo induz força eletromotriz tal que:

$$\oint_C \boldsymbol{E}.\boldsymbol{dl} = -\int_S \frac{\partial \boldsymbol{B}}{\partial t}\boldsymbol{ds} \tag{2.22}$$

onde a intensidade do campo elétrico ***E***, integrada sobre o contorno fechado *C* na direção do incremento de distância ***dl***, é igual à taxa de diminuição da densidade de fluxo magnético ***B*** no tempo (*t*), integrado na direção do vetor de incremento de superfície ***ds***, normal a cada ponto da superfície aberta *S* englobada pelo contorno *C*. A direção de ***dl*** deve ser consistente com a direção do vetor incremento de superfície ***ds*** de acordo com a regra da mão direita. A unidade de intensidade de campo elétrico ***E*** é volt/metro (V/m) e a unidade de densidade de fluxo magnético ***B*** é tesla (T) ou weber/metro^2 (Wb/m^2).

A segunda equação de Maxwell é obtida da lei de Gauss, que é baseada no fato experimental que cargas elétricas atraem ou repelem umas às outras com uma força inversamente proporcional ao quadrado da distância entre elas (lei de Coulomb). Assim, tem-se:

$$\oint_S \boldsymbol{D}.\boldsymbol{ds} = \int_V \rho_v dv \tag{2.23}$$

onde a densidade de fluxo elétrico ***D***, integrada sobre a superfície fechada *S* que engloba o volume *V*, é igual à integral de volume da densidade volumétrica ρ_v de cargas elétricas neste volume *V*. A unidade de densidade de fluxo elétrico ***D*** é coulomb/metro^2 (C/m^2) e a unidade de densidade de cargas elétricas ρ_v é coulomb/metro^3 (C/m^3).

A terceira lei de Maxwell é uma generalização da lei de Ampère dada por:

$$\oint_C \boldsymbol{H}.\boldsymbol{dl} = \int_S \boldsymbol{J}.\boldsymbol{ds} + \int_S \frac{\partial \boldsymbol{D}}{\partial t}\boldsymbol{ds} \tag{2.24}$$

onde a integral de linha da intensidade de campo magnético ***H*** sobre qualquer contorno fechado *C* deve ser igual à corrente elétrica total (em ampère) englobada por este contorno *C*. O contorno fechado *C* engloba a superfície aberta *S*. A intensidade de campo magnético ***H*** tem unidade de ampère/metro (A/m), e a densidade de corrente de condução ***J*** tem unidade de ampère/metro^2 (A/m^2). A equação (2.24), apenas com o 1º termo do lado direito (corrente de condução) é a lei de Ampère baseada nos experimentos do físico dinamarquês Hans Christian Oersted. O 2º termo no lado direito da equação (2.24) é a corrente de deslocamento introduzida teoricamente por Maxwell em 1862 e verificada experimentalmente em 1887 por Hertz. Este 2° termo prova a existência das ondas eletromagnéticas, e indica que a luz é uma onda eletromagnética.

A quarta equação de Maxwell, conhecida como a lei de Gauss para campos magnéticos, é dada por:

$$\oint_S \boldsymbol{B}.\boldsymbol{ds} = 0 \tag{2.25}$$

É baseada no fato de que não existem cargas magnéticas livres na natureza, ou seja, monopolos magnéticos e, consequentemente, as linhas de campo magnético sempre fecham sobre si mesmas (campo solenoidal como expressado pela equação (2.19)).

Uma outra relação importante, associada às equações de Maxwell de (2.22) a (2.25), é a equação da continuidade, que expressa o princípio da conservação da carga elétrica como:

$$\oint_S \boldsymbol{J}.\boldsymbol{ds} = -\frac{\partial}{\partial t}\int_V \rho_v dv \tag{2.26}$$

A densidade de corrente $\boldsymbol{J}$, integrada sobre a superfície fechada *S* (que engloba o volume *V*), é igual à taxa de diminuição no tempo (*t*) da densidade volumétrica de cargas elétricas ρ_v integrada dentro do volume *V*.

As formas diferenciais das equações de Maxwell, na forma integral de (2.22) a (2.25), e da equação da continuidade, na forma integral em (2.26), podem ser obtidas usando o teorema da divergência da equação (2.12) ou o teorema de Stokes da equação (2.13). Assim tem-se:

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t} \tag{2.27}$$

$$\nabla.\boldsymbol{D} = \rho_v \tag{2.28}$$

$$\nabla \times \boldsymbol{H} = \boldsymbol{J} + \frac{\partial \boldsymbol{D}}{\partial t} \tag{2.29}$$

$$\nabla.\boldsymbol{B} = 0 \tag{2.30}$$

$$\nabla.\boldsymbol{J} = -\frac{\partial \rho_v}{\partial t} \tag{2.31}$$

É também possível, a partir das equações na forma diferencial de (2.27) a (2.31), aplicando o teorema da divergência (2.12) ou o teorema de Stokes (2.13), obter as equações na forma integral de (2.22) a (2.25).

A equação diferencial da continuidade (2.31) pode ser derivada a partir das equações (2.28) e (2.29), indicando que estas duas equações de Maxwell não são

totalmente independentes, se o princípio da conservação de carga elétrica é aceito. Logo, usando as equações (2.29) e (2.31) pode-se obter a equação (2.28); a derivação da equação (2.28) pode ser feita aplicando o divergente em ambos os lados da equação (2.29), e então usando na equação resultante a identidade vetorial (2.16) e a equação (2.31).

2.2.2 Relações constitutivas

Associadas às equações de Maxwell, tem-se as relações constitutivas que relacionam as densidades de fluxos elétrico (***D***) e magnético (***B***) com as intensidades de campos elétrico (***E***) e magnético (***H***), respectivamente, como:

$$\boldsymbol{D} = \varepsilon . \boldsymbol{E} \tag{2.32}$$

$$\boldsymbol{B} = \mu . \boldsymbol{H} \tag{2.33}$$

onde os parâmetros característicos do meio físico são a permissividade elétrica ($\boldsymbol{\varepsilon}$) com unidade de farad/metro (F/m) e a permeabilidade magnética (μ) com unidade de henry/metro (H/m). A rigor a densidade de fluxo elétrico ***D*** e a intensidade de campo magnético ***H*** são as variáveis independentes das características do meio. Outra relação importante é:

$$\boldsymbol{J_c} = \sigma . \boldsymbol{E} \tag{2.34}$$

onde a densidade de corrente de condução $\boldsymbol{J_c}$ num meio está relacionada à intensidade de campo elétrico ***E*** pela condutividade elétrica (σ) do meio. A condutividade elétrica tem unidade de siemens/metro (S/m).

As propriedades eletromagnéticas de um meio (BALANIS, 1989; JIN, 2010) estão relacionadas às três relações constitutivas dadas pelas equações (2.32), (2.33) e (2.34). Um meio exibe dependência espacial se quaisquer dos três parâmetros característicos do meio ($\boldsymbol{\varepsilon}$, μ, σ) é função da posição no espaço; neste caso o meio é chamado não-homogêneo ou heterogêneo; em caso contrário, é chamado homogêneo.

Se a direção da densidade de fluxo elétrico (***D***) é paralela à direção de intensidade de campo elétrico (***E***) e a direção da densidade de fluxo magnético (***B***) é paralela à direção da intensidade de campo magnético (***H***), o meio é chamado isotrópico, de outra maneira, é chamado anisotrópico; neste caso a permissividade elétrica (ε) ou a permeabilidade magnética (μ) são tensores.

Se quaisquer dos três parâmetros característicos do meio (ε, μ, σ) é função do tempo, o meio é chamado não-estacionário; caso contrário, é chamado estacionário.

Se quaisquer dos três parâmetros característicos do meio (ε, μ, σ) depende das intensidades de campos elétrico ou magnético, o meio é considerado não-linear; caso contrário, é considerado linear.

Um meio é dependente da frequência (em Hz) se a permissividade elétrica ou a permeabilidade magnética depende da frequência (f) tal que: $\varepsilon = \varepsilon(f)$ ou $\mu = \mu(f)$; neste caso o meio é chamado dispersivo; de outra maneira é não dispersivo. Como exemplo, a permissividade elétrica de um material no domínio da frequência é dada por:

$$\varepsilon(\omega) = \varepsilon_0[\,1 + \chi_e(\omega)\,] \tag{2.35}$$

onde $\chi_e(\omega)$ é a transformada de Fourier da susceptibilidade elétrica do material, ε_0 é a permissividade elétrica no vácuo ($\varepsilon_0 = 8{,}854\ 10^{-12}$ F/m) e a frequência angular ω (em rad/s) é definida como $\omega = 2\pi.f$, onde f é a frequência (em Hz). Assim, no domínio do tempo, a relação constitutiva é expressa em termos de uma convolução integral:

$$\boldsymbol{D}(\boldsymbol{r},t) = \varepsilon_0.\varepsilon_\infty.\boldsymbol{E}(\boldsymbol{r},t) + \varepsilon_0 \int_0^t \boldsymbol{E}[\boldsymbol{r},(t-\tau)]\,.\chi_e\,(\boldsymbol{r},t).d\tau \tag{2.36}$$

onde $\chi_e(\boldsymbol{r},t)$ é a susceptibilidade elétrica, isto é, a transformada de Fourier inversa de $\chi_e(\omega)$, e ε_∞ é a permissividade relativa (adimensional) para o final superior da banda de frequências considerada para uma particular aplicação. Observa-se que a susceptibilidade elétrica χ_e (ou a permissividade elétrica ε) é uma função da frequência que, por sua vez, é dependente das propriedades particulares de um certo material.

Uma importante característica do método FDTD, quando comparado com outras técnicas, é a facilidade de aplicação na solução de problemas eletromagnéticos

envolvendo a propagação em meios que não são simples, tais como meios não homogêneos, dispersivos, anisotrópicos e não-lineares.

2.2.3 Campos harmônicos no tempo

Embora o algoritmo FDTD use as equações de Maxwell no domínio do tempo, há ocasiões em que o entendimento das equações no domínio da frequência é prático. Muitas aplicações como radiodifusão de rádio ou TV, radar, redes *Wireless*, micro-ondas, óptica, etc., utilizam uma banda estreita de frequências, cujo comportamento é muito similar a um sinal senoidal de frequência única. Dessa forma, é possível obter as equações de Maxwell na forma senoidal em estado estacionário (ou harmônica no tempo) como:

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t} \qquad \nabla \times \mathbf{E} = -j\omega\mathbf{B} \tag{2.37}$$

$$\nabla . \boldsymbol{D} = \rho_v \qquad \nabla . \mathbf{D} = \rho_v \tag{2.38}$$

$$\nabla \times \boldsymbol{H} = \boldsymbol{J} + \frac{\partial \boldsymbol{D}}{\partial t} \qquad \nabla \times \mathbf{H} = \mathbf{J} + j\omega\mathbf{D} \tag{2.39}$$

$$\nabla . \boldsymbol{B} = 0 \qquad \nabla . \mathbf{B} = 0 \tag{2.40}$$

onde $j = \sqrt{-1}$ é a unidade imaginária e ω é a frequência angular (rad/s). Observa-se que nas equações de (2.37) a (2.40) os vetores ***E***, ***D***, ***H***, ***B***, ***J*** e o escalar ρ_v são quantidades reais (mensuráveis) que variam com o tempo, enquanto os vetores **E**, **D**, **H**, **B**, **J** e o escalar ρ_v (“todos sem itálico”) são fasores complexos, que não variam com o tempo. Por exemplo, o fasor para um campo elétrico é definido como:

$$\mathbf{E}(\boldsymbol{r}) = E_0 e^{-j\boldsymbol{k}.\boldsymbol{r}} = E_0 e^{-j\phi} \tag{2.41}$$

onde o ângulo de fase Φ é obtido a partir do produto escalar entre o vetor distância ***r*** e o vetor número de onda ***k***, que são definidos como:

$$\boldsymbol{r} = r_x\boldsymbol{i} + r_y\boldsymbol{j} + r_z\boldsymbol{k} \tag{2.42}$$

$$\boldsymbol{k} = k_x \boldsymbol{i} + k_y \boldsymbol{j} + k_z \boldsymbol{k} \tag{2.43}$$

O conceito de vetor número de onda será explicado mais adiante. Para obter a forma no tempo, multiplica-se a forma fasorial do vetor campo elétrico da equação (2.41) por $e^{j\omega t}$, e toma-se a parte real como mostrado a seguir:

$$\boldsymbol{E}(\boldsymbol{r}, t) = Re\{\mathbf{E}(\boldsymbol{r}) e^{j\omega t}\} = Re\{E_0 e^{-j\phi} e^{j\omega t}\} = E_0 . cos(\omega t - \phi) \tag{2.44}$$

A forma fasorial do vetor campo pode ser entendida como a transformada de Fourier do vetor campo variando no tempo, bastando substituir $\partial/\partial t$ por $j\omega$, como mostrado nas equações (2.37) a (2.40).

2.2.4 Equações de Maxwell generalizadas

Neste ponto será conveniente introduzir nas equações fasoriais de Maxwell alguns termos fictícios de forma a simplificar posteriores aplicações da teoria eletromagnética neste texto. Assim, ter-se-á o seguinte conjunto de equações:

$$\nabla \times \mathbf{E} = -\sigma_m \mathbf{H} - j\omega\mu \mathbf{H} - \mathbf{M_i} \tag{2.45}$$

$$\nabla . \mathbf{D} = \rho_v \tag{2.46}$$

$$\nabla \times \mathbf{H} = \sigma \mathbf{E} + j\omega\varepsilon \mathbf{E} + \mathbf{J_i} \tag{2.47}$$

$$\nabla . \mathbf{B} = \rho_{vm} \tag{2.48}$$

Apesar de que no conhecimento cientifico atual não existem “cargas magnéticas” na natureza, é útil introduzir os seguintes termos fictícios: condutividade magnética σ_m com unidade de ohms/metro (Ω/m), densidade de carga magnética ρ_{vm} com unidade de weber/metro^3 (Wb/m^3) e fonte (sinal aplicado externamente) de densidade de corrente magnética $\mathbf{M_i}$ com unidade de volt/metro^2 (V/m^2). O termo densidade de

corrente $\mathbf{J_i}$ corresponde a uma fonte (fisicamente real) de sinal aplicada externamente, ou seja, imposta sobre o sistema a ser analisado.

2.2.5 Permissividade e permeabilidade complexas

Muitos materiais dielétricos são bons isoladores apresentando uma corrente contínua desprezível, ou seja, uma condutividade elétrica muito baixa ($\sigma \rightarrow 0$). No entanto, em frequências altas, os campos elétricos interagem com as moléculas do dielétrico, produzindo perdas (sob a forma de calor) nesse meio (BALANIS, 1989). Estas perdas são proporcionais ao valor da parte imaginária da permissividade complexa relativa (adimensional) do dielétrico, dada por:

$$\varepsilon_r = \varepsilon'_r - j\varepsilon''_r \tag{2.49}$$

Os valores $\boldsymbol{\varepsilon'_r}$ e $\boldsymbol{\varepsilon''_r}$ são funções da frequência angular (ω). Assim a permissividade elétrica absoluta complexa do meio é dada por:

$$\varepsilon = \varepsilon_r . \varepsilon_0 \tag{2.50}$$

onde $\boldsymbol{\varepsilon_0}$ é a permissividade elétrica no vácuo ($\boldsymbol{\varepsilon_0}$ = 8,854 10^{-12} F/m). Aplicando as equações (2.49) e (2.50) na equação (2.47) e desprezando o termo $\mathbf{J_i}$, obtém-se a equação (2.51):

$$\nabla \times \mathbf{H} = j\omega\varepsilon_0 \left[\varepsilon'_r - j\left(\varepsilon''_r + \frac{\sigma}{\omega\varepsilon_0}\right)\right]\mathbf{E} \tag{2.51}$$

Por simplicidade, incorpora-se o valor de $\boldsymbol{\varepsilon''_r}$ à nova condutividade elétrica denominada σ' (SCHNEIDER, 2013), tal que:

$$\nabla \times \mathbf{H} = j\omega\varepsilon_0 \left[\varepsilon'_r - j\frac{\sigma'}{\omega\varepsilon_0}\right]\mathbf{E} = j\omega\varepsilon\mathbf{E} \tag{2.52}$$

com a permissividade elétrica absoluta complexa ($\boldsymbol{\varepsilon}$) do meio expressa por:

$$\varepsilon = \varepsilon' - j\frac{\sigma'}{\omega} \qquad \text{e} \qquad \varepsilon' = \varepsilon'_r \varepsilon_0 \tag{2.53}$$

Lembrando que: $\varepsilon' = \varepsilon'(\omega)$ e $\sigma' = \sigma'(\omega)$.

Para materiais magnéticos é possível obter, também, uma permeabilidade magnética complexa relativa (adimensional) dada por:

$$\mu_r = \mu'_r - j\mu''_r \tag{2.54}$$

Os valores μ'_r e μ''_r são funções da frequência angular (ω). Assim, a permeabilidade magnética absoluta complexa do meio é dada por:

$$\mu = \mu_r . \mu_0 \tag{2.55}$$

onde μ_0 é a permeabilidade magnética no vácuo ($\mu_0 = 4\pi.10^{-7}$ H/m). Aplicando as equações (2.54) e (2.55) na equação (2.45) e desprezando o termo $\mathbf{M_i}$, obtém-se:

$$\nabla \times \mathbf{E} = -j\omega\mu_0 \left[\mu'_r - j\left(\mu''_r + \frac{\sigma_m}{\omega\mu_0}\right)\right]\mathbf{H} \tag{2.56}$$

Os valores de μ'_r e μ''_r podem ser valores reais (existentes na natureza), mas a condutividade magnética σ_m é um valor fictício, sendo o seu valor na natureza igual a zero. Por simplicidade, incorpora-se o valor de μ''_r dentro da nova condutividade magnética denominada σ'_m (SCHNEIDER, 2013), como mostrado a seguir:

$$\nabla \times \mathbf{E} = -j\omega\mu_0 \left[\mu'_r - j\frac{\sigma'_m}{\omega\mu_0}\right]\mathbf{H} = j\omega\mu\mathbf{H} \tag{2.57}$$

com a permeabilidade magnética absoluta complexa do meio definida por:

$$\mu = \mu' - j\frac{\sigma'_m}{\omega} \qquad \text{e} \qquad \mu' = \mu'_r \mu_0 \tag{2.58}$$

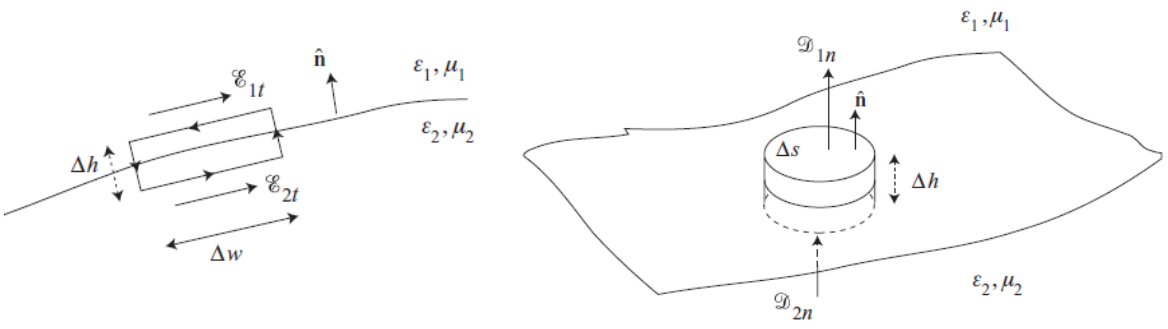
e lembrando novamente que: $\mu' = \mu'(\omega)$ e $\sigma'_m = \sigma'_m(\omega)$.

Quando desenvolvendo condições de contorno artificiais com o método FDTD para absorver campos incidentes e reduzir a níveis desprezíveis os campos refletidos,

de forma a simular um espaço infinito, a possibilidade de perdas elétricas dada pela equação (2.52) e, também, de perdas magnéticas dada pela equação (2.57), será muito útil (ver capítulo 5). As equações (2.52) e (2.57) também podem ser aplicadas em materiais dispersivos e com perdas, necessitando-se fazer a transformada de Fourier inversa dessas equações, resultando em uma convolução integral no domínio do tempo, semelhante à equação (2.36). Apesar dessa complexidade adicional, existem algoritmos eficientes com o método FDTD para meios dispersivos.

2.2.6 Condições de contorno

Nas interfaces entre materiais é que ocorrem alterações de interesse na propagação de ondas eletromagnéticas, tais como reflexão e refração. Simulações com o método FDTD ocorrem frequentemente em superfícies que não podem ser facilmente descritas analiticamente. No entanto, há um conjunto de regras que descrevem as condições de contorno com precisão. Essas condições de contorno são obtidas das equações de Maxwell na forma integral. Utilizar-se-á também o conceito de condutor elétrico perfeito ($\sigma = \infty$) que simplifica a análise teórica. Considera-se a Figura 2.1 que ilustra a interface entre dois meios distintos.



(a) (b)

Figura 2.1- Condições de contorno na interface entre dois meios distintos. Fonte: (INAN; MARSHALL, 2011).

Como mostrado na Figura 2.1a, considera-se dois meios com parâmetros constitutivos σ_1, ε_1, μ_1 e σ_2, ε_2, μ_2. Aplica-se a equação de Maxwell (2.22), repetida aqui por conveniência, sobre o contorno retangular *C* da Figura 2.1a, tal que:

$$\oint_C \boldsymbol{E}.\boldsymbol{dl} = -\int_S \frac{\partial \boldsymbol{B}}{\partial t}\boldsymbol{ds} \tag{2.59}$$

Fazendo Δ*h* tender a zero tal que a área do lado direito da equação (2.59) também tende a zero, obtém-se que a componente tangencial do campo elétrico é contínua através da interface:

$$\boldsymbol{E_1}.\boldsymbol{a_t}\Delta w - \boldsymbol{E_2}.\boldsymbol{a_t}\Delta w = 0$$

$$E_{1t} - E_{2t} = 0 \;\Rightarrow\; E_{1t} = E_{2t} \tag{2.60a}$$

Ou ainda:

$$\boldsymbol{a_n} \times (\boldsymbol{E_1} - \boldsymbol{E_2}) = 0 \quad \text{com } \sigma_1 \text{ e } \sigma_2 \text{ finitos} \tag{2.60b}$$

onde $\boldsymbol{a_t}$ e $\boldsymbol{a_n}$ são o vetor unitário tangente e o vetor unitário normal ao lado Δ*w* do contorno, respectivamente.

Aplicando a equação de Maxwell (2.24), repetida aqui por conveniência, sobre o contorno retangular *C* da Figura 2.1a, tal que:

$$\oint_C \boldsymbol{H}.\boldsymbol{dl} = \int_S \boldsymbol{J}.\boldsymbol{ds} + \int_S \frac{\partial \boldsymbol{D}}{\partial t}\boldsymbol{ds} \tag{2.61}$$

e considerando a inexistência de uma densidade de corrente na superfície da interface, obtém-se uma equação, similar à equação (2.60), indicando a continuidade das componentes tangenciais dos campos magnéticos na interface, tal que:

$$\boldsymbol{H_1}.\boldsymbol{a_t}\Delta w - \boldsymbol{H_2}.\boldsymbol{a_t}\Delta w = 0$$

$$H_{1t} - H_{2t} = 0 \;\Rightarrow\; H_{1t} = H_{2t} \tag{2.62a}$$

Ou ainda:

$$\boldsymbol{a_n} \times (\boldsymbol{H_1} - \boldsymbol{H_2}) = 0 \quad \text{com } \sigma_1 \text{ e } \sigma_2 \text{ finitos} \tag{2.62b}$$

Uma exceção a esta regra ocorre quando o meio 2 for um condutor elétrico perfeito ($\sigma_2 = \infty$) e uma densidade de corrente de superfície ($\boldsymbol{J_s}$) existir. Assim tem-se:

$$\boldsymbol{a_n} \times \boldsymbol{H_1} = \boldsymbol{J_s} \quad \text{com } \sigma_1 \text{ finito e } \sigma_2 = \infty \tag{2.63}$$

Observa-se que o campo magnético $\boldsymbol{H_2}$ dentro do condutor elétrico perfeito é zero. A densidade de corrente de superfície ($\boldsymbol{J_s}$), com unidade de ampères/metro (A/m), é uma película extremamente fina contida na superfície do condutor perfeito, e definida como:

$$\boldsymbol{J_s} = \frac{\Delta I}{\Delta w}\,\boldsymbol{a_{nw}} \quad \text{onde:} \quad \boldsymbol{a_{nw}} = \boldsymbol{a_t} \times \boldsymbol{a_n} \tag{2.64}$$

O vetor unitário $\boldsymbol{a_{nw}}$ aponta na direção do incremento de corrente (ΔI) que se movimenta na superfície do condutor perfeito (interface entre os dois meios), e na direção transversal à largura Δw.

Em adição às condições de contorno das componentes tangenciais dos campos elétrico e magnético através da interface, as componentes normais dos campos também obedecem a certas condições de contorno. Aplicando a equação de Maxwell da lei de Gauss (2.23), repetida aqui por conveniência, na caixa cilíndrica da Figura 2.1b:

$$\oint_S \boldsymbol{D}.\boldsymbol{ds} = \int_V \rho_v dv \tag{2.65}$$

Fazendo Δh tender a zero, e supondo a existência de uma densidade de carga de superfície (ρ_s), com unidade de coulombs/metro^2 (C/m^2), tal como na superfície de um condutor metálico ou na interface entre dois dielétricos com perdas ($\sigma_1 \neq 0$ e $\sigma_2 \neq 0$), tem-se a seguinte relação para as componentes normais da densidade de fluxo elétrico na interface:

$$D_{1n} - D_{2n} = \rho_s \tag{2.66a}$$

Ou ainda:

$$\boldsymbol{a_n}.(\boldsymbol{D_1} - \boldsymbol{D_2}) = \rho_s \tag{2.66b}$$

No caso de não existirem cargas elétricas de superfície, a componente normal do fluxo elétrico é contínua na interface: $D_{1n} = D_{2n}$. Para um condutor perfeito ($\sigma_2 = \infty$) tem-se:

$$D_{1n} = \rho_s \quad \Rightarrow \quad \boldsymbol{a_n}.\boldsymbol{D_1} = \rho_s \tag{2.67}$$

o que implica que $\boldsymbol{D_2} = 0$ ou $\boldsymbol{E_2} = 0$ no interior do condutor perfeito.

Uma relação similar aplica-se às componentes normais da densidade de fluxo magnético na interface entre os dois meios; aplicando a equação de Maxwell da lei de Gauss magnética (2.25), repetida aqui por conveniência, na caixa cilíndrica da Figura 2.1b:

$$\oint_S \boldsymbol{B}.\boldsymbol{ds} = 0 \tag{2.68}$$

obtém-se:

$$B_{1n} = B_{2n} \quad \Rightarrow \quad \boldsymbol{a_n}.(\boldsymbol{B_1} - \boldsymbol{B_2}) = 0 \tag{2.69}$$

Como não existem cargas magnéticas livres na natureza, a componente normal da densidade de fluxo magnético é sempre contínua na interface entre dois meios.

2.2.7 Equação da onda e velocidades de fase e grupo

No método FDTD Yee ou no método FDTD para prismas hexagonais utilizar-se-ão as equações de diferença finita obtidas a partir das equações de Maxwell na forma integral ou na forma diferencial. No entanto, far-se-á uma breve análise da equação da onda para se compreender os importantes conceitos de velocidade de fase e velocidade de grupo (BALANIS, 1989).

Considerando um meio livre de fontes, isto é, $\rho_v = 0$ e $\boldsymbol{J} = 0$, e que os parâmetros $\boldsymbol{\varepsilon}$ e μ não são funções do tempo, então as equações de Maxwell (2.27) e (2.29), usando as relações constitutivas de (2.32) e (2.33) tornam-se:

$$\nabla \times \boldsymbol{E} = -\mu \frac{\partial \boldsymbol{H}}{\partial t} \tag{2.70}$$

$$\nabla \times \boldsymbol{H} = \varepsilon \frac{\partial \boldsymbol{E}}{\partial t} \tag{2.71}$$

Tomando o rotacional da equação (2.70) e inserindo (2.71), obtém-se:

$$\nabla \times \nabla \times \boldsymbol{E} = -\mu \frac{\partial(\nabla \times \boldsymbol{H})}{\partial t} = -\mu\varepsilon \frac{\partial^2 \boldsymbol{E}}{\partial t^2} \tag{2.72}$$

Usando a identidade vetorial da equação (2.17) para o campo elétrico tem-se:

$$\nabla \times (\nabla \times \boldsymbol{E}) = \nabla(\nabla . \boldsymbol{E}) - \nabla^2 \boldsymbol{E} \tag{2.73}$$

Mas da equação (2.28) como $\rho_v = 0$, o divergente do campo elétrico $\boldsymbol{E}$ é zero na equação (2.73), que aplicada na equação (2.72), produz o seguinte resultado:

$$\nabla^2 \boldsymbol{E} - \mu\varepsilon \frac{\partial^2 \boldsymbol{E}}{\partial t^2} = 0 \tag{2.74}$$

Esta é a equação da onda na sua forma mais simples, pois o meio é considerado homogêneo, linear, isotrópico e não dispersivo. Para o caso unidimensional, com $\boldsymbol{E} = E_X \boldsymbol{i}$ e $\partial/\partial x = \partial/\partial y = 0$, a solução geral da equação da onda é dada por:

$$E_X(z,t) = f(z - c.t) + g(z + c.t) \tag{2.75}$$

onde *f* e *g* são funções gerais da onda propagando-se nas direções +*z* e –*z*, respectivamente, com velocidade de fase (*c*) dada por:

$$c = \frac{1}{\sqrt{\mu\varepsilon}} \tag{2.76}$$

Para a equação de Maxwell na forma senoidal, substitui-se $\partial/\partial t$ por $j\omega$ na equação (2.74), obtendo-se:

$$\frac{\partial^2 E_X}{\partial z^2} + k^2 E_X = 0 \qquad e \qquad k = \omega\sqrt{\mu\varepsilon} = 2\pi/\lambda \tag{2.77}$$

onde *k* é o número de onda ou constante de fase (em rad/m), *ω* é a frequência angular (em rad/s) e *λ* é o comprimento de onda (em metros). A solução geral da equação (2.77) é:

$$E_X(z,t) = Re\{(C_1 e^{-jk.z} + C_2 e^{jk.z}) e^{j\omega t}\}$$

$$E_X(z,t) = C_1.\cos(\omega t - kz) + C_2.\cos(\omega t + kz) \tag{2.78}$$

A equação (2.78) descreve duas ondas senoidais propagando-se nas direções +*z* e –*z*, respectivamente. A velocidade de fase é obtida seguindo-se um ponto de fase constante, dado pelo argumento dos cossenos, de forma que:

$$\omega t \pm kz = constante \tag{2.79}$$

Aplicando para todos os termos da equação (2.79), a derivada em função do tempo, obtém-se:

$$c = \frac{dz}{dt} = \frac{\omega}{k} \tag{2.80}$$

onde a velocidade de fase (*c*) é função de *ω* e *k*. Para um meio dispersivo, o número de onda *k* pode ser uma função complicada de *ω*, ou vice-versa. Assim também a velocidade de fase é função de *ω* ou *k*.

Associada à velocidade de fase, que é específica a uma certa frequência (*ω*), tem-se a velocidade de grupo, que corresponde à velocidade de um pacote ou envelope, como, por exemplo, um pulso gaussiano no domínio do tempo. Este pulso gaussiano é formado no domínio da frequência por um espectro de frequências. Se o meio for dispersivo, cada frequência deste pulso propagar-se-á com um distinto coeficiente de atenuação e uma distinta velocidade de fase e, consequentemente, produzirá distorção no envelope ou formato do pulso. Este envelope tem uma velocidade conhecida como velocidade de grupo v_g dada pela seguinte derivada:

$$v_g = \frac{d\omega}{dk} \tag{2.81}$$

Frequentemente, a velocidade de grupo é também a velocidade com a qual a energia ou informação é transmitida. A velocidade de grupo é, também, em meios dispersivos uma função complicada de ω ou k. Num meio sem dispersão com $\omega = \pm c.k$, onde k é uma constante, encontra-se que $v_g = d\omega/dk = \pm c$, ou seja, as velocidades de fase e grupo são iguais.

Mas quando o método FDTD é utilizado para qualquer meio, mesmo um meio sem dispersão, sempre haverá uma dispersão chamada de dispersão numérica, intrínseca ao processo de discretização da grade, que corresponde à diferença entre a velocidade de fase numérica e a velocidade de fase real (física) da onda eletromagnética. O valor da dispersão numérica também dependerá da direção de propagação da onda numérica, caracterizando uma anisotropia de velocidade de fase numérica. O levantamento da curva de anisotropia numérica em função da direção (ou de ângulos em coordenadas esféricas) permite medir a máxima diferença (em %) entre as velocidades de fase numérica máxima e mínima na grade em um certo plano desejado.

2.3 REVISÃO DO MÉTODO FDTD YEE

Antes de analisar a nova formulação com prismas hexagonais, é importante relembrar e compreender como são obtidas as equações de diferença finita para o método FDTD tradicional com células hexaédricas, também conhecido como método FDTD Yee (YEE, 1966). A equação da onda (2.74) é classificada como uma equação diferencial do tipo hiperbólico. Da mesma forma as duas equações diferenciais parciais de Maxwell (lei de Faraday (2.27) e lei de Ampère-Maxwell (2.29)) são também classificadas como hiperbólicas (INAN; MARSHALL, 2011).

2.3.1 Equações de diferença finita a partir de derivadas no tempo e espaço

Na formulação do método FDTD é necessário obter equações de diferença finita a partir de derivadas parciais no tempo e espaço. A derivação destas equações de diferença finita é feita a partir, de forma geral, da expansão em séries de Taylor.

Primeiro analisar-se-á a obtenção das equações de diferença finita no domínio do tempo a partir de uma função *f*(*x*, *t*), para o caso unidimensional, em torno de um tempo t_0 como:

$$f(x,t) = f(x,t_0) + (t-t_0)\frac{\partial f}{\partial t}\Big|_x^{t_0} + \frac{1}{2}(t-t_0)^2\frac{\partial^2 f}{\partial t^2}\Big|_x^{t_0} + \cdots + \frac{1}{n!}(t-t_0)^n\frac{\partial^n f}{\partial t^n}\Big|_x^{t_0} + \cdots \quad (2.82)$$

Considere a expansão em série de Taylor no tempo de f_i^{n+1} para um ponto de grade (*i*, *n*) = (*i.Δx*, *n.Δt*), onde *i* é o índice no eixo cartesiano x e *n* é o índice no tempo, tal que:

$$f_i^{n+1} = f_i^n + \Delta t\frac{\partial f}{\partial t}\Big|_i^n + \frac{1}{2}\Delta t^2\frac{\partial^2 f}{\partial t^2}\Big|_i^n + \cdots + \frac{1}{n!}\Delta t^n\frac{\partial^n f}{\partial t^n}\Big|_i^n + \cdots \text{ onde: } \Delta t = t^{n+1} - t^n \quad (2.83)$$

Os índices *n* e *n+1* na equação (2.83) correspondem aos tempos t_0 e *t* na equação (2.82), respectivamente. Solucionando a equação (2.83) para $[\partial f / \partial t]_i^n$ obtém-se:

$$\frac{\partial f}{\partial t}\Big|_i^n = \frac{f_i^{n+1} - f_i^n}{\Delta t} - \frac{1}{2}\Delta t\,\frac{\partial^2 f}{\partial t^2}\Big|_i^n - \cdots - \frac{1}{n!}\Delta t^{(n-1)}\frac{\partial^n f}{\partial t^n}\Big|_i^n - \cdots \quad (2.84)$$

A aproximação de diferença finita da derivada de primeira ordem, a partir da equação (2.84), é:

$$\frac{\partial f}{\partial t}\Big|_i^n \simeq \frac{f_i^{n+1} - f_i^n}{\Delta t} \quad (2.85)$$

Na equação (2.84) os termos restantes, que são multiplicados por potências mais altas que a primeira de Δ*t*, são ignorados; assim, o erro restante é de 1ª ordem em Δ*t*, dado como:

$$O(\Delta t) = -\frac{1}{2}\Delta t\,\frac{\partial^2 f}{\partial t^2}\Big|_i^n \quad (2.86)$$

A equação (2.85) é conhecida como diferença direta de 1ª ordem, pois ignora-se o resto de 1ª ordem O(Δ*t*) como definido na equação (2.86).

Existem outros caminhos para obter a aproximação da derivada de 1ª ordem, como expandindo em séries de Taylor f_i^{n+1} e f_i^{n-1} em torno do ponto de grade (i, n) e subtraindo a segunda da primeira para obter uma aproximação centrada de 2ª ordem no tempo para $[\partial f / \partial t]_i^n$ mais acurada, dada por:

$$\left.\frac{\partial f}{\partial t}\right|_i^n \simeq \frac{f_i^{n+1} - f_i^{n-1}}{2.\Delta t} \tag{2.87}$$

O erro é de 2ª ordem no tempo:

$$O[(\Delta t)^2] = -\frac{1}{6}(\Delta t)^2 \left.\frac{\partial^3 f}{\partial t^3}\right|_i^n \tag{2.88}$$

Os termos com potências de Δt maior que dois são ignorados. A equação (2.87) é conhecida como diferença centrada de 2ª ordem, pois é centrada no tempo (n), com valores de campo para tempos acima ($n+1$) e abaixo ($n-1$) deste tempo.

Outra formulação, que será conveniente para o uso com o método FDTD Yee, utiliza valores intermediários (ou temporários) de derivada no tempo para meios passos discretos no tempo. Assim, uma aproximação de diferença direta de 2ª ordem de $[\partial f / \partial t]_i^{n+1/2}$ pode ser obtida subtraindo expansões de séries de Taylor de f_i^{n+1} e f_i^n em torno do ponto de grade (i, n + 1/2), tal que:

$$\left.\frac{\partial f}{\partial t}\right|_i^{n+1/2} \simeq \frac{f_i^{n+1} - f_i^n}{\Delta t} \tag{2.89}$$

com o termo restante de erro de 2ª ordem dado por:

$$O[(\Delta t)^2] = -\frac{1}{24}(\Delta t)^2 \left.\frac{\partial^3 f}{\partial t^3}\right|_i^{n+1/2} \tag{2.90}$$

A equação (2.89) é conhecida como diferença direta de 2ª ordem.

As aproximações para as derivadas no espaço podem ser obtidas da mesma maneira que as das derivadas no tempo, expandindo a função em séries de Taylor para a variável x. Aqui é de interesse obter a aproximação de diferença direta de 2ª ordem de $[\partial f / \partial t]_{i+1/2}^n$. Esta é obtida a partir das expansões em séries de Taylor de f_{i+1}^n

e f_i^n em torno do ponto de grade intermediário (i + 1/2, n), como mostrado nas equações a seguir:

$$f_{i+1}^n = f_{i+1/2}^n + \left(\frac{\Delta x}{2}\right)\frac{\partial f}{\partial x}\Big|_{i+1/2}^n + \frac{1}{2}\left(\frac{\Delta x}{2}\right)^2\frac{\partial^2 f}{\partial x^2}\Big|_{i+1/2}^n + \cdots + \frac{1}{n!}\left(\frac{\Delta x}{2}\right)^n\frac{\partial^n f}{\partial x^n}\Big|_{i+1/2}^n + \cdots \quad (2.91)$$

onde $\Delta x/2 = x_{i+1} - x_{i+1/2}$. E com:

$$f_i^n = f_{i+1/2}^n + \left(\frac{-\Delta x}{2}\right)\frac{\partial f}{\partial x}\Big|_{i+1/2}^n + \frac{1}{2}\left(\frac{-\Delta x}{2}\right)^2\frac{\partial^2 f}{\partial x^2}\Big|_{i+1/2}^n + \cdots + \frac{1}{n!}\left(\frac{-\Delta x}{2}\right)^n\frac{\partial^n f}{\partial x^n}\Big|_{i+1/2}^n + \cdots \quad (2.92)$$

onde $-\Delta x/2 = x_i - x_{i+1/2}$. Logo, subtraindo a equação (2.92) da equação (2.91), obtém-se a seguinte equação de diferença finita aproximada como:

$$\frac{\partial f}{\partial x}\Big|_{i+1/2}^n \simeq \frac{f_{i+1}^n - f_i^n}{\Delta x} \quad (2.93)$$

O termo restante de erro de 2ª ordem é dado por:

$$O[(\Delta x)^2] = -\frac{1}{24}(\Delta x)^2\frac{\partial^3 f}{\partial x^3}\Big|_{i+1/2}^n \quad (2.94)$$

Os termos com potência maior que dois de Δx são ignorados na equação (2.94).

As equações (2.89) e (2.93) são as equações de diferença finita direta com precisão de 2ª ordem para derivadas no tempo e espaço, respectivamente. Ambas serão usadas na formulação do método FDTD Yee para as equações de Maxwell, a ser analisado a seguir.

2.3.2 Método FDTD Yee

As equações de Maxwell de interesse no processo de discretização do método FDTD Yee são obtidas a partir das equações (2.27) e (2.29), e usando as relações constitutivas de (2.32) e (2.33), como:

$$\nabla \times \boldsymbol{E} = -\mu\frac{\partial \boldsymbol{H}}{\partial t} \quad (2.95)$$

$$\nabla \times \boldsymbol{H} = \varepsilon \frac{\partial \boldsymbol{E}}{\partial t} \tag{2.96}$$

O meio, por simplicidade, é considerado homogêneo, isotrópico, linear, não variante no tempo e não dispersivo, sem perdas elétricas ($\sigma = 0$), livre de cargas elétricas ($\rho_v = 0$) ou fontes de corrente elétrica ($\boldsymbol{J_i} = 0$). Dessa forma, a permissividade elétrica (ε) e a permeabilidade magnética (μ) são consideradas constantes em todo o espaço de simulação. Em coordenadas cartesianas, aplicando o rotacional da equação (2.11) no lado esquerdo das equações (2.95) e (2.96) e rearranjando os termos obtém-se, para o caso tridimensional geral, as seguintes equações:

$$\frac{\partial H_X}{\partial t} = \frac{1}{\mu}\left(\frac{\partial E_Y}{\partial z} - \frac{\partial E_Z}{\partial y}\right) \tag{2.97a}$$

$$\frac{\partial H_Y}{\partial t} = \frac{1}{\mu}\left(\frac{\partial E_Z}{\partial x} - \frac{\partial E_X}{\partial z}\right) \tag{2.97b}$$

$$\frac{\partial H_Z}{\partial t} = \frac{1}{\mu}\left(\frac{\partial E_X}{\partial y} - \frac{\partial E_Y}{\partial x}\right) \tag{2.97c}$$

$$\frac{\partial E_X}{\partial t} = \frac{1}{\varepsilon}\left(\frac{\partial H_Z}{\partial y} - \frac{\partial H_Y}{\partial z}\right) \tag{2.97d}$$

$$\frac{\partial E_Y}{\partial t} = \frac{1}{\varepsilon}\left(\frac{\partial H_X}{\partial z} - \frac{\partial H_Z}{\partial x}\right) \tag{2.97e}$$

$$\frac{\partial E_Z}{\partial t} = \frac{1}{\varepsilon}\left(\frac{\partial H_Y}{\partial x} - \frac{\partial H_X}{\partial y}\right) \tag{2.97f}$$

As derivadas no tempo de (2.95) e (2.96) são aproximadas com diferença finita direta de 2ª ordem como definido pela equação (2.89), mas as atualizações dos campos elétrico e magnético são deslocadas por meio passo de tempo. Assim, na forma vetorial as equações de Maxwell (2.95) e (2.96) ficam aproximadas como:

$$\left.\frac{\partial \boldsymbol{H}}{\partial t}\right|^n \simeq \frac{\boldsymbol{H}^{n+\frac{1}{2}} - \boldsymbol{H}^{n-\frac{1}{2}}}{\Delta t} = -\frac{1}{\mu}\left[\nabla \times \boldsymbol{E}\right]^n \tag{2.98}$$

$$\left.\frac{\partial \boldsymbol{E}}{\partial t}\right|^{n+\frac{1}{2}} \simeq \frac{\boldsymbol{E}^{n+1} - \boldsymbol{E}^{n}}{\Delta t} = \frac{1}{\varepsilon}[\boldsymbol{\nabla} \times \boldsymbol{H}]^{n+\frac{1}{2}} \tag{2.99}$$

Rearranjando os termos nas equações (2.98) e (2.99), o processo de marcha no tempo fica definido como:

$$\boldsymbol{H}^{n+\frac{1}{2}} = \boldsymbol{H}^{n-\frac{1}{2}} - \frac{\Delta \mathrm{t}}{\mu}[\boldsymbol{\nabla} \times \boldsymbol{E}]^{n} \tag{2.100}$$

$$\boldsymbol{E}^{n+1} = \boldsymbol{E}^{n} + \frac{\Delta \mathrm{t}}{\varepsilon}[\boldsymbol{\nabla} \times \boldsymbol{H}]^{n+\frac{1}{2}} \tag{2.101}$$

A atualização no tempo dos campos elétrico e magnético nas equações (2.100) e (2.101) é chamada passo de sapo, mais conhecido em inglês como *leapfrog*. O processo de discretização no espaço é também aproximado por diferença finita direta de 2ª ordem como definido na equação (2.93), e será aplicado aos 2º termos à direita das equações (2.100) e (2.101). O problema a ser analisado pode ser unidimensional (1D), bidimensional (2D) ou tridimensional (3D); assim, simplificações poderão ocorrer nas equações (2.100) e (2.101) para problemas 1D ou 2D.

2.3.3 Equações do método FDTD Yee em uma dimensão

Para problemas 1D os campos elétrico e magnético apresentam variações em unicamente uma direção. Escolhendo, por exemplo, a direção no eixo x como de interesse, considera-se que, nas direções y e z, $\partial/\partial y = \partial/\partial z = 0$. Aplicando-se estes valores de derivada nas equações de (2.97a) a (2.97f) obtém-se dois conjuntos de equações 1D independentes; um para o modo TM (Transversal Magnético) definido como:

$$\frac{\partial H_Y}{\partial t} = \frac{1}{\mu}\frac{\partial E_Z}{\partial x} \tag{2.102a}$$

$$\frac{\partial E_Z}{\partial t} = \frac{1}{\varepsilon}\frac{\partial H_Y}{\partial x} \tag{2.102b}$$

e outro para o modo TE (Transversal Elétrico), definido como:

$$\frac{\partial H_Z}{\partial t} = -\frac{1}{\mu}\frac{\partial E_Y}{\partial x} \tag{2.103a}$$

$$\frac{\partial E_Y}{\partial t} = -\frac{1}{\varepsilon}\frac{\partial H_Z}{\partial x} \tag{2.103b}$$

É importante observar que o conceito de modos TM e TE usado aqui é diferente daquele usado em análise clássica de guias de onda. Tal conceito se tornará claro ao se analisar as equações de diferença finita em duas dimensões. Assim, por exemplo, o processo de discretização para o modo TE usando as equações (2.89) e (2.93) em (2.103) torna-se:

$$H_Z|_{i+1/2}^{n+1/2} = H_Z|_{i+1/2}^{n-1/2} - \frac{\Delta t}{\mu_{i+1/2}\Delta x}\left(E_Y|_{i+1}^{n} - E_Y|_{i}^{n}\right) \tag{2.104a}$$

$$E_Y|_{i}^{n+1} = E_Y|_{i}^{n} - \frac{\Delta t}{\varepsilon_i \Delta x}\left(H_Z|_{i+1/2}^{n+1/2} - H_Z|_{i-1/2}^{n+1/2}\right) \tag{2.104b}$$

Observa-se que, também, há um deslocamento de meio passo discreto no espaço no processo de discretização espacial. Estes deslocamentos de meio passo discreto no tempo e espaço entre campos elétricos e magnéticos aumentam a precisão do método FDTD, como proposto por K. S. Yee (YEE,1966).

2.3.4 Equações do método FDTD Yee em duas dimensões

Para problemas 2D os campos elétrico e magnético apresentam variações em duas dimensões. Escolhendo, por exemplo, as direções *x* e *y* como de interesse, e considerando que na direção *z* a derivada $\partial/\partial z = 0$ é aplicada nas equações de (2.97a) a (2.97f), obtém-se dois conjuntos de equações 2D independentes; um para o modo TM (ou TM_Z) definido como:

$$\frac{\partial H_X}{\partial t} = -\frac{1}{\mu}\frac{\partial E_Z}{\partial y} \tag{2.105a}$$

$$\frac{\partial H_Y}{\partial t} = \frac{1}{\mu}\frac{\partial E_Z}{\partial x} \tag{2.105b}$$

$$\frac{\partial E_Z}{\partial t} = \frac{1}{\varepsilon}\left(\frac{\partial H_Y}{\partial x} - \frac{\partial H_X}{\partial y}\right) \tag{2.105c}$$

O outro conjunto de equações para o modo TE (ou TE_Z) é dado por:

$$\frac{\partial E_X}{\partial t} = \frac{1}{\varepsilon}\frac{\partial H_Z}{\partial y} \tag{2.106a}$$

$$\frac{\partial E_Y}{\partial t} = -\frac{1}{\varepsilon}\frac{\partial H_Z}{\partial x} \tag{2.106b}$$

$$\frac{\partial H_Z}{\partial t} = \frac{1}{\mu}\left(\frac{\partial E_X}{\partial y} - \frac{\partial E_Y}{\partial x}\right) \tag{2.106c}$$

Em teoria eletromagnética de guias de onda, a direção *z* (eixo do guia) é considerada a direção de propagação dos campos elétrico e magnético. Assim o modo TM é aquele que não possui componente do campo magnético na direção de propagação (eixo *z*); e da mesma maneira o modo TE é aquele que não possui componente do campo elétrico na direção de propagação (eixo *z*). No entanto, os modos TM e TE para o método FDTD são definidos de outra forma. Neste caso específico (equações (2.105) e (2.106)) não há variações dos campos elétrico ou magnético na direção *z*. Assim, o modo TM (ou TM_Z) implica que unicamente as componentes de campo H_X e H_Y são diferentes de zero; e o modo TE (ou TE_Z) implica que unicamente as componentes de campo E_X e E_Y são diferentes de zero. Logicamente, outros modos podem ser obtidos no sistema cartesiano, tais como TM_X, TM_Y, TE_X ou TE_Y.

Aplicando as equações de diferença finita (2.89) e (2.93) nas equações (2.105) do modo TM_Z, com o devido cuidado de considerar a discretização numa grade 2D com índices *i* e *j*, obtém-se:

$$H_X|_{i,j+1/2}^{n+1/2} = H_X|_{i,j+1/2}^{n-1/2} - \frac{\Delta t}{\mu_{i,j+1/2}\Delta y}\left(E_Z|_{i,j+1}^{n} - E_Z|_{i,j}^{n}\right) \tag{2.107a}$$

$$H_Y|_{i+1/2,j}^{n+1/2} = H_Y|_{i+1/2,j}^{n-1/2} + \frac{\Delta t}{\mu_{i+1/2,j}\Delta x}\left(E_Z|_{i+1,j}^{n} - E_Z|_{i,j}^{n}\right) \tag{2.107b}$$

$$E_Z|_{i,j}^{n+1} = E_Z|_{i,j}^{n} + \frac{\Delta t}{\varepsilon_{i,j}}\left(\frac{H_Y|_{i+1/2,j}^{n+1/2} - H_Y|_{i-1/2,j}^{n+1/2}}{\Delta x} - \frac{H_X|_{i,j+1/2}^{n+1/2} - H_X|_{i,j-1/2}^{n+1/2}}{\Delta y}\right) \quad (2.107c)$$

2.3.5 Condição de estabilidade do método FDTD Yee

Neste ponto é interessante analisar a condição de estabilidade do método FDTD. Esta condição é determinada pela escolha do máximo incremento de tempo (Δt) em relação a um incremento de espaço mínimo (Δw), e está relacionada à velocidade de fase (c) real da onda eletromagnética em um meio, como:

$$c.\Delta t \leq \Delta w \quad ou \quad \Delta t \leq \frac{\Delta w}{c} \quad (2.108)$$

A velocidade de fase real da onda eletromagnética é aquela definida na equação (2.76). O valor de Δw para o caso geral tridimensional, especifico ao método FDTD Yee (INAN; MARSHALL, 2011), com hexaedros é dado por:

$$\Delta w = \frac{1}{\sqrt{\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2} + \frac{1}{(\Delta z)^2}}} \quad (2.109)$$

Supondo as seguintes relações:

$$R_Y = \frac{\Delta x}{\Delta y} \quad (2.110a)$$

$$R_Z = \frac{\Delta x}{\Delta z} \quad (2.110b)$$

Aplicando as equações (2.110a) e (2.110b) na equação (2.109), obtém-se:

$$\Delta w = \frac{\Delta x}{\sqrt{1 + {R_Y}^2 + {R_Z}^2}} \quad (2.111)$$

Agora é possível definir um coeficiente de estabilidade, conhecido como número de

Courant-Friedrichs-Lewy, ou simplesmente número de Courant (S_{CFL}), aplicando a equação (2.111) na equação (2.108), como:

$$S_{CFL} = \frac{c.\Delta t}{\Delta x} \leq \frac{1}{\sqrt{1 + {R_Y}^2 + {R_Z}^2}} \tag{2.112}$$

Para células cúbicas onde $\Delta y = \Delta z = \Delta x$, tem-se a partir da equação (2.112): $S_{CFL} \leq 1/\sqrt{3}$. Para o caso 1D, com variações dos campos apenas na direção *x*, como, por exemplo, o modo TE 1D definido pela equação (2.104), com $R_Y = R_Z = 0$, tem-se o seguinte valor para o número de Courant:

$$S_{CFL} = \frac{c.\Delta t}{\Delta x} \leq 1 \tag{2.113}$$

Para o caso 2D, com variações dos campos nas direções *x* e *y*, como o modo TM_Z 2D definido pela equação (2.107), com $R_Z = 0$, tem-se:

$$S_{CFL} = \frac{c.\Delta t}{\Delta x} \leq \frac{1}{\sqrt{1 + {R_Y}^2}} \tag{2.114}$$

Para $\Delta y = \Delta x$, tem-se: $S_{CFL} \leq 1/\sqrt{2}$.

É importante lembrar que a velocidade de fase numérica (v_{pn}), ou seja, numa grade 2D ou 3D, em geral não é igual à velocidade de fase (*c*) da onda eletromagnética em um meio real (físico). Uma única exceção a esta regra ocorre para a velocidade de fase numérica na direção da diagonal principal da grade 2D ou 3D, quando se usa o número de Courant máximo permitido pelas equações (2.114) ou (2.112), respectivamente. Uma outra exceção a esta regra ocorre no caso 1D, quando se usa número de Courant máximo: $S_{CFL} = 1$, e assim:

$$v_{pn} = \frac{\Delta x}{\Delta t} = c \tag{2.115}$$

Neste caso o processo de discretização não introduz erro. Mas, para os casos 2D e 3D, sempre serão introduzidos erros inerentes ao processo de discretização, e a onda numérica nestas grades (2D ou 3D) sofrerá dispersão numérica, que produzirá erros

de fase cada vez maiores à medida que a onda na grade se distancia da sua fonte de origem. Ocorrerá também uma anisotropia de velocidade numérica, em que o valor da velocidade de fase numérica é função da direção de propagação da onda na grade.

No método FDTD Yee uma forma de reduzir esta anisotropia numérica é reduzir o tamanho do incremento espacial mínimo (Δ*w*) definido pela equação (2.111), mas isto reduz também o incremento de tempo (Δ*t*), como definido pela equação (2.108), para que a estabilidade numérica do processo de marcha no tempo seja mantida; a consequência dessas reduções é aumentar a quantidade de memória necessária e também aumentar o tempo de simulação. Estes recursos computacionais (memória e tempo de simulação) serão mais críticos e custosos, quando simulando estruturas eletricamente grandes, isto é, cujas dimensões são muito maiores que o comprimento de onda mínimo presente na onda numérica que se propaga na grade.

2.3.6 Uma nova formulação dos coeficientes nas equações de diferença finita

É conveniente para o processo de simulação com o método FDTD Yee, alterar a forma de entrada dos coeficientes que multiplicam os valores de campos elétrico e magnético (SCHNEIDER, 2013). Sabe-se que, em um meio simples, a velocidade de fase (*c*) e a impedância característica do meio (*Z*) são definidas como:

$$c = \frac{1}{\sqrt{\mu\varepsilon}} \tag{2.116}$$

$$Z = \sqrt{\frac{\mu}{\varepsilon}} \tag{2.117}$$

O termo presente, por exemplo, na equação (2.104b) será transformado usando as equações (2.112), (2.116) e (2.117), de forma que:

$$\frac{1}{\varepsilon}\frac{\Delta t}{\Delta x} = \frac{1}{\varepsilon}\frac{\sqrt{\mu\varepsilon}}{\sqrt{\mu\varepsilon}}\frac{\Delta t}{\Delta x} = \frac{\sqrt{\mu\varepsilon}}{\varepsilon}\frac{\mathrm{c}.\Delta t}{\Delta x} = \sqrt{\frac{\mu}{\varepsilon}}\frac{\mathrm{c}.\Delta t}{\Delta x} = S_{CFL}.Z \tag{2.118a}$$

o que resulta no coeficiente C_{EX}:

$$C_{EX} = S_{CFL}.Z \tag{2.118b}$$

O termo presente, por exemplo, na equação (2.104a) também será transformado usando as equações (2.112), (2.116) e (2.117), de forma que:

$$\frac{1}{\mu}\frac{\Delta t}{\Delta x} = \frac{1}{\mu}\frac{\sqrt{\mu\varepsilon}}{\sqrt{\mu\varepsilon}}\frac{\Delta t}{\Delta x} = \frac{\sqrt{\mu\varepsilon}}{\mu}\frac{\mathrm{c}.\Delta t}{\Delta x} = \sqrt{\frac{\varepsilon}{\mu}}\frac{\mathrm{c}.\Delta t}{\Delta x} = \frac{S_{CFL}}{Z} \tag{2.119a}$$

o que resulta no coeficiente C_{HX}:

$$C_{HX} = \frac{S_{CFL}}{Z} \tag{2.119b}$$

Raciocínio semelhante aplica-se para coeficientes com Δ*y* ou Δ*z* no denominador, mas usando também as equações (2.110a) e (2.110b), respectivamente, de forma que:

$$C_{EY} = \frac{1}{\varepsilon}\frac{\Delta t}{\Delta y} = R_Y\, S_{CFL}.Z \tag{2.120}$$

$$C_{HY} = \frac{1}{\mu}\frac{\Delta t}{\Delta y} = R_Y\frac{S_{CFL}}{Z} \tag{2.121}$$

$$C_{EZ} = \frac{1}{\varepsilon}\frac{\Delta t}{\Delta z} = R_Z\, S_{CFL}.Z \tag{2.122}$$

$$C_{HZ} = \frac{1}{\mu}\frac{\Delta t}{\Delta z} = R_Z\frac{S_{CFL}}{Z} \tag{2.123}$$

Desta forma, aplicando os coeficientes necessários, dos deduzidos nas equações de (2.118) a (2.123), para o modo TM$_Z$ 2D da equação (2.107), obtém-se:

$$H_X|_{i,j+1/2}^{n+1/2} = C_{HX0}H_X|_{i,j+1/2}^{n-1/2} - C_{HY}\left(E_Z|_{i,j+1}^{n} - E_Z|_{i,j}^{n}\right) \tag{2.124a}$$

$$H_Y|_{i+1/2,j}^{n+1/2} = C_{HY0}H_Y|_{i+1/2,j}^{n-1/2} + C_{HX}\left(E_Z|_{i+1,j}^{n} - E_Z|_{i,j}^{n}\right) \tag{2.124b}$$

$$E_Z|_{i,j}^{n+1} = C_{EZ0}E_Z|_{i,j}^{n} + C_{EX}\left(H_Y|_{i+1/2,j}^{n+1/2} - H_Y|_{i-1/2,j}^{n+1/2}\right) - C_{EY}\left(H_X|_{i,j+1/2}^{n+1/2} - H_X|_{i,j-1/2}^{n+1/2}\right) \quad (2.124c)$$

Os novos coeficientes $C_{HX0} = C_{HY0} = C_{EZ0} = 1$ foram introduzidos para permitir o uso das equações de diferença finita com materiais que possuam perdas elétricas ou magnéticas, sem precisar alterar a forma matemática das equações em si; é necessário apenas alterar os valores de todos os coeficientes nas equações de diferença finita (ver subseção 2.3.8). Estruturas que possuam diferentes materiais, e/ou materiais não-homogêneos, podem ser representadas no espaço de simulação, apenas alterando os valores dos coeficientes em cada posição na grade. Isto é facilmente realizado criando uma matriz 2D ou 3D para cada um dos coeficientes utilizados nas equações de diferença finita.

2.3.7 Equações do método FDTD Yee em três dimensões

As equações de diferença finita, para o caso tridimensional, podem ser obtidas aplicando a mesma aproximação das equações de diferença finita para tempo (2.89) e espaço (2.93) nas equações diferenciais parciais de (2.97a) à (2.97f), com o devido cuidado de considerar a discretização numa grade 3D com índices *i*, *j* e *k* correspondendo às direções *x*, *y* e *z*, respectivamente. Utilizar-se-á também os coeficientes deduzidos nas equações (2.118) a (2.123), assim obtendo o seguinte conjunto de equações:

$$E_X|_{i+\frac{1}{2},j,k}^{n+1} = C_{EX0}E_X|_{i+\frac{1}{2},j,k}^{n} + C_{EY}\left(H_Z|_{i+\frac{1}{2},j+\frac{1}{2},k}^{n+1/2} - H_Z|_{i+\frac{1}{2},j-\frac{1}{2},k}^{n+1/2}\right) - C_{EZ}\left(H_Y|_{i+\frac{1}{2},j,k+\frac{1}{2}}^{n+1/2} - H_Y|_{i+\frac{1}{2},j,k-\frac{1}{2}}^{n+1/2}\right) \quad (2.125a)$$

$$E_Y|_{i,j+\frac{1}{2},k}^{n+1} = C_{EY0}E_Y|_{i,j+\frac{1}{2},k}^{n} + C_{EZ}\left(H_X|_{i,j+\frac{1}{2},k+\frac{1}{2}}^{n+1/2} - H_X|_{i,j+\frac{1}{2},k-\frac{1}{2}}^{n+1/2}\right) - C_{EX}\left(H_Z|_{i+\frac{1}{2},j+\frac{1}{2},k}^{n+1/2} - H_Z|_{i-\frac{1}{2},j+\frac{1}{2},k}^{n+1/2}\right) \quad (2.125b)$$

$$E_Z|_{i,j,k+\frac{1}{2}}^{n+1} = C_{EZ0}E_Z|_{i,j,k+\frac{1}{2}}^{n} + C_{EX}\left(H_Y|_{i+\frac{1}{2},j,k+\frac{1}{2}}^{n+1/2} - H_Y|_{i-\frac{1}{2},j,k+\frac{1}{2}}^{n+1/2}\right) - C_{EY}\left(H_X|_{i,j+\frac{1}{2},k+\frac{1}{2}}^{n+1/2} - H_X|_{i,j-\frac{1}{2},k+\frac{1}{2}}^{n+1/2}\right) \quad (2.125c)$$

$$H_X|_{i,j+\frac{1}{2},k+\frac{1}{2}}^{n+1/2} = C_{HX0}H_X|_{i,j+\frac{1}{2},k+\frac{1}{2}}^{n-1/2} + C_{HZ}\left(E_Y|_{i,j+\frac{1}{2},k+1}^{n} - E_Y|_{i,j+\frac{1}{2},k}^{n}\right) - C_{HY}\left(E_Z|_{i,j+1,k+\frac{1}{2}}^{n} - E_Z|_{i,j,k+\frac{1}{2}}^{n}\right) \quad (2.125d)$$

$$H_Y|_{i+\frac{1}{2},j,k+\frac{1}{2}}^{n+1/2} = C_{HY0}H_Y|_{i+\frac{1}{2},j,k+\frac{1}{2}}^{n-1/2} + C_{HX}\left(E_Z|_{i+1,j,k+\frac{1}{2}}^{n} - E_Z|_{i,j,k+\frac{1}{2}}^{n}\right) - C_{HZ}\left(E_X|_{i+\frac{1}{2},j,k+1}^{n} - E_X|_{i+\frac{1}{2},j,k}^{n}\right) \quad (2.125e)$$

$$H_Z|_{i+\frac{1}{2},j+\frac{1}{2},k}^{n+1/2} = C_{HZ0}H_Z|_{i+\frac{1}{2},j+\frac{1}{2},k}^{n-1/2} + C_{HY}\left(E_X|_{i+\frac{1}{2},j+1,k}^{n} - E_X|_{i+\frac{1}{2},j,k}^{n}\right) - C_{HX}\left(E_Y|_{i+1,j+\frac{1}{2},k}^{n} - E_Y|_{i,j+\frac{1}{2},k}^{n}\right) \quad (2.125f)$$

Os coeficientes para um meio simples, homogêneo e sem perdas são definidos como: $C_{EX0} = C_{EY0} = C_{EZ0} = C_{HX0} = C_{HY0} = C_{HZ0} = 1$. O método FDTD Yee para o caso tridimensional pode ser entendido como a superposição e acoplamento dos modos TMz e TEz (SCHNEIDER, 2013), como mostrado na Figura 2.2.

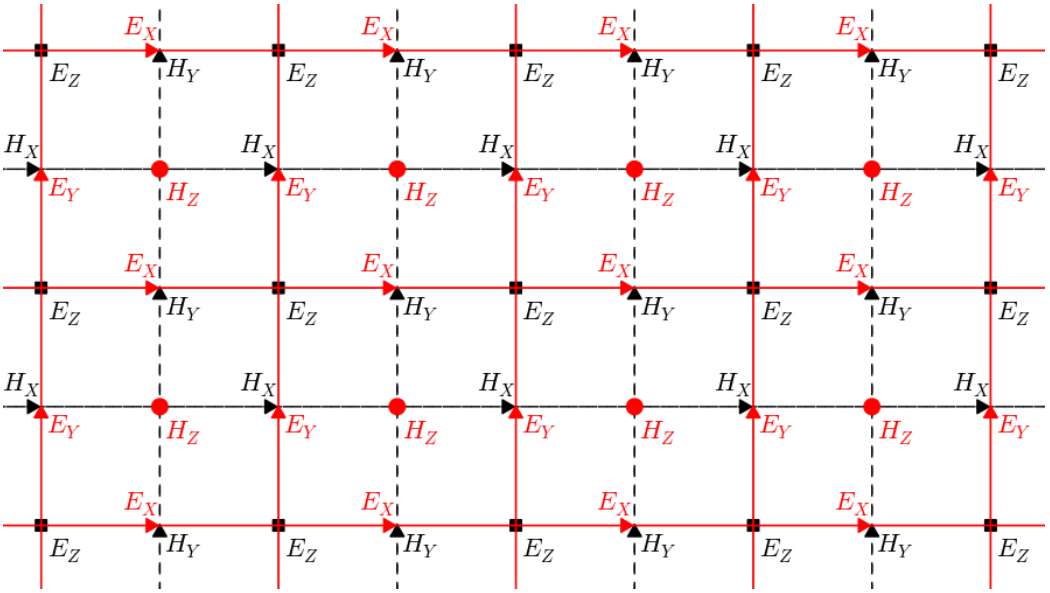


Figura 2.2 - Superposição de grades para modos TEz e TMz no método FDTD Yee.

Na Figura 2.2 o modo TM_Z tem campos magnéticos H_X e H_Y no plano x-y e o campo elétrico E_Z na direção positiva do eixo z. O modo TE_Z tem campos elétricos E_X e E_Y no plano x-y e o campo magnético H_Z na direção positiva do eixo z. É necessário acoplar os modos TM_Z e TE_Z já analisados nas equações diferenciais parciais (2.105) e (2.106), respectivamente, de forma a obter as equações diferenciais parciais (2.97a) a (2.97f) para o caso tridimensional. Este conceito será útil na formulação do método FDTD para a grade de prismas hexagonais.

2.3.8 Coeficientes para meios com perdas e não-homogêneos

Transformando as equações (2.45) e (2.47) para o domínio do tempo e aplicando o teorema de Stokes da equação (2.13), mas ignorando as densidades de correntes magnética ($\boldsymbol{M_i}$) e elétrica ($\boldsymbol{J_i}$), é obtido para meio linear, isotrópico, invariante no tempo, mas com perdas e não-homogêneo, o seguinte conjunto de equações de Maxwell nas formas diferencial e integral:

$$\oint_C \boldsymbol{E}.\boldsymbol{dl} = -\int_S \frac{\partial \mu \boldsymbol{H}}{\partial t}\boldsymbol{ds} - \int_S \sigma_m \boldsymbol{H}.\boldsymbol{ds} \tag{2.126a}$$

$$\nabla \times \boldsymbol{E} = -\mu\frac{\partial \boldsymbol{H}}{\partial t} - \sigma_m \boldsymbol{H} \tag{2.126b}$$

$$\oint_C \boldsymbol{H}.\boldsymbol{dl} = \int_S \frac{\partial \varepsilon \boldsymbol{E}}{\partial t}\boldsymbol{ds} + \int_S \sigma \boldsymbol{E}.\boldsymbol{ds} \tag{2.126c}$$

$$\nabla \times \boldsymbol{H} = \varepsilon\frac{\partial \boldsymbol{E}}{\partial t} + \sigma \boldsymbol{E} \tag{2.126d}$$

Por exemplo, aplicando a equação (2.126b) para o campo H_X do modo TM_Z da grade Yee 2D obtém-se uma equação semelhante à equação (2.107a), mas agora incluindo as perdas do meio:

$$\left.\frac{\partial \mu H_X}{\partial t}\right|_{i,j+1/2}^{n} + \sigma_m H_X|_{i,j+1/2}^{n} = -\frac{1}{\Delta y}\left(E_Z|_{i,j+1}^{n} - E_Z|_{i,j}^{n}\right) \tag{2.127}$$

A componente de campo H_X no segundo termo do lado esquerdo da equação (2.127), pode ser estimada com precisão de 2ª ordem usando média de dois pontos no tempo tal que:

$$H_X|_{i,j+1/2}^{n} = \frac{H_X|_{i,j+1/2}^{n+1/2} + H_X|_{i,j+1/2}^{n-1/2}}{2} \tag{2.128}$$

Substituindo a equação (2.128) na equação (2.127), aplicando diferenças finitas no tempo na equação (2.127) e rearranjando os termos, é encontrado:

$$H_X|_{i,j+1/2}^{n+1/2} = \left[\frac{2\mu - \sigma_m \Delta t}{2\mu + \sigma_m \Delta t}\right] . H_X|_{i,j+1/2}^{n-1/2} - \frac{2\Delta t}{(2\mu + \sigma_m \Delta t).\Delta y}\left(E_Z|_{i,j+1}^{n} - E_Z|_{i,j}^{n}\right) \tag{2.129}$$

O mesmo procedimento usado para obter a equação (2.129) pode ser aplicado nas equações (2.125a) a (2.125f) da grade 3D de hexaedros do método FDTD Yee. Consequentemente os coeficientes nestas equações, incorporando as perdas do meio, são alterados para:

$$C_{EX0} = C_{EY0} = C_{EZ0} = \frac{2\varepsilon - \sigma \Delta t}{2\varepsilon + \sigma \Delta t} \tag{2.130a}$$

$$C_{HX0} = C_{HY0} = C_{HZ0} = \frac{2\mu - \sigma_m \Delta t}{2\mu + \sigma_m \Delta t} \tag{2.130b}$$

$$C_{EX} = \frac{2\Delta t}{(2\varepsilon + \sigma \Delta t).\Delta x} \tag{2.130c}$$

$$C_{EY} = \frac{2\Delta t}{(2\varepsilon + \sigma \Delta t).\Delta y} = \frac{2R_Y.\Delta t}{(2\varepsilon + \sigma \Delta t).\Delta \mathrm{x}} \tag{2.130d}$$

$$C_{EZ} = \frac{2\Delta t}{(2\varepsilon + \sigma \Delta t).\Delta \mathrm{z}} = \frac{2R_Z.\Delta t}{(2\varepsilon + \sigma \Delta t).\Delta \mathrm{x}} \tag{2.130e}$$

$$C_{HX} = \frac{2\Delta t}{(2\mu + \sigma_m \Delta t).\Delta x} \tag{2.130f}$$

$$C_{HY} = \frac{2\Delta t}{(2\mu + \sigma_m \Delta t).\Delta \mathrm{y}} = \frac{2R_Y \Delta t}{(2\mu + \sigma_m \Delta t).\Delta \mathrm{x}} \tag{2.130g}$$

$$C_{HZ} = \frac{2\Delta t}{(2\mu + \sigma_m \Delta t).\Delta z} = \frac{2R_Z \Delta t}{(2\mu + \sigma_m \Delta t).\Delta x} \quad (2.130h)$$

Os coeficientes das equações (2.130a) a (2.130h) podem ser definidos em cada ponto da grade 3D; logo, é possível simular meios não-homogêneos. As razões R_Y e R_Z são definidas pelas equações (2.110a) e (2.110b), respectivamente. O tamanho do passo de tempo Δt é calculado a partir da equação (2.112) como:

$$\Delta t \leq \frac{S_{CFL}.\Delta x}{c_0} \quad (2.131)$$

onde c_0 é a velocidade física da onda no vácuo ($c_0 \approx 3\ 10^8$ m/s) e o número de Courant (S_{CFL}) é obtido da equação (2.112).

3 FORMULAÇÃO DO MÉTODO FDTD APLICADO À GRADE FORMADA POR PRISMAS HEXAGONAIS

Como já analisado no capítulo 2, o método FDTD Yee, com células hexaédricas, apresenta dispersão numérica e anisotropia de velocidade de fase numérica inerente ao processo de discretização. Muita pesquisa tem sido feita, no sentido de reduzir a anisotropia numérica e, consequentemente, a dispersão numérica (LIU, 1996; SCHNEIDER; KRUBLAK, 2001; SHLAGER; SCHNEIDER, 2005; TAFLOVE; HAGNESS, 2005; HAMILTON; BILBAO, 2013; SESCU; HIXON, 2014). Nesta seção far-se-á a dedução das equações de diferença finita para a grade tridimensional formada por prismas hexagonais, a partir da formulação bidimensional de hexágonos como dado em (LIU, 1996) ou (TAFLOVE; HAGNESS, 2005), e a validação desta nova formulação, comparando-a com o método FDTD Yee.

3.1 FORMULAÇÃO BIDIMENSIONAL DE HEXÁGONOS

É conhecido, para simulações bidimensionais com o método FDTD, que a anisotropia numérica em uma grade formada por hexágonos é da ordem de centenas de vezes menor que aquela da grade retangular bidimensional do método FDTD Yee (LIU, 1996; FEI; XIAOHONG; XIANJING, 2005). Nesta formulação utiliza-se uma grade dual, formada por duas distintas grades hexagonais. A grade primária, formada pelos hexágonos maiores, e a grade secundária, formada pelos hexágonos menores, conforme a Figura 3.1.

Antes de mostrar as equações de diferença finita para esta grade bidimensional de hexágonos, é útil analisar a geometria da Figura 3.1. A relação entre o lado do hexágono menor (Δb) e o lado do hexágono maior (Δd) é dada por:

$$\Delta b = \Delta d/\sqrt{3} \tag{3.1}$$

Na Figura 3.1 utiliza-se uma notação mais simples para definir a posição dos campos que a de (LIU, 1996). Nesta notação a distância entre *i* e *i+1* é Δx e entre *j* e *j+1* é Δy.

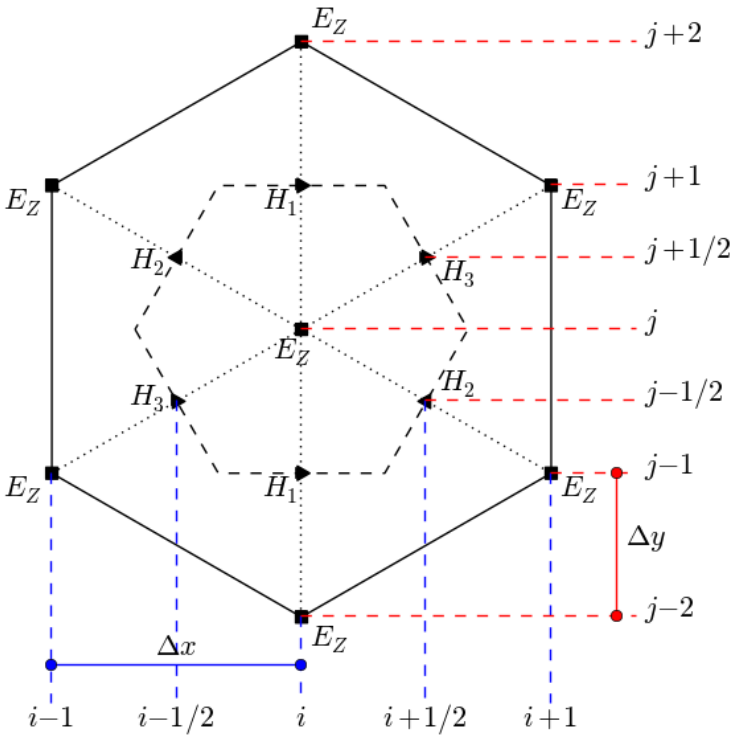


Figura 3.1 - Células primária e secundária de uma grade dual de hexágonos.

Estes incrementos de espaço são definidos em função do lado do hexágono maior (Δd) como:

$$\Delta x = \Delta d . \cos 30° = \frac{\sqrt{3}}{2} \Delta d \tag{3.2}$$

$$\Delta y = \frac{1}{2} \Delta d \tag{3.3}$$

A área do hexágono menor (A_H) pode ser definida como:

$$A_H = \frac{3\sqrt{3}}{2} \Delta b^2 = \frac{\sqrt{3}}{2} \Delta d^2 \tag{3.4}$$

As equações de diferença finita para grade bidimensional de hexágonos podem ser obtidas usando as formulações integral e diferencial das equações de Maxwell para um meio simples e homogêneo, como já definidas na seção 2.2, e repetidas aqui, por conveniência, como:

$$\int_C \boldsymbol{E}.\boldsymbol{dl} = -\oint_S \frac{\partial \mu \boldsymbol{H}}{\partial t}.\boldsymbol{ds} \quad (3.5a)$$

$$\nabla \times \boldsymbol{E} = -\mu \frac{\partial \boldsymbol{H}}{\partial t} \quad (3.5b)$$

$$\int_C \boldsymbol{H}.\boldsymbol{dl} = \oint_S \frac{\partial \varepsilon \boldsymbol{E}}{\partial t}.\boldsymbol{ds} \quad (3.6a)$$

$$\nabla \times \boldsymbol{H} = \varepsilon \frac{\partial \boldsymbol{E}}{\partial t} \quad (3.6b)$$

onde se tem as intensidades de campos elétrico (***E***) e magnético (***H***). O meio é considerado também sem perdas ($\sigma = 0$), e livre de cargas elétricas ($\rho_v = 0$) e fontes de corrente ($\boldsymbol{J_i} = 0$). Assim, em um meio simples, a velocidade de fase (c) e a impedância característica do meio (Z) são definidas como:

$$c = \frac{1}{\sqrt{\mu\varepsilon}} \quad (3.7)$$

$$Z = \sqrt{\frac{\mu}{\varepsilon}} \quad (3.8)$$

O número de Courant (S_{CFL}), como já analisado na subseção 2.3.5, determina a condição de estabilidade numérica, e é definido nesta seção em função do lado do hexágono maior (Δd) como:

$$S_{CFL} = \frac{c.\Delta t}{\Delta d} \leq \frac{1}{\sqrt{2}} \quad (3.9)$$

onde Δt é o incremento no tempo. Para a grade bidimensional de hexágonos o número de Courant máximo é $S_{CFL} = 1/\sqrt{2}$ (LIU, 1996).

Na Figura 3.1 tem-se o modo TM_Z com os campos magnéticos H_1, H_2 e H_3 no plano x-y e o campo elétrico E_Z na direção positiva do eixo z. Utilizando a equação de Maxwell na forma integral (3.6a), que é aplicada no hexágono menor da Figura 3.1, obtém-se para o campo E_Z, na posição (i, j), a seguinte equação:

$$\left.\frac{\partial \varepsilon E_z}{\partial t}\right|_{i,j}^{n+\frac{1}{2}} . A_H = \Delta b . \left(H_1|_{i,j-1}^{n+\frac{1}{2}} - H_1|_{i,j+1}^{n+\frac{1}{2}} + H_2|_{i+\frac{1}{2},j-\frac{1}{2}}^{n+\frac{1}{2}} - H_2|_{i-\frac{1}{2},j+\frac{1}{2}}^{n+\frac{1}{2}} + \right.$$
$$\left. + H_3|_{i+\frac{1}{2},j+\frac{1}{2}}^{n+\frac{1}{2}} - H_3|_{i-\frac{1}{2},j-\frac{1}{2}}^{n+\frac{1}{2}} \right) \quad (3.10)$$

onde a área do hexágono menor (A_H) já foi definida na equação (3.4) e é aplicada juntamente com as equações (3.1), (3.7), (3.8) e (3.9) na equação (3.10), em um procedimento similar àquele desenvolvido na seção 2.3.6, para obter a seguinte equação de diferença finita (com precisão de 2ª ordem no tempo e espaço) para o campo elétrico E_z:

$$E_Z|_{i,j}^{n+1} = E_Z|_{i,j}^{n} + C_{EZ}\left(H_1|_{i,j-1}^{n+\frac{1}{2}} - H_1|_{i,j+1}^{n+\frac{1}{2}} \right) + C_{EZ}\left(H_2|_{i+\frac{1}{2},j-\frac{1}{2}}^{n+\frac{1}{2}} - H_2|_{i-\frac{1}{2},j+\frac{1}{2}}^{n+\frac{1}{2}} \right) +$$
$$+ C_{EZ}\left(H_3|_{i+\frac{1}{2},j+\frac{1}{2}}^{n+\frac{1}{2}} - H_3|_{i-\frac{1}{2},j-\frac{1}{2}}^{n+\frac{1}{2}} \right) \quad (3.11)$$

O coeficiente C_{EZ} em (3.11) é definido como:

$$C_{EZ} = \frac{2}{3} S_{CFL} . Z \quad (3.12)$$

Para o campo magnético H_1 a partir da equação de Maxwell na forma diferencial (3.5b), obtém-se:

$$\left.\frac{\partial \mu H_1}{\partial t}\right|_{i,j-1}^{n} = \frac{1}{\Delta d}\left(E_Z|_{i,j-2}^{n} - E_Z|_{i,j}^{n} \right) \quad (3.13)$$

Aplicando as equações (3.7), (3.8) e (3.9) na equação (3.13), também em um procedimento similar àquele desenvolvido na seção 2.3.6, obtém-se uma equação de diferença finita (com precisão de 2ª ordem no tempo e espaço) para campo magnético H_1 como:

$$H_1|_{i,j-1}^{n+\frac{1}{2}} = H_1|_{i,j-1}^{n-\frac{1}{2}} + C_{HA}\left(E_Z|_{i,j-2}^{n} - E_Z|_{i,j}^{n} \right) \quad (3.14)$$

O coeficiente C_{HA} em (3.14) é definido como:

$$C_{HA} = \frac{S_{CFL}}{Z} \tag{3.15}$$

As equações de diferença finita para os campos magnéticos H_2 e H_3 são obtidas com o mesmo procedimento usado na obtenção da equação de diferença finita (3.14) do campo magnético H_1, de forma que se tem:

$$H_2\Big|_{i+\frac{1}{2},j-\frac{1}{2}}^{n+\frac{1}{2}} = H_2\Big|_{i+\frac{1}{2},j-\frac{1}{2}}^{n-\frac{1}{2}} + C_{HA}\left(E_Z|_{i+1,j-1}^{n} - E_Z|_{i,j}^{n}\right) \tag{3.16}$$

$$H_3\Big|_{i+\frac{1}{2},j+\frac{1}{2}}^{n+\frac{1}{2}} = H_3\Big|_{i+\frac{1}{2},j+\frac{1}{2}}^{n-\frac{1}{2}} + C_{HA}\left(E_Z|_{i+1,j+1}^{n} - E_Z|_{i,j}^{n}\right) \tag{3.17}$$

onde o coeficiente C_{HA} é aquele definido na equação (3.15). As equações (3.11), (3.14), (3.16) e (3.17) são semelhantes àquelas de (LIU, 1996), mas a dedução aqui é baseada em uma geometria dos hexágonos com um ângulo deslocado de 30° a partir do eixo x, no sentido anti-horário (ver Figura 3.1).

3.2 FORMULAÇÃO DO MÉTODO FDTD PARA GRADE FORMADA POR PRISMAS HEXAGONAIS

O método FDTD com células hexaédricas (ou cúbicas) pode ser definido como a superposição e acoplamento de duas grades bidimensionais, uma para o modo TM_Z e outra para o modo TE_Z, como já analisado na subseção 2.3.7. Utilizando este mesmo raciocínio para a grade hexagonal, ter-se-á duas grades hexagonais (modos TM_Z e TE_Z) superpostas no espaço, com a geometria mostrada na Figura 3.2. Na Figura 3.2, para o modo TE_Z tem-se os campos elétricos E_1, E_2 e E_3 no plano x-y e o campo magnético H_Z na direção positiva do eixo *z*. As grades de hexágonos nos modos TM_Z e TE_Z são empilhadas alternadamente na direção do eixo *z*, com separação igual à metade do incremento de espaço Δz.

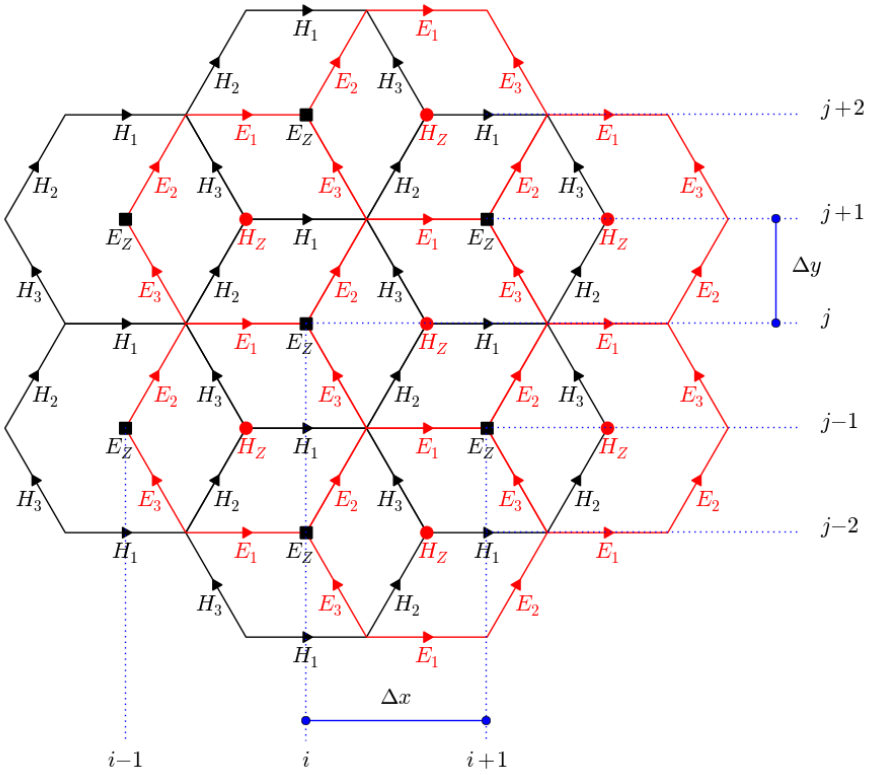


Figura 3.2 - Superposição de grades secundárias de hexágonos para os modos TM_Z e TE_Z.

Para o modo TM_Z o campo elétrico E_Z terá uma equação de diferença finita semelhante àquela da equação (3.11), mas adicionando o índice *k*, que corresponde aos incrementos de espaço (Δ*z*) na direção do eixo z, tal que:

$$E_Z|_{i,j,k}^{n+1} = C_{EZ0}.E_Z|_{i,j,k}^{n} + C_{EZ}\left(H_1|_{i,j-1,k}^{n+\frac{1}{2}} - H_1|_{i,j+1,k}^{n+\frac{1}{2}}\right) + C_{EZ}\left(H_2|_{i+\frac{1}{2},j-\frac{1}{2},k}^{n+\frac{1}{2}} - H_2|_{i-\frac{1}{2},j+\frac{1}{2},k}^{n+\frac{1}{2}}\right) + C_{EZ}\left(H_3|_{i+\frac{1}{2},j+\frac{1}{2},k}^{n+\frac{1}{2}} - H_3|_{i-\frac{1}{2},j-\frac{1}{2},k}^{n+\frac{1}{2}}\right) \quad (3.18)$$

onde o coeficiente C_{EZ0} = 1, mas poderá ser alterado para meios com perdas, bem como os demais coeficientes (ver subseção 3.2.1). O coeficiente C_{EZ} é o mesmo definido na equação (3.12), e repetido por conveniência, como:

$$C_{EZ} = \frac{2}{3}S_{CFL}.Z \quad (3.19)$$

A dedução para o campo magnético H_1 (no modo TM_Z) é um pouco mais complexa, pois, além das componentes de campo elétrico E_Z, é necessário considerar as componentes dos campos elétricos E_2 e E_3 das grades TE_Z acima ($k + 1/2$) e abaixo ($k - 1/2$) da grade TM_Z (na posição k). Por consistência geométrica e física é necessário considerar, por exemplo na Figura 3.3 (que é uma visão expandida da Figura 3.2), o campo magnético H_1 na posição (i, $j - 1$, k) dividido em duas componentes (H_{1E2} e H_{1E3}) perpendiculares aos contornos dos dois campos E_2 nas posições ($i + 1/6$, $j - 3/2$, $k \pm 1/2$) e dos dois campos E_3 nas posições ($i + 1/6$, $j - 1/2$, $k \pm 1/2$), respectivamente.

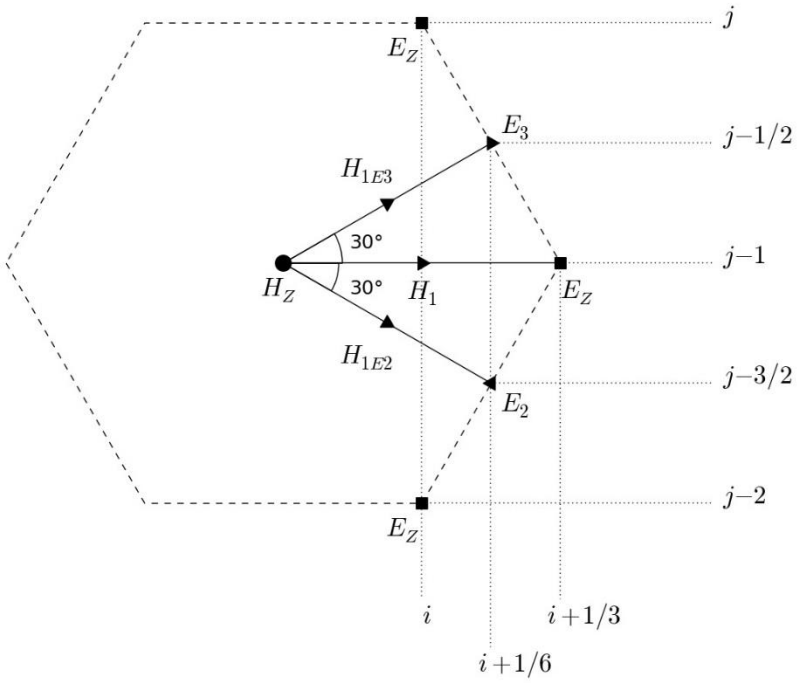


Figura 3.3 - Visão expandida da Figura 3.2 para mostrar as duas componentes (H_{1E2} e H_{1E3}) do campo magnético H_1.

Para obter as duas componentes do campo magnético H_1 (H_{1E2} e H_{1E3}), aplica-se a forma integral da equação de Maxwell (3.5a) nas duas faces retangulares do prisma com o hexágono menor da Figura 3.3, de forma que:

$$\left.\frac{\partial \mu H_{1E2}}{\partial t}\right|^{n}_{i+\frac{1}{6},j-\frac{3}{2},k} . A_R = \left(E_z|^{n}_{i,j-2,k} - E_z|^{n}_{i+\frac{1}{3},j-1,k}\right).\Delta z + \\ + \left(E_2|^{n}_{i+\frac{1}{6},j-\frac{3}{2},k+\frac{1}{2}} - E_2|^{n}_{i+\frac{1}{6},j-\frac{3}{2},k-\frac{1}{2}}\right).\Delta b \qquad (3.20a)$$

$$\left.\frac{\partial \mu H_{1E3}}{\partial t}\right|_{i+\frac{1}{6},j-\frac{1}{2},k}^{n} . A_R = \left(E_z|_{i+\frac{1}{3},j-1,k}^{n} - E_z|_{i,j,k}^{n}\right).\Delta z + \\ + \left(E_3|_{i+\frac{1}{6},j-\frac{1}{2},k+\frac{1}{2}}^{n} - E_3|_{i+\frac{1}{6},j-\frac{1}{2},k-\frac{1}{2}}^{n}\right).\Delta b \quad (3.20b)$$

onde a área (A_R) de cada uma das duas faces retangulares do prisma com hexágono menor é dada por:

$$A_R = \Delta b.\Delta z \quad (3.21)$$

Os fluxos magnéticos (Φ_A e Φ_B) das componentes de campo H_{1E2} e H_{1E3}, nas suas respectivas faces retangulares (com área A_R), são definidos como:

$$\Phi_A = \mu.H_{1E2}|_{i+\frac{1}{6},j-\frac{3}{2},k}^{n}.A_R \quad (3.22a)$$

$$\Phi_B = \mu.H_{1E3}|_{i+\frac{1}{6},j-\frac{1}{2},k}^{n}.A_R \quad (3.22b)$$

O fluxo magnético (Φ_T) produzido pelo campo H_1 na face retangular (com área $A_T = \Delta d.\Delta z$), entre os campos elétricos E_Z nas posições (i, j, k) e (i, j - 2, k) da Figura 3.3, é definido como:

$$\Phi_T = \mu.H_1|_{i,j-1,k}^{n}.A_T \quad (3.23)$$

onde a área (A_T) é definida em função da área (A_R) usando as equações (3.1) e (3.21) de forma a obter:

$$A_T = \Delta d.\Delta z = \sqrt{3}\,\Delta b.\Delta z = \sqrt{3}\,A_R \quad (3.24)$$

Aplica-se a lei de Gauss para campos magnéticos, dada pela equação (2.25), nas superfícies onde existem os fluxos magnéticos (Φ_T, Φ_A e Φ_B) produzidos pelos seus respectivos campos H_1, H_{1E2} e H_{1E3}. Assim, obtém-se:

$$\Phi_T = \Phi_A + \Phi_B \quad (3.25a)$$

e usa-se as equações (3.22a), (3.22b), (3.23) e (3.24) em (3.25a) para obter:

$$\mu.H_1|^n_{i,j-1,k}.\sqrt{3}.A_R = \mu.H_{1E2}|^n_{i+\frac{1}{6},j-\frac{3}{2},k}.A_R + \mu.H_{1E3}|^n_{i+\frac{1}{6},j-\frac{1}{2},k}.A_R \quad (3.25b)$$

Deriva-se a equação (3.25b) em relação ao tempo de forma a obter:

$$\frac{\partial \mu H_1}{\partial t}\Big|^n_{i,j-1,k} \sqrt{3}.A_R = \frac{\partial \mu H_{1E2}}{\partial t}\Big|^n_{i+\frac{1}{6},j-\frac{3}{2},k}.A_R + \frac{\partial \mu H_{1E3}}{\partial t}\Big|^n_{i+\frac{1}{6},j-\frac{1}{2},k}.A_R \quad (3.26a)$$

e simplifica-se para:

$$\frac{\partial \mu H_1}{\partial t}\Big|^n_{i,j-1,k} = \frac{1}{\sqrt{3}}\left(\frac{\partial \mu H_{1E2}}{\partial t}\Big|^n_{i+\frac{1}{6},j-\frac{3}{2},k} + \frac{\partial \mu H_{1E3}}{\partial t}\Big|^n_{i+\frac{1}{6},j-\frac{1}{2},k}\right) \quad (3.26b)$$

Será provado na seção 4.6 que a relação definida pela equação (3.26b) é aquela que produz mínima anisotropia numérica de velocidade de fase nos planos x-z ou y-z.

O campo elétrico E_Z na posição ($i + 1/3, j - 1, k$), presente nas equações (3.20a) e (3.20b), não é calculado diretamente pela equação (3.18), mas isso não será um problema, pois ele será eliminado quando somando as equações (3.20a) e (3.20b). As equações (3.20a) e (3.20b) são aplicadas e somadas no lado direito da equação (3.26a). Usa-se, também, as equações (3.1), (3.7), (3.8), (3.9) e (3.21) na equação (3.26a), em um procedimento similar àquele desenvolvido na seção 2.3.6. Assim, obtém-se uma equação de diferença finita (com precisão de 2ª ordem no tempo e espaço) para o campo magnético H_1 definida como:

$$\begin{aligned} H_1|^{n+\frac{1}{2}}_{i,j-1,k} = {} & C_{H10}.H_1|^{n-\frac{1}{2}}_{i,j-1,k} + C_{HA}\left(E_z|^n_{i,j-2,k} - E_z|^n_{i,j,k}\right) + \\ & + C_{HB}\left(E_2|^n_{i+\frac{1}{6},j-\frac{3}{2},k+\frac{1}{2}} - E_2|^n_{i+\frac{1}{6},j-\frac{3}{2},k-\frac{1}{2}}\right) + \\ & + C_{HB}\left(E_3|^n_{i+\frac{1}{6},j-\frac{1}{2},k+\frac{1}{2}} - E_3|^n_{i+\frac{1}{6},j-\frac{1}{2},k-\frac{1}{2}}\right) \end{aligned} \quad (3.27)$$

onde o coeficiente $C_{H10} = 1$. Os coeficientes C_{HA} e C_{HB} na equação (3.27) são definidos como:

$$C_{HA} = \frac{S_{CFL}}{Z} \tag{3.28}$$

$$C_{HB} = \frac{R}{\sqrt{3}} \frac{S_{CFL}}{Z} \tag{3.29}$$

A razão *R*, na equação (3.29), é definida como:

$$R = \frac{\Delta d}{\Delta z} \tag{3.30}$$

Observa-se que o coeficiente C_{HA} na equação (3.27) é igual àquele da equação (3.14); ambos unicamente multiplicam os termos com componentes de campo elétrico (E_Z). Isto é sinal de dedução consistente das equações de diferença finita da grade de primas hexagonais em relação às equações de diferença finita da grade 2D de hexágonos. O número de Courant para a grade de prismas hexagonais é definido como:

$$S_{CFL} = \frac{c.\Delta t}{\Delta d} \leq \sqrt{\frac{6}{\sqrt{9R^4 + 28R^2 + 36} + 3R^2 + 6}} \tag{3.31}$$

Esta equação será deduzida teoricamente na seção 4.3 deste texto.

Aplicando o mesmo procedimento utilizado para o campo magnético H_1, para os campos magnéticos H_2 e H_3, obtêm-se as seguintes equações:

$$\begin{aligned} H_2\big|_{i+\frac{1}{2},j-\frac{1}{2},k}^{n+\frac{1}{2}} = {} & C_{H20}.H_2\big|_{i+\frac{1}{2},j-\frac{1}{2},k}^{n-\frac{1}{2}} + C_{HA}\left(E_z\big|_{i+1,j-1,k}^{n} - E_z\big|_{i,j,k}^{n}\right) + \\ & + C_{HB}\left(-E_1\big|_{i+\frac{2}{3},j-1,k+\frac{1}{2}}^{n} + E_1\big|_{i+\frac{2}{3},j-1,k-\frac{1}{2}}^{n}\right) + \\ & + C_{HB}\left(E_3\big|_{i+\frac{1}{6},j-\frac{1}{2},k+\frac{1}{2}}^{n} - E_3\big|_{i+\frac{1}{6},j-\frac{1}{2},k-\frac{1}{2}}^{n}\right) \end{aligned} \tag{3.32}$$

$$H_3\big|_{i+\frac{1}{2},j+\frac{1}{2},k}^{n+\frac{1}{2}} = C_{H30}.H_3\big|_{i+\frac{1}{2},j+\frac{1}{2},k}^{n-\frac{1}{2}} + C_{HA}\left(E_z\big|_{i+1,j+1,k}^{n} - E_z\big|_{i,j,k}^{n}\right) +$$

$$+\ C_{HB}\left(-E_1|^{n}_{i+\frac{2}{3},j+1,k+\frac{1}{2}}+\ E_1|^{n}_{i+\frac{2}{3},j+1,k-\frac{1}{2}}\right)+$$
$$+\ C_{HB}\left(-E_2|^{n}_{i+\frac{1}{6},j+\frac{1}{2},k+\frac{1}{2}}+\ E_2|^{n}_{i+\frac{1}{6},j+\frac{1}{2},k-\frac{1}{2}}\right) \quad (3.33)$$

onde os coeficientes C_{H20} = C_{H30} = 1. Os coeficientes C_{HA} e C_{HB}, presentes nas equações (3.32) e (3.33), já foram definidos nas equações (3.28) e (3.29), respectivamente.

As equações de diferença finita para o modo TE$_Z$ são obtidas usando o mesmo procedimento do modo TM$_Z$, mas com a necessária troca de sinais e alterações nos coeficientes para consistência física. Assim, para o campo magnético H_Z tem-se a seguinte equação:

$$H_z|^{n+\frac{1}{2}}_{i+\frac{2}{3},j,k+\frac{1}{2}} = C_{HZ0}.H_z|^{n-\frac{1}{2}}_{i+\frac{2}{3},j,k+\frac{1}{2}} - C_{HZ}\left(E_1|^{n}_{i+\frac{2}{3},j-1,k+\frac{1}{2}} - E_1|^{n}_{i+\frac{2}{3},j+1,k+\frac{1}{2}}\right) -$$
$$- C_{HZ}\left(E_2|^{n}_{i+\frac{7}{6},j-\frac{1}{2},k+\frac{1}{2}} - E_2|^{n}_{i+\frac{1}{6},j+\frac{1}{2},k+\frac{1}{2}}\right) -$$
$$- C_{HZ}\left(E_3|^{n}_{i+\frac{7}{6},j+\frac{1}{2},k+\frac{1}{2}} - E_3|^{n}_{i+\frac{1}{6},j-\frac{1}{2},k+\frac{1}{2}}\right) \quad (3.34)$$

onde o coeficiente C_{HZ0} = 1. O coeficiente C_{HZ} na equação (3.34) é definido como:

$$C_{HZ} = \frac{2}{3}\frac{S_{CFL}}{Z} \quad (3.35)$$

E as equações de diferença finita para os campos elétricos E_1, E_2 e E_3 são definidas como:

$$E_1|^{n+1}_{i+\frac{2}{3},j-1,k+\frac{1}{2}} = C_{E10}.E_1|^{n}_{i+\frac{2}{3},j-1,k+\frac{1}{2}} - C_{EA}\left(H_z|^{n+\frac{1}{2}}_{i+\frac{2}{3},j-2,k+\frac{1}{2}} - H_z|^{n+\frac{1}{2}}_{i+\frac{2}{3},j,k+\frac{1}{2}}\right) -$$
$$- C_{EB}\left(H_2|^{n+\frac{1}{2}}_{i+\frac{1}{2},j-\frac{1}{2},k+1} - H_2|^{n+\frac{1}{2}}_{i+\frac{1}{2},j-\frac{1}{2},k}\right) -$$
$$- C_{EB}\left(H_3|^{n+\frac{1}{2}}_{i+\frac{1}{2},j-\frac{3}{2},k+1} - H_3|^{n+\frac{1}{2}}_{i+\frac{1}{2},j-\frac{3}{2},k}\right) \quad (3.36)$$

$$E_2\big|_{i+\frac{7}{6},j-\frac{1}{2},k+\frac{1}{2}}^{n+1} = C_{E20}.E_2\big|_{i+\frac{7}{6},j-\frac{1}{2},k+\frac{1}{2}}^{n} - C_{EA}\left(H_z\big|_{i+\frac{5}{3},j-1,k+\frac{1}{2}}^{n+\frac{1}{2}} - H_z\big|_{i+\frac{2}{3},j,k+\frac{1}{2}}^{n+\frac{1}{2}}\right) - C_{EB}\left(-H_1\big|_{i+1,j,k+1}^{n+\frac{1}{2}} + H_1\big|_{i+1,j,k}^{n+\frac{1}{2}}\right) - C_{EB}\left(H_3\big|_{i+\frac{3}{2},j-\frac{1}{2},k+1}^{n+\frac{1}{2}} - H_3\big|_{i+\frac{3}{2},j-\frac{1}{2},k}^{n+\frac{1}{2}}\right) \quad (3.37)$$

$$E_3\big|_{i+\frac{7}{6},j+\frac{1}{2},k+\frac{1}{2}}^{n+1} = C_{E30}.E_3\big|_{i+\frac{7}{6},j+\frac{1}{2},k+\frac{1}{2}}^{n} - C_{EA}\left(H_z\big|_{i+\frac{5}{3},j+1,k+\frac{1}{2}}^{n+\frac{1}{2}} - H_z\big|_{i+\frac{2}{3},j,k+\frac{1}{2}}^{n+\frac{1}{2}}\right) - C_{EB}\left(-H_1\big|_{i+1,j,k+1}^{n+\frac{1}{2}} + H_1\big|_{i+1,j,k}^{n+\frac{1}{2}}\right) - C_{EB}\left(-H_2\big|_{i+\frac{3}{2},j+\frac{1}{2},k+1}^{n+\frac{1}{2}} + H_2\big|_{i+\frac{3}{2},j+\frac{1}{2},k}^{n+\frac{1}{2}}\right) \quad (3.38)$$

com os coeficientes $C_{E10} = C_{E20} = C_{E30} = 1$. Os coeficientes C_{EA} e C_{EB} nas equações (3.36), (3.37) e (3.38) são definidos como:

$$C_{EA} = S_{CFL}.Z \quad (3.39)$$

$$C_{EB} = \frac{R}{\sqrt{3}} S_{CFL}.Z \quad (3.40)$$

A razão *R* na equação (3.40) é definida como na equação (3.30).

A fim de que a formulação proposta (uso de prismas hexagonais) produza resultado físicos corretos, é desejável que ela implicitamente garanta a conservação dos fluxos elétrico e magnético. Isto é, se as condições iniciais do problema produzem divergência zero dos campos elétrico e magnético, esta divergência zero deve ser satisfeita para qualquer instante de tempo. É fácil provar que as equações deduzidas para os modos TM_Z e TE_Z satisfazem a condição de divergência zero para qualquer célula da grade (prisma com o hexágono menor), seguindo o procedimento descrito em (INAN; MARSHALL, 2011), a partir das equações:

$$\oint_S \varepsilon . \boldsymbol{E} . \boldsymbol{ds} = 0 \quad \rightarrow \quad \oint_S \frac{\partial \varepsilon \boldsymbol{E}}{\partial t} . \boldsymbol{ds} = 0 \tag{3.41a}$$

$$\oint_S \mu . \boldsymbol{H} . \boldsymbol{ds} = 0 \quad \rightarrow \quad \oint_S \frac{\partial \mu \boldsymbol{H}}{\partial t} . \boldsymbol{ds} = 0 \tag{3.41b}$$

3.2.1 Coeficientes para meios com perdas e não-homogêneos

Para obter coeficientes para meios com perdas, um procedimento similar àquele desenvolvido na subseção 2.3.8 pode ser aplicado nas oito equações de diferença finita da grade de prismas hexagonais. Assim, os coeficientes nas equações (3.18), (3.27), (3.32), (3.33), (3.34), (3.36), (3.37) e (3.38) são alterados para:

$$C_{E10} = C_{E20} = C_{E30} = C_{EZ0} = \frac{2\varepsilon - \sigma \Delta t}{2\varepsilon + \sigma \Delta t} \tag{3.42a}$$

$$C_{H10} = C_{H20} = C_{H30} = C_{HZ0} = \frac{2\mu - \sigma_m \Delta t}{2\mu + \sigma_m \Delta t} \tag{3.42b}$$

$$C_{EZ} = \frac{4.\Delta t}{3.\Delta d.(2\varepsilon + \sigma \Delta t)} \tag{3.42c}$$

$$C_{HZ} = \frac{4.\Delta t}{3.\Delta d.(2\mu + \sigma_m \Delta t)} \tag{3.42d}$$

$$C_{EA} = \frac{2.\Delta t}{\Delta d.(2\varepsilon + \sigma \Delta t)} \tag{3.42e}$$

$$C_{HA} = \frac{2.\Delta t}{\Delta d.(2\mu + \sigma_m \Delta t)} \tag{3.42f}$$

$$C_{EB} = \frac{2.R.\Delta t}{\sqrt{3}.\Delta d.(2\varepsilon + \sigma \Delta t)} \tag{3.42g}$$

$$C_{HB} = \frac{2.R.\Delta t}{\sqrt{3}.\Delta d.(2\mu + \sigma_m \Delta t)} \tag{3.42h}$$

Os coeficientes das equações (3.42a) a (3.42h) podem ser definidos em cada ponto da grade 3D, logo é possível simular meios não-homogêneos. A razão *R* é definida

pela equação (3.30). O tamanho do passo de tempo Δt é calculado a partir da equação (3.9) como:

$$\Delta t \leq \frac{S_{CFL}.\Delta d}{c_0} \tag{3.43}$$

onde c_0 é a velocidade física da onda no vácuo ($c_0 \approx 3\ 10^8$ m/s) e o número de Courant (S_{CFL}) é obtido da equação (3.31) em função da razão R.

3.3 VALIDAÇÃO DO MÉTODO FDTD APLICADO À GRADE DE PRISMAS HEXAGONAIS

Antes de analisar as anisotropia e dispersão numéricas para o método FDTD aplicado à grade de prismas hexagonais, é importante validar esta nova formulação comparando-a com o método FDTD Yee com células cúbicas. A comparação será feita para grades tridimensionais equivalentes que tenham dimensões (L_X, L_Y, L_Z) iguais nas três direções (x, y, z), tal que:

$$L_X = N_X \Delta x \tag{3.44a}$$

$$L_Y = N_Y \Delta y \tag{3.44b}$$

$$L_Z = N_Z \Delta z \tag{3.44c}$$

$$L_X = L_Y = L_Z = L \tag{3.44d}$$

Para a grade de células cúbicas (método Yee) o número de passos espaciais nas três direções (x, y, z) será igual, isto é: $N_X = N_Y = N_Z = N$ e $\Delta x = \Delta y = \Delta z = \Delta$. Assim, obtém-se a mesma dimensão (L) para as três direções.

Para a grade de prismas hexagonais o número de passos espaciais nas três direções (x, y, z) é definido como: $N_X = N$ / cos(30°), $N_Y = 2.N$, $N_Z = R.N$; os incrementos de espaço (Δx, Δy, Δz) são definidos pelas equações (3.2), (3.3) e (3.30), respectivamente, repetidas aqui, por conveniência, como:

$$\Delta x = \Delta d . \cos 30° = \frac{\sqrt{3}}{2} \Delta d \quad (3.45a)$$

$$\Delta y = \frac{1}{2} \Delta d \quad (3.45b)$$

$$\Delta z = \frac{\Delta d}{R} \quad (3.45c)$$

O valor para o lado do hexágono maior é: $\Delta d = \Delta$, e a razão $R = 1$. Estes valores são aplicados nas equações (3.45) para obter uma grade tridimensional de dimensões equivalentes às da grade tridimensional do método FDTD Yee.

É interessante observar que, para o método FDTD com prismas hexagonais, os campos elétricos ou magnéticos, em uma certa linha (índice *j* na direção *y*), são unicamente calculados para índices (*i*) par ou ímpar na direção *x* (ver Figura 3.2). Dessa forma, o número total de pontos que será armazenado para cada campo na grade de prismas hexagonais é em torno de $N_X.N_Y.N_Z$ / 2. Este valor, para as condições desta seção, é em torno de 16% maior que o valor total de pontos armazenados para cada campo na grade Yee. Contudo, na grade de prismas hexagonais é necessário calcular oito campos em vez dos seis campos da grade Yee. Consequentemente, a grade de prismas hexagonais necessita armazenar 54% a mais de número de pontos para todos os campos que a grade Yee.

3.3.1 Tipos de fontes usadas nas simulações com os dois métodos FDTD

A validação do método FDTD com células em forma de prismas hexagonais será feita para dois tipos de fontes (pontuais) de sinal: senoidal e pulso. Estas fontes de sinal geram o campo elétrico E_Z no ponto central (N_X / 2, N_Y / 2, N_Z / 2) da grade tridimensional do método FDTD Yee ou do método FDTD com prismas hexagonais. A fonte senoidal é definida pela equação:

$$F_s(t) = A.sen(\omega t) \quad (3.46)$$

A amplitude de pico da função senoidal (*A*) é, por simplicidade, normalizada como: $A = 1$. A frequência angular (ω), como já visto, é definida como: $\omega = 2\pi.f$, e a frequência (*f*) pode ser expressada como:

$$c = f.\lambda \quad \Rightarrow \quad f = \frac{c}{\lambda} \tag{3.47}$$

onde *c* é a velocidade (física) da onda eletromagnética e λ é o seu comprimento de onda (em metros). O argumento (ωt) pode ser escrito, usando a equação (3.47), da seguinte forma:

$$\omega t = 2\pi f t = \frac{2\pi c}{\lambda} t \tag{3.48}$$

O comprimento de onda pode ser definido a partir do passo espacial (Δ) como:

$$\lambda = N_\lambda . \Delta \tag{3.49}$$

Para o método FDTD com prismas hexagonais: Δ*d* = Δ; para o método FDTD Yee: Δ*x* = Δ. O parâmetro N_λ é o número de pontos por comprimento de onda; este número não necessita ser um inteiro. O tempo (*t*) também é expresso por passos no tempo, tal que:

$$t = n.\Delta \mathrm{t} \tag{3.50}$$

O número de passos no tempo (*n*) é sempre um número inteiro. Aplicando as equações (3.49) e (3.50) na equação (3.48), e usando a definição do número de Courant (S_{CFL}) da equação (3.9), obtém-se:

$$\omega t = \frac{2\pi c}{N_\lambda . \Delta} . n\Delta t = \frac{2\pi}{N_\lambda} . \frac{c.\Delta t}{\Delta} . n = \frac{2\pi . S_{CFL}}{N_\lambda} . n \tag{3.51}$$

Aplicando a equação (3.51) na equação (3.46), tem-se:

$$F_s(n) = sen\left(\frac{2\pi . S_{CFL}}{N_\lambda} . n\right) \tag{3.52}$$

O uso do número de pontos por comprimento de onda (N_λ) e do número de Courant (S_{CFL}), que contém, implicitamente, a taxa de passos temporal e espacial,

elimina a necessidade de definir explicitamente os incrementos temporal (Δ*t*) ou espacial (Δ*d* ou Δ*x*). Assim, é possível tratar as simulações FDTD como simulações genéricas, escalonadas para qualquer tamanho ou frequência (SCHNEIDER, 2013).

É possível, também, suavizar o início da função senoidal da equação (3.52) para reduzir oscilações ou ruídos indesejáveis na grade. Antes de fazer esta suavização é necessário usar a definição de período (*T*) da função senoidal como:

$$T = \frac{1}{f} = \frac{\lambda}{c} = \frac{N_\lambda.\Delta}{c} \tag{3.53}$$

O número de passos de um período usando a equação (3.53) é dado por:

$$\frac{T}{\Delta t} = \frac{N_\lambda.\Delta}{c.\Delta t} = \frac{N_\lambda}{S_{CFL}} \tag{3.54}$$

Dessa forma define-se uma constante de tempo (ς) em função desse número de passos por período e de um fator de atenuação (α_T) como:

$$\varsigma = \alpha_T.\frac{N_\lambda}{S_{CFL}} \tag{3.55}$$

Assim, a função senoidal, que gera o campo elétrico E_Z, torna-se:

$$F_s(n) = \left(1 - e^{-n/\varsigma}\right).sen\left(\frac{2\pi.S_{CFL}}{N_\lambda}.n\right) \tag{3.56}$$

Quanto maior o valor do fator de atenuação (α_T), mais suave será o crescimento do sinal senoidal no início da simulação. O valor usado para o fator de atenuação nas simulações com fonte senoidal foi α_T = 0,5. A Figura 3.4 ilustra esta fonte senoidal variando no tempo.

O segundo tipo de fonte de sinal utiliza um pulso especial, chamado em inglês *Ricker wavelet*, que é equivalente à derivada segunda de um pulso gaussiano (SCHNEIDER, 2013). Tem como vantagens não possuir uma componente contínua (DC), como ocorre com o pulso gaussiano, o que evita a introdução de ruídos na grade; e, também, permite um ajuste mais simples da frequência desejada (com maior energia) na simulação. A equação do pulso *Ricker* é definida pela equação (3.57):

$$F_R(t) = \left(1 - 2.\left[\pi f_p(t - d_r)\right]^2\right).exp\left(-\left[\pi f_p(t - d_r)\right]^2\right) \quad (3.57)$$

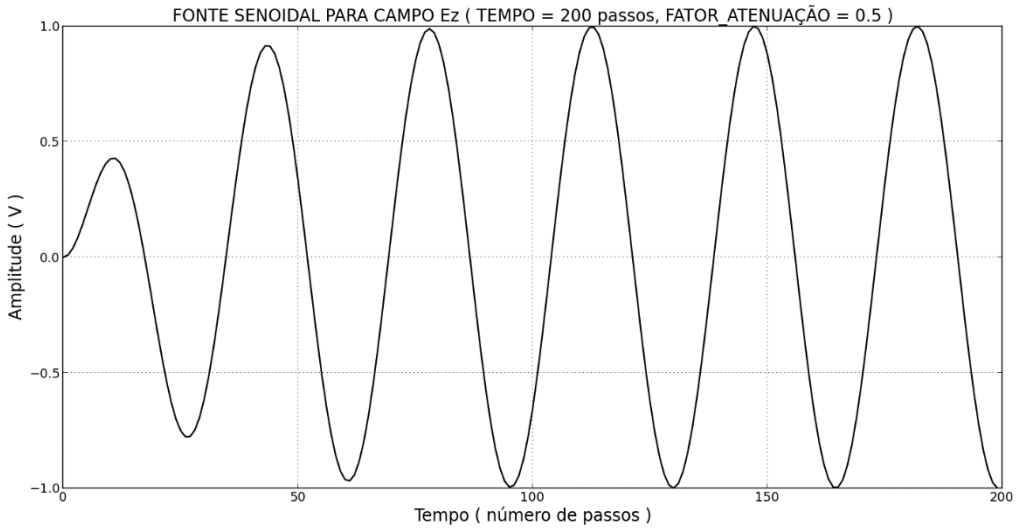


Figura 3.4 - Função senoidal no tempo para gerar o campo elétrico E_Z.

Na equação (3.57) f_p é a frequência de pico (com maior energia no espectro de frequências do pulso) e d_r é o atraso temporal. O atraso pode ser qualquer quantidade desejada, mas é conveniente expressá-lo como um múltiplo M_d (não necessariamente um número inteiro) de 1 / f_p, como na equação (3.58):

$$d_r = M_d.\frac{1}{f_p} \quad (3.58)$$

Quanto maior M_d mais suave será a entrada do pulso na grade. A frequência de pico (f_p) é mais convenientemente definida como:

$$f_p = \frac{c}{\lambda_p} = \frac{c}{N_p \Delta} \quad (3.59)$$

onde N_p é o número de pontos por comprimento de onda (λ_p). O incremento de espaço é: $\Delta = c.\Delta t / S_{CFL}$; quando aplicado na equação (3.59), resulta em:

$$f_p = \frac{S_{CFL}}{N_p \Delta t} \quad (3.60)$$

O atraso de tempo, aplicando a equação (3.60) na equação (3.58), resulta em:

$$d_r = M_d.\frac{1}{f_p} = M_d \frac{N_p \Delta t}{S_{CFL}} \tag{3.61}$$

Usando $t = n.\Delta t$, o valor f_p da equação (3.60) e o atraso d_r da equação (3.61) na equação (3.57), obtém-se a equação (3.62) em forma discreta do pulso *Ricker*:

$$F_R(n) = \left(1 - 2.\pi^2.\left[\frac{S_{CFL}\ n}{N_p} - M_d\right]^2\right).exp\left(-\pi^2.\left[\frac{S_{CFL}\ n}{N_p} - M_d\right]^2\right) \tag{3.62}$$

Por simplicidade, usou-se na equação (3.62): $N_p = N_\lambda$ e $M_d = 0$, obtendo-se a seguinte forma de onda no tempo do pulso *Ricker*, como mostrado na Figura 3.5.

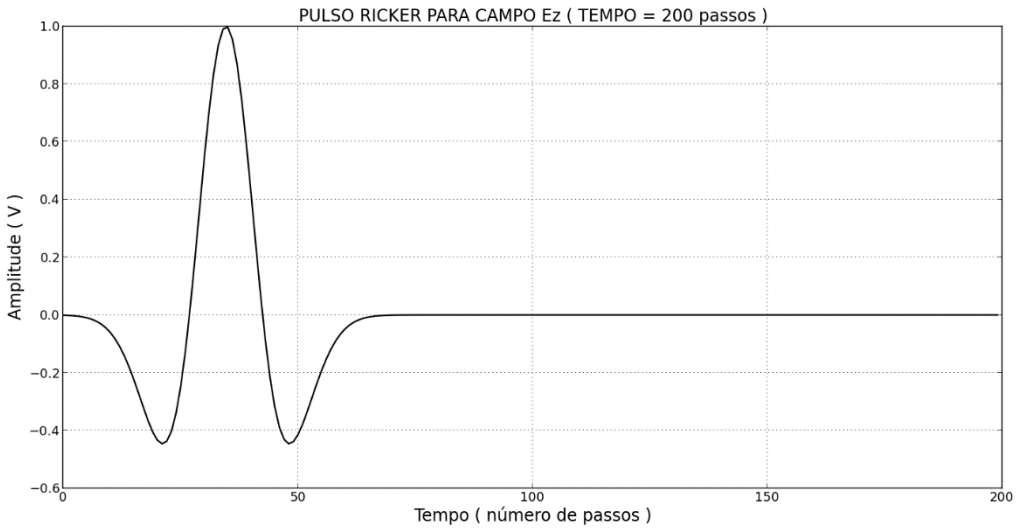


Figura 3.5 - Pulso *Ricker* no tempo para gerar o campo elétrico E_Z.

As fontes de sinal senoidal e de pulso *Ricker* podem, também, ser aplicadas de duas formas: *hard* ou *soft*. A forma *hard*, para uma fonte genérica (F_G) usada para gerar o campo elétrico E_Z no centro da grade, é dada por:

$$E_Z\left(n, \frac{N_X}{2}, \frac{N_Y}{2}, \frac{N_Z}{2}\right) = F_G\left(n, \frac{N_X}{2}, \frac{N_Y}{2}, \frac{N_Z}{2}\right) \tag{3.63}$$

A forma *soft* é dada por:

$$E_Z\left(n,\frac{N_X}{2},\frac{N_Y}{2},\frac{N_Z}{2}\right) = E_Z\left(n-1,\frac{N_X}{2},\frac{N_Y}{2},\frac{N_Z}{2}\right) + F_G\left(n,\frac{N_X}{2},\frac{N_Y}{2},\frac{N_Z}{2}\right) \quad (3.64)$$

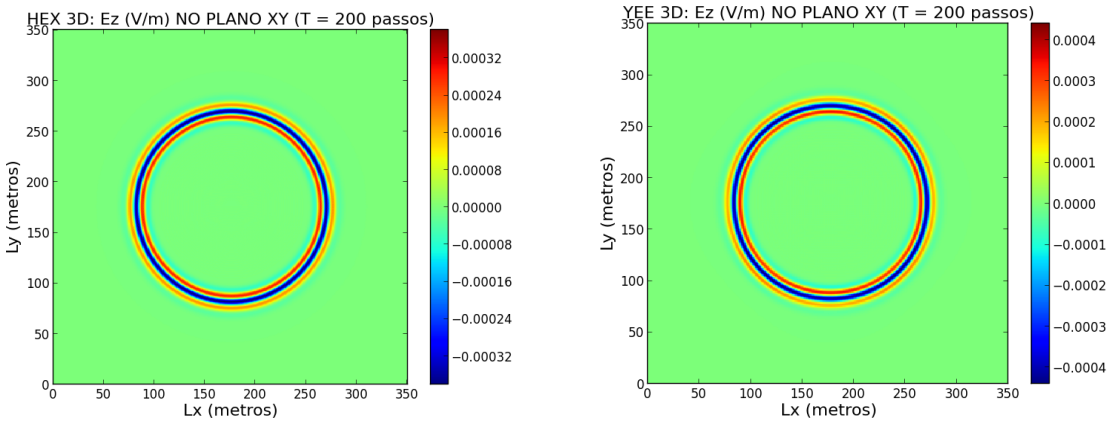
A forma de fonte *hard* impõe o seu sinal e pode ser considerada um ponto de perturbação (reflexão) para sinais refletidos ou provenientes de outras fontes. A forma de fonte *soft* soma o seu sinal ao sinal já existente, e pode ser considerada transparente para sinais refletidos ou de outras fontes existentes no espaço de simulação.

Para que comparações corretas de propagação de onda entre ambos os métodos FDTD possam ser feitas, é necessário que as ondas em ambos os métodos FDTD tenham iguais comprimentos de onda ($\lambda = N_\lambda.\Delta$) e que elas possam ser comparadas para iguais períodos de tempo, usando pulso *Ricker* ou fonte senoidal. Essas condições são satisfeitas usando $\Delta x = \Delta y = \Delta z = \Delta$ para o método FDTD Yee e $\Delta d = \Delta$ para o método FDTD com prismas hexagonais. Devem ser também iguais, para ambos os métodos FDTD, o número de pontos por comprimento de onda (N_λ), o número de Courant (S_{CFL}) e o número de passos de tempo (n); é escolhida razão $R = 1$ ($\Delta z = \Delta d$) na grade de prismas hexagonais, que permite usar o mesmo número de Courant (S_{CFL}) em ambos os métodos FDTD (para mais detalhes sobre número de Courant consultar a seção 4.3). É admitido que todos os campos elétricos e magnéticos, no espaço de simulação de ambos os métodos FDTD, têm valores iniciais iguais a zero (INAN; MARSHALL, 2011).

3.3.3 Comparando os dois métodos FDTD com a fonte de pulso *Ricker*

Os valores a serem usados na simulação do método FDTD Yee são: $N_X = N_Y = N_Z = 351$; $\Delta x = \Delta y = \Delta z = 1{,}0$ m; $N_\lambda = 20$ e $S_{CFL} = 1/\sqrt{3} \approx 0{,}57$. Para o método FDTD com prismas hexagonais usar-se-ão: $N_X = 405$, $N_Y = 701$, $N_Z = 351$; $\Delta d = 1{,}0$ m; R = 1; $N_\lambda = 20$ e $S_{CFL} = 1/\sqrt{3} \approx 0{,}57$. O número máximo de passos de tempo em ambos os métodos FDTD é igual a 200. A fonte de pulso *Ricker* usada é do tipo *hard* e gera um campo elétrico E_Z no centro de cada grade tridimensional. Esta fonte pontual é equivalente a um dipolo curto com dimensão aproximada de $(\lambda / N_\lambda) = \Delta z = 1{,}0$ m em ambos os métodos FDTD (TAFLOVE; HAGNESS, 2005).

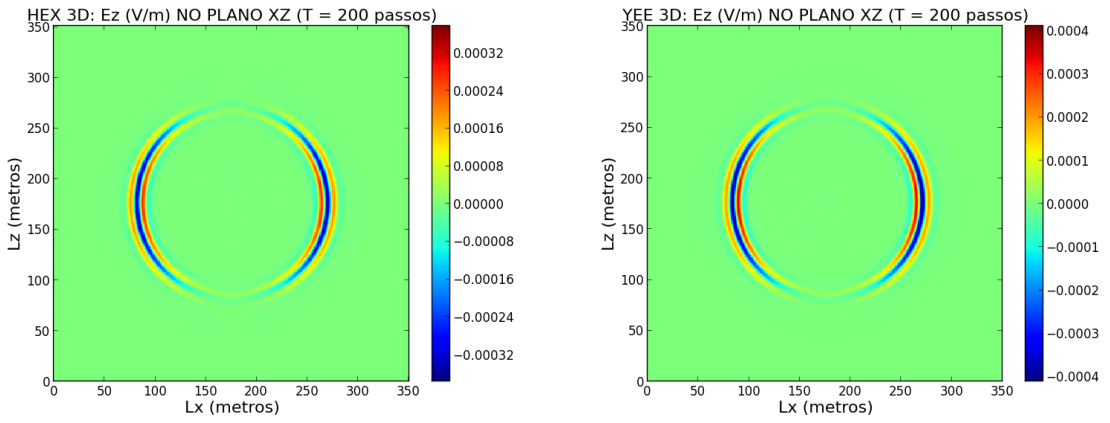
Na Figura 3.6 compara-se visualmente a propagação da onda gerada pelo pulso *Ricker* no plano x-y, usando ambos os métodos FDTD. Nestas figuras as dimensões nos eixos x e y são iguais (L_X = L_Y = 351 m), e os níveis de campo elétrico E_Z são em volt/metro (V/m). Observa-se que a onda se propaga circularmente e com a mesma intensidade em todas as direções nas Figuras 3.6a e 3.6b.



(a) (b)

Figura 3.6 - Comparação de ondas para pulso *Ricker* no plano x-y: (a) prismas hexagonais; (b) células cúbicas (método Yee). A cor verde corresponde ao nível zero do campo neste e nos demais gráficos 2D.

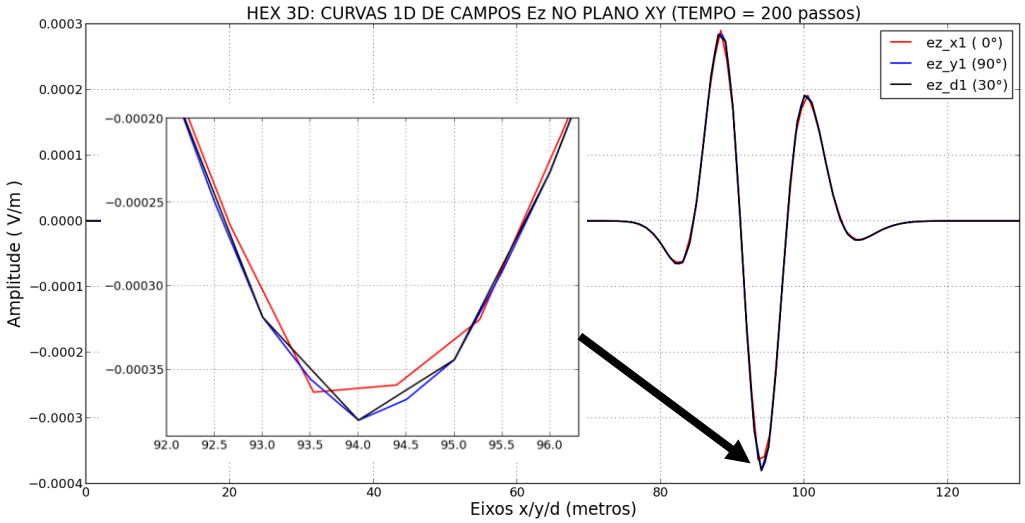
Na Figura 3.7 compara-se visualmente a propagação da onda gerada pelo pulso *Ricker* no plano x-z, usando ambos os métodos FDTD. Nestas figuras as dimensões nos eixos x e z são iguais (L_X = L_Z = 351 m), e os níveis de campo elétrico E_Z são em volt/metro (V/m).



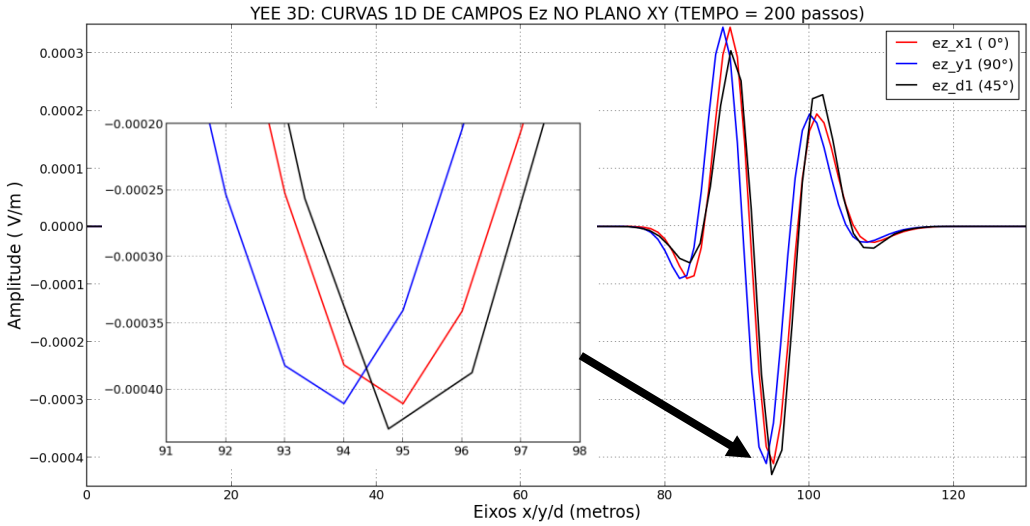
(a) (b)

Figura 3.7 - Comparação de ondas para pulso *Ricker* no plano x-z: (a) prismas hexagonais; (b) células cúbicas (método Yee).

Observa-se que a propagação da onda é também circular, mas a intensidade da onda diminui, à medida que a sua direção de propagação se aproxima da direção *z*. Os níveis de sinal parecem muito próximos entre os dois métodos FDTD nas Figuras 3.6 e 3.7. Na Figura 3.6a, a partir do centro do plano x-y escolhem-se três direções positivas: x_1 (0°), y_1 (90°), d_1 (30°); e plotam-se os níveis de campo elétrico E_Z nestas direções como mostrado na Figura 3.8a. Na Figura 3.6b, a partir do centro do plano x-y escolhem-se três direções positivas: x_1 (0°), y_1 (90°), d_1 (45°); e, também, plotam-se os níveis de campo elétrico E_Z nestas direções como mostrado na Figura 3.8b.



(a)

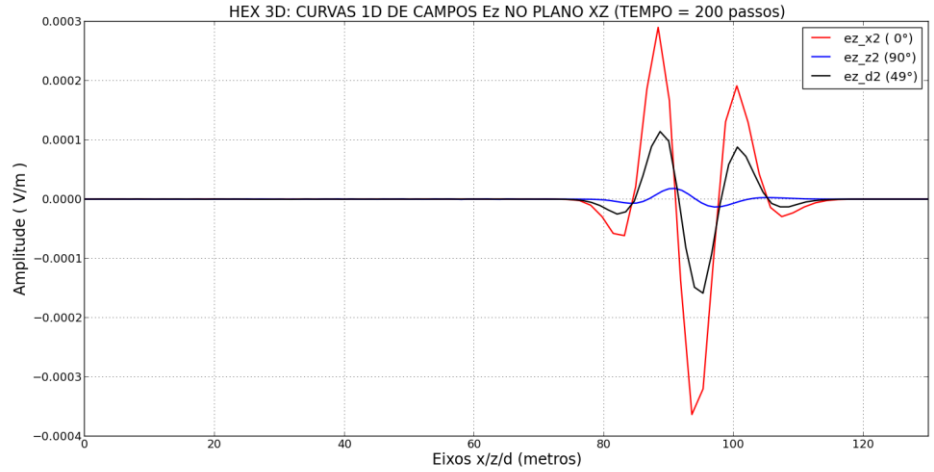


(b)

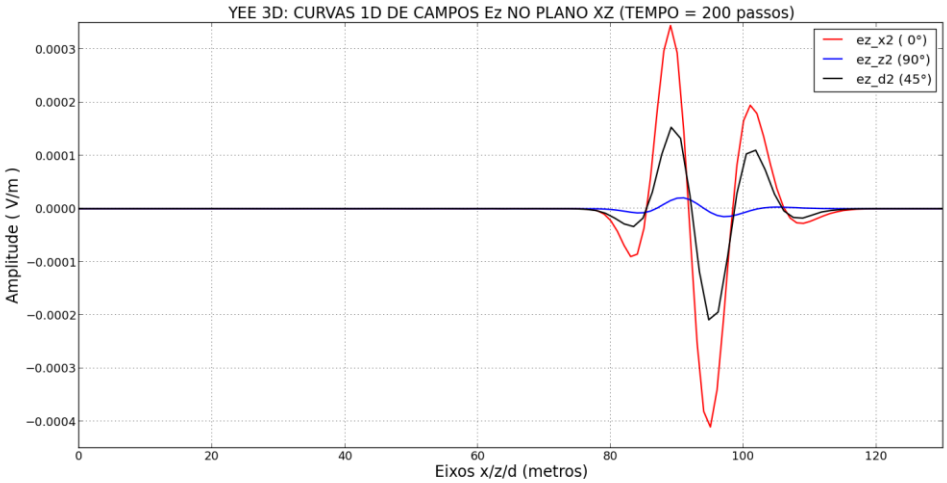
Figura 3.8 - Gráficos para pulso *Ricker* nas direções x_1, y_1, d_1 do plano x-y: (a) prismas hexagonais; (b) células cúbicas (método Yee).

Observa-se visualmente que a anisotropia de velocidade de grupo numérica é menor na Figura 3.8a e que os níveis são muito próximos em ambas as figuras.

Novamente, para a Figura 3.7a, a partir do centro do plano x-z escolhem-se três direções positivas: x_2 (0°), z_2 (90°), d_2 (49°); e plotam-se os níveis de campo elétrico E_Z nestas direções como mostrado na Figura 3.9a. A direção d_2 (49°) corresponde à diagonal das células retangulares no plano x-z, para a grade de prismas hexagonais. E, também, para a Figura 3.7b, a partir do centro do plano x-z escolhem-se três direções positivas: x_2 (0°), z_2 (90°), d_2 (45°); e plotam-se os níveis de campo elétrico E_Z nestas direções como mostrado na Figura 3.9b. A direção d_2 (45°) corresponde à diagonal das células quadradas no plano x-z, para a grade de células cúbicas.



(a)



(b)

Figura 3.9 - Gráficos para pulso *Ricker* nas direções x_2, y_2, d_2 do plano x-z: (a) prismas hexagonais; (b) células cúbicas (método Yee).

Neste caso, a observação visual entre as Figuras 3.9a e 3.9b é muito subjetiva, embora os níveis estejam próximos. Para uma observação mais acurada, a partir das Figuras 3.9a e 3.9b, geram-se gráficos que comparam os níveis de campo elétrico E_Z de ambos os métodos FDTD, de forma separada, para as três direções positivas (x_2, z_2, d_2) presentes no plano x-z como mostram as Figuras 3.10, 3.11 e 3.12. Não é necessário comparar as ondas para as três direções positivas (x_1, y_1, d_1) no plano x-y de ambos os métodos FDTD, pois nesse plano as formas de onda são quase idênticas, e a onda na direção x_2 (no plano x-z) é idêntica à onda na direção x_1 (no plano x-y).

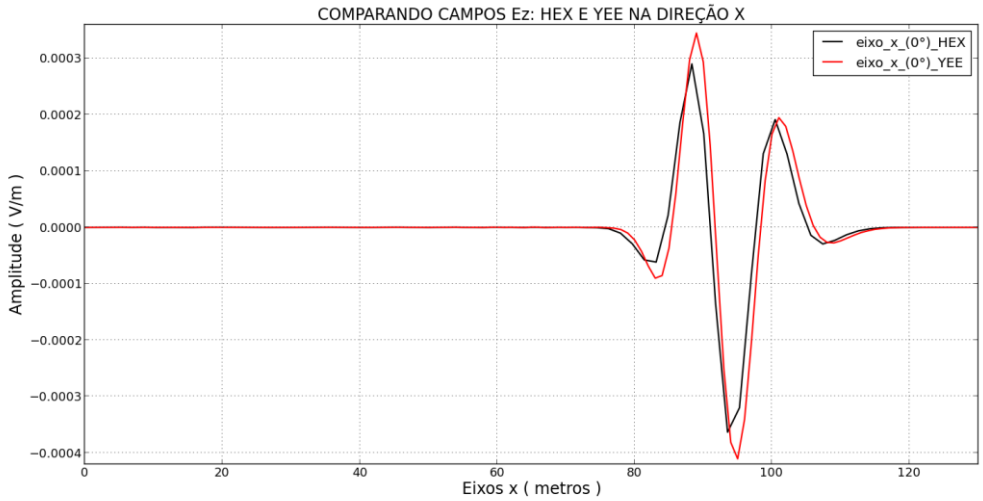


Figura 3.10 - Comparação de campos elétricos E_Z na direção x_2 (plano x-z) usando pulso *Ricker* para os dois métodos FDTD.

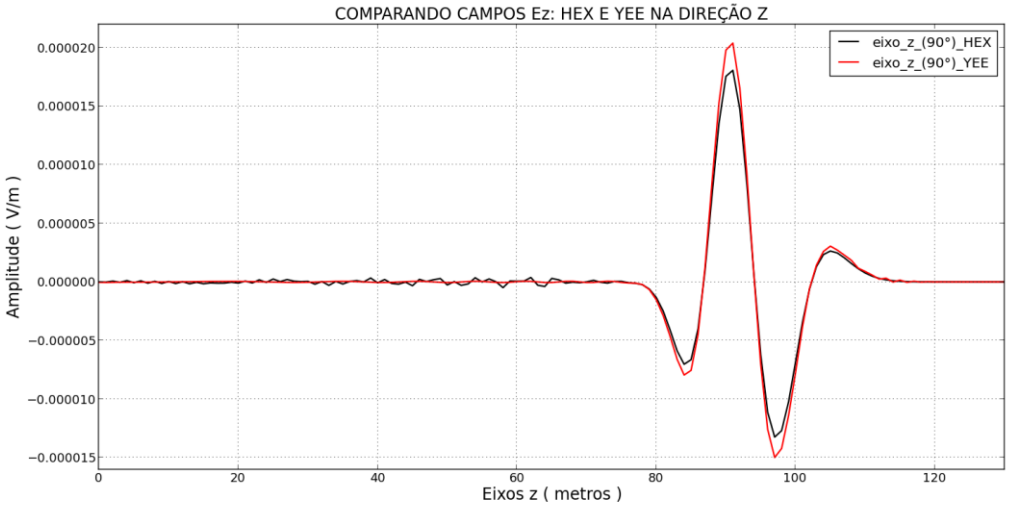


Figura 3.11 - Comparação de campos elétricos E_Z na direção z_2 (plano x-z) usando pulso *Ricker* para os dois métodos FDTD.

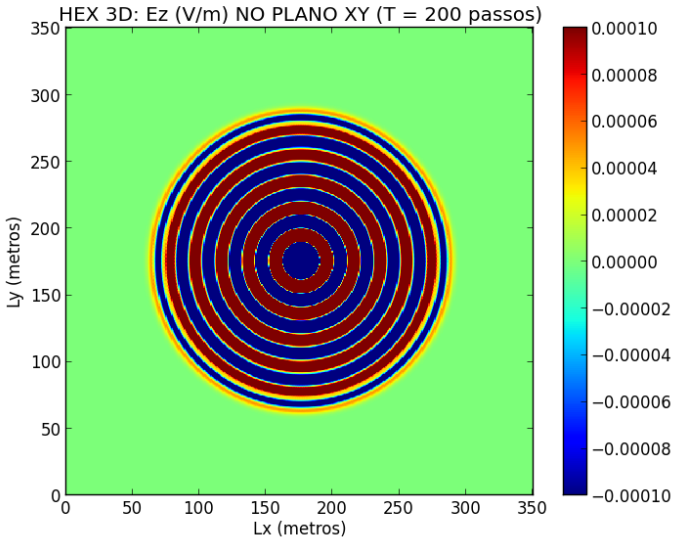
Figura 3.12 - Comparação de campos elétricos E_Z na direção d_2 (plano x-z) usando pulso *Ricker* para os dois métodos FDTD.

Observa-se nas Figuras 3.10, 3.11 e 3.12 uma similaridade nas formas de onda entre os dois métodos FDTD, quando utilizando como fonte um pulso *Ricker*, embora os níveis de campo elétrico E_Z do método FDTD com prismas hexagonais, sejam um pouco menores em relação aqueles do método FDTD Yee. Na Figura 3.12, embora as direções de comparação entre os dois métodos FDTD não sejam exatamente iguais, as formas de onda e fase são similares; a principal diferença está no nível levemente maior da onda para o método Yee.

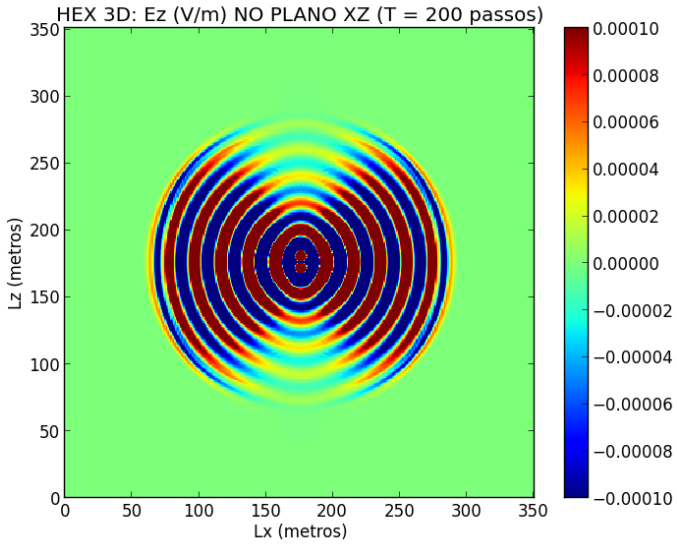
3.3.3 Comparando os dois métodos FDTD com a fonte senoidal

Os valores dos parâmetros utilizados são os mesmos com a exceção do tipo de fonte, que será do tipo senoidal *hard*. O número máximo de passos de tempos em ambos os métodos FDTD será, também, igual a 200. Utilizar-se-á um fator de atenuação $\alpha_T = 0{,}5$ para suavizar um pouco a entrada do sinal senoidal em ambas as grades tridimensionais. As formas de onda bidimensionais no plano x-y e no plano x-z para a grade de prismas hexagonais é mostrada na Figura 3.13.

Estas mesmas formas de onda bidimensionais não serão mostradas para o método FDTD Yee, por serem redundantes. Os gráficos bidimensionais das Figuras 3.13a e 3.13b (com fonte senoidal) apresentam diagramas de radiação semelhantes àqueles das Figuras 3.6 e 3.7 (com pulso *Ricker*), respectivamente.



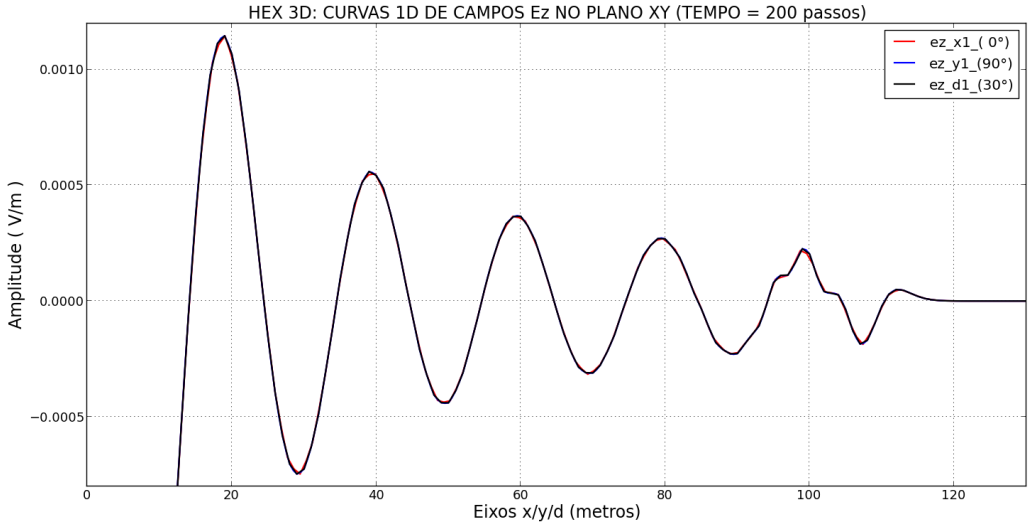
(a)



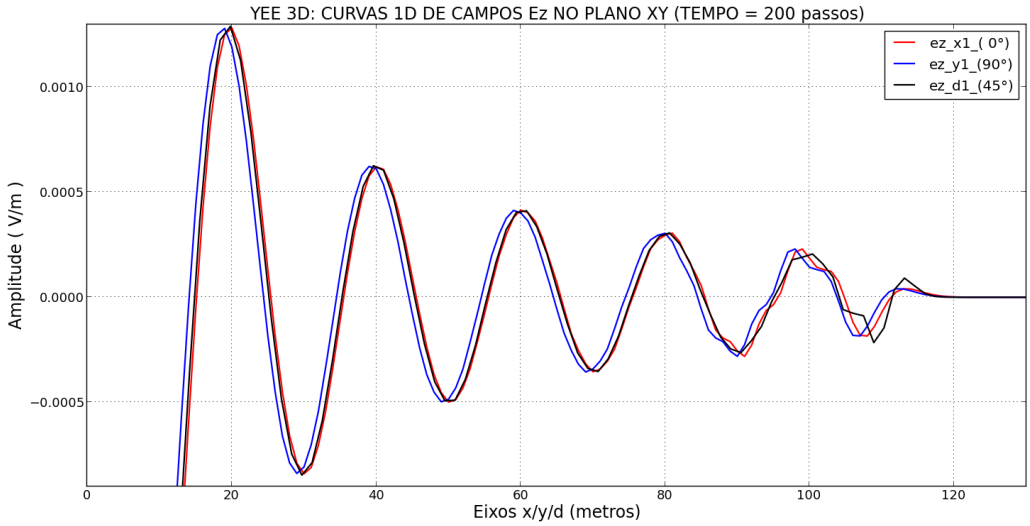
(b)

Figura 3.13 - Gráficos bidimensionais para o método FDTD com grade de prismas hexagonais: (a) plano x-y; (b) plano x-z. O nível máximo é reduzido por um fator de 10^{-4}, em ambos os gráficos, para facilitar a visualização.

Na Figura 3.13a, a partir do centro do plano x-y escolhe-se três direções positivas: x_1 (0°), y_1 (90°), d_1 (30°); e plota-se os níveis de campo elétrico E_Z nestas direções como mostrado na Figura 3.14a. Procede-se da mesma forma para o método FDTD Yee, embora o gráfico bidimensional deste método não seja mostrado neste texto, e escolhe-se as direções positivas x_1 (0°), y_1 (90°), d_1 (45°), plotando-se os gráficos de campo elétrico E_Z nestas direções, como mostrado na Figura 3.14b.



(a)



(b)

Figura 3.14 - Gráficos para fonte senoidal nas direções x_1, y_1, d_1 no plano x-y: (a) prismas hexagonais; (b) células cúbicas (método Yee).

Observa-se visualmente que a anisotropia de velocidade de fase numérica é bem menor na Figura 3.14a e os níveis de campo elétrico E_Z são muito próximos em ambas as figuras. Teoricamente, para as células hexagonais no plano x-y (LIU, 1996), as direções que seguem os lados do hexágono maior (como mostrado na Figura 3.1), isto é, as direções y_1 (90°) e d_1 (30°) correspondem à máxima velocidade de fase numérica, e a direção x_1 (0°) à mínima velocidade de fase numérica.

Da mesma forma como foi feito com o pulso *Ricker*, para uma observação mais acurada gera-se gráficos unidimensionais a partir dos gráficos bidimensionais de

ambos os métodos FDTD, de forma a comparar-se os níveis de campo elétrico E_Z para as três direções positivas (x_2, z_2, d_2), como mostram as Figuras 3.15, 3.16 e 3.17.

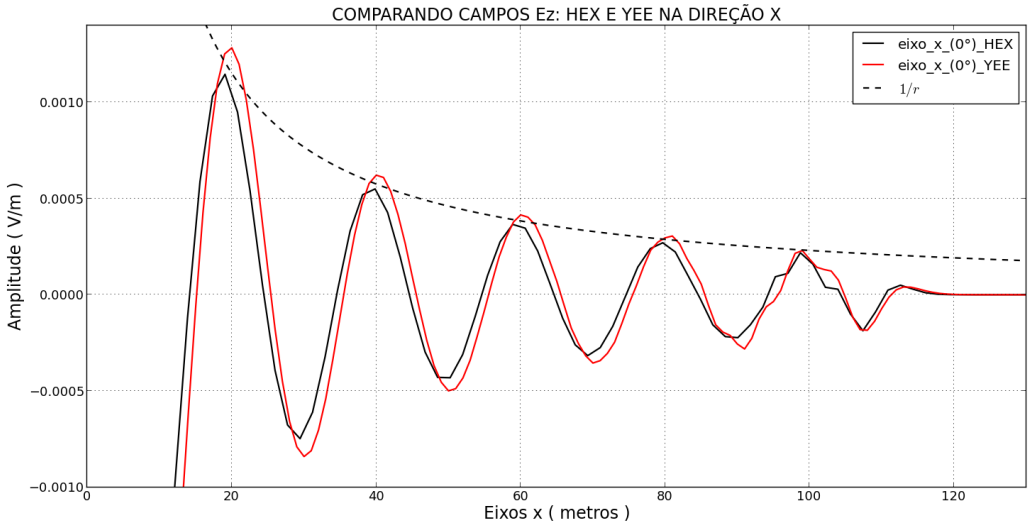


Figura 3.15 - Comparação de campos elétricos E_Z na direção x_2 (plano x-z), usando fonte senoidal, para os dois métodos FDTD.

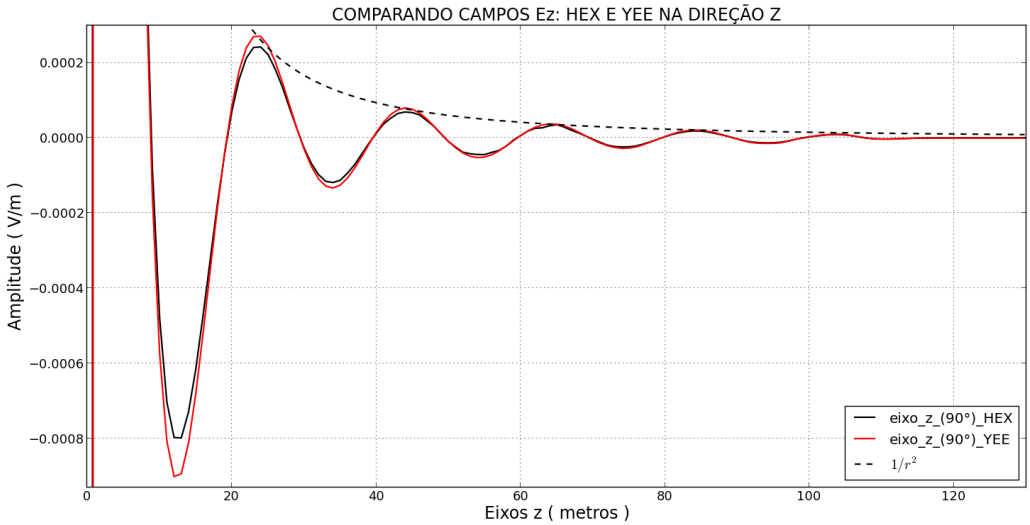


Figura 3.16. Comparação de campos elétricos E_Z na direção z_2 (plano x-z), usando fonte senoidal, para os dois métodos FDTD.

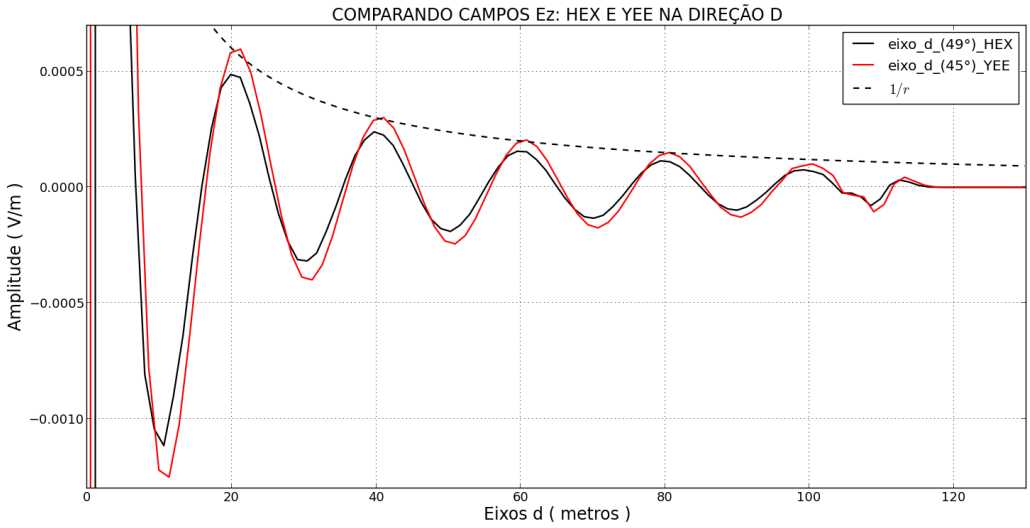


Figura 3.17. Comparação de campos elétricos E_Z na direção d_2 (plano x-z), usando fonte senoidal, para os dois métodos FDTD.

Observa-se, novamente, nas Figuras 3.15, 3.16 e 3.17 uma similaridade nas formas de onda entre os dois métodos FDTD, quando utilizando como fonte um sinal senoidal. As curvas nas direções *x* (Figura 3.15) e *d* (Figura 3.17) atenuam em função de (1 / *r*), enquanto as curvas na direção z (Figura 3.16) atenuam em função de (1 / r^2). Na Figura 3.17 as curvas tem níveis e fases próximos e similar comportamento em função da distância (1 / *r*), embora as suas direções não sejam exatamente iguais.

A partir das comparações de propagação de onda com o método FDTD Yee foi possível validar o método FDTD com prismas hexagonais para dois tipos distintos de fontes de sinal: fonte senoidal, que apresenta uma única frequência, e fonte do tipo pulso com uma larga faixa espectral de frequências. Observou-se visualmente uma menor anisotropia de velocidade de fase numérica no plano x-y, para o método FDTD com prismas hexagonais. Observa-se nas Figuras 3.15, 3.16 e 3.17 que o método FDTD com grade de prismas hexagonais é não-dissipativo, ou seja, não produz atenuação artificial (não-física), pois as ondas senoidais, desse método FDTD, têm as mesmas taxas de atenuação que as respectivas ondas senoidais do método (não-dissipativo) FDTD Yee (TAFLOVE; HAGNESS, 2005; INAN; MARSHALL, 2011).

Uma análise matemática precisa da anisotropia, observada nas figuras desta seção, será feita a seguir neste texto. Será provado, também, que a anisotropia numérica em outros planos, como por exemplo, planos x-z ou y-z na grade de prismas hexagonais, será menor que aquela do método FDTD Yee com células cúbicas.

4 ANÁLISES DE ANISOTROPIA E DISPERSÃO NUMÉRICAS DE GRADE FORMADA POR PRISMAS HEXAGONAIS UTILIZANDO O MÉTODO FDTD

Neste capítulo, usando análise de Fourier, determinar-se-á teoricamente as anisotropia e dispersão de velocidade de fase numéricas da grade tridimensional formada por prismas hexagonais, comparando os resultados obtidos com a grade de células cúbicas (ou hexaedros) do método FDTD Yee. Serão, também, feitas medidas de anisotropia de velocidade de fase numérica na grade de prismas hexagonais utilizando o método FDTD, que serão comparadas com os resultados da análise de Fourier (JOAQUIM; SCHEER, 2014).

4.1 ANISOTROPIA E DISPERSÃO DE VELOCIDADE DE FASE NUMÉRICAS DA GRADE DE PRISMAS HEXAGONAIS

Uma análise precisa das anisotropia e dispersão numéricas, da grade com prismas hexagonais, pode ser feita aplicando a análise de Fourier (LIU, 1996; INAN; MARSHALL, 2011) às equações de diferença finita, desta grade, obtidas da seção 3.2. Assim, utilizam-se os modos de Fourier discretos no espaço e tempo definidos a seguir como:

$$\boldsymbol{v} = \boldsymbol{v_0} \exp[\, J(\omega.n.\Delta t - k_X.i.\Delta x - k_Y.j.\Delta y - k_Z.k.\Delta z)\,] \tag{4.1}$$

Os incrementos no espaço, repetidos do capítulo 3 por conveniência, são da seguinte forma:

$$\Delta x = \Delta d.\cos 30° = \frac{\sqrt{3}}{2}.\Delta d \tag{4.2}$$

$$\Delta y = \frac{\Delta d}{2} \tag{4.3}$$

$$\Delta z = \frac{\Delta d}{R} \tag{4.4}$$

onde Δd é o lado do hexágono maior (ver Figura 3.1). Na equação (4.1) os índices *i*, *j*, *k* (números inteiros) correspondem aos passos espaciais nas direções (x, y, z),

respectivamente; $J = \sqrt{-1}$ é a unidade imaginária; k_X, k_Y e k_Z são as componentes do vetor número de onda (***k***) nas direções *x*, *y* e *z*, respectivamente; e ω é a frequência angular numérica na grade.

Os vetores ***v*** (componentes dos campos) e $\boldsymbol{v_0}$ (valores de pico constantes), na equação (4.1), são definidos como:

$$\boldsymbol{v} = [\, H_1 \;\; H_2 \;\; H_3 \;\; H_Z \;\; E_1 \;\; E_2 \;\; E_3 \;\; E_Z \,]^T \tag{4.5}$$

$$\boldsymbol{v_0} = [\, H_{10} \;\; H_{20} \;\; H_{30} \;\; H_{Z0} \;\; E_{10} \;\; E_{20} \;\; E_{30} \;\; E_{Z0} \,]^T \tag{4.6}$$

O vetor ***v***, como definido pelas equações (4.1), (4.5) e (4.6), será substituído nas oito equações de diferença finita (seção 3.2) da grade de prismas hexagonais. Para ilustrar o procedimento aplicar-se-á a equação (4.1) na equação de diferença finita do campo magnético H_1 da seção 3.2 (equação (3.27)), repetida aqui, por conveniência, como:

$$\begin{aligned} H_1\big|_{i,j-1,k}^{n+\frac{1}{2}} = \; & C_{H10}.H_1\big|_{i,j-1,k}^{n-\frac{1}{2}} + C_{HA}\left(E_z\big|_{i,j-2,k}^{n} - E_z\big|_{i,j,k}^{n}\right) + \\ & + C_{HB}\left(E_2\big|_{i+\frac{1}{6},j-\frac{3}{2},k+\frac{1}{2}}^{n} - E_2\big|_{i+\frac{1}{6},j-\frac{3}{2},k-\frac{1}{2}}^{n}\right) + \\ & + C_{HB}\left(E_3\big|_{i+\frac{1}{6},j-\frac{1}{2},k+\frac{1}{2}}^{n} - E_3\big|_{i+\frac{1}{6},j-\frac{1}{2},k-\frac{1}{2}}^{n}\right) \end{aligned} \tag{4.7}$$

com o coeficiente $C_{H10} = 1$. Assim, aplicando a equação (4.1) na equação (4.7), obtém-se:

$$-H_{10}\, exp\left[J\left(\omega.\left(n+\frac{1}{2}\right).\Delta t - k_X.i.\Delta x - k_Y.(j-1).\Delta y - k_Z.k.\Delta z\right)\right] +$$

$$+ H_{10}\, exp\left[J\left(\omega.\left(n-\frac{1}{2}\right).\Delta t - k_X.i.\Delta x - k_Y.(j-1).\Delta y - k_Z.k.\Delta z\right)\right] +$$

$$+ C_{HA}E_{Z0}\left\{\begin{array}{c} exp[J(\omega.n.\Delta t - k_X.i.\Delta x - k_Y.(j-2).\Delta y - k_Z.k.\Delta z)] - \\ - exp[J(\omega.n.\Delta t - k_X.i.\Delta x - k_Y.j.\Delta y - k_Z.k.\Delta z)] \end{array}\right\} +$$

$$+ C_{HB}E_{20}\left\{\begin{array}{l} exp\left[J\left(\omega.n.\Delta t - k_X.\left(i+\frac{1}{6}\right).\Delta x - k_Y.\left(j-\frac{3}{2}\right).\Delta y - \right.\right.\right. \\ \left.\left.\left. - k_Z.\left(k+\frac{1}{2}\right).\Delta z\right)\right] - \\ - exp\left[J\left(\omega.n.\Delta t - k_X.\left(i+\frac{1}{6}\right).\Delta x - k_Y.\left(j-\frac{3}{2}\right).\Delta y - \right.\right. \\ \left.\left. - k_Z.\left(k-\frac{1}{2}\right).\Delta z\right)\right] \end{array}\right\} +$$

$$+ C_{HB}E_{30}\left\{\begin{array}{l} exp\left[J\left(\omega.n.\Delta t - k_X.\left(i+\frac{1}{6}\right).\Delta x - k_Y.\left(j-\frac{1}{2}\right).\Delta y - \right.\right. \\ \left.\left. - k_Z.\left(k+\frac{1}{2}\right).\Delta z\right)\right] - \\ - exp\left[J\left(\omega.n.\Delta t - k_X.\left(i+\frac{1}{6}\right).\Delta x - k_Y.\left(j-\frac{1}{2}\right).\Delta y - \right.\right. \\ \left.\left. - k_Z.\left(k-\frac{1}{2}\right).\Delta z\right)\right] \end{array}\right\} = 0 \qquad (4.8)$$

A equação (4.8) apresenta um termo comum com os índices (*n*, *i*, *j* - 1, *k*), ou seja:

$$exp[J(\omega.n.\Delta t - k_X.i.\Delta x - k_Y.(j-1).\Delta y - k_Z.k.\Delta z)] \qquad (4.9)$$

Assim, dividindo a equação (4.8) pelo termo comum da equação (4.9), obtém-se:

$$H_{10}\left\{-exp\left[J\left(\omega.\frac{\Delta t}{2}\right)\right] + exp\left[-J\left(\omega.\frac{\Delta t}{2}\right)\right]\right\} +$$

$$+ C_{HA}E_{Z0}\{exp[J(k_Y.\Delta y)] - exp[-J(k_Y.\Delta y)]\} +$$

$$+ C_{HB}E_{20}\left\{exp\left[-J\left(k_X.\frac{\Delta x}{6} - k_Y.\frac{\Delta y}{2} + k_Z.\frac{\Delta z}{2}\right)\right] - exp\left[-J\left(k_X.\frac{\Delta x}{6} - k_Y.\frac{\Delta y}{2} - k_Z.\frac{\Delta z}{2}\right)\right]\right\} +$$

$$+ C_{HB}E_{30}\left\{exp\left[-J\left(k_X.\frac{\Delta x}{6} + k_Y.\frac{\Delta y}{2} + k_Z.\frac{\Delta z}{2}\right)\right] - exp\left[-J\left(k_X.\frac{\Delta x}{6} + k_Y.\frac{\Delta y}{2} - k_Z.\frac{\Delta z}{2}\right)\right]\right\} = 0 \qquad (4.10)$$

Aplicando a relação de Euler dada por: $e^{\pm J\Phi} = \cos\Phi \pm J\mathrm{sen}\Phi$; e usando as equações (4.2), (4.3) e (4.4) na equação (4.10), após simplificar termos comuns, obtém-se:

$$-H_{10}.sen\left(\omega.\frac{\Delta t}{2}\right) + C_{HA}E_{Z0}.sen\left(k_Y.\frac{\Delta d}{2}\right) -$$

$$- C_{HB}E_{20}exp\left[-J\left(\frac{\sqrt{3}}{12}\ k_X.\Delta d - \frac{1}{4}k_Y.\Delta d\right)\right].sen\left(k_Z.\frac{\Delta d}{2R}\right) -$$

$$- C_{HB}E_{30}exp\left[-J\left(\frac{\sqrt{3}}{12}\ k_X.\Delta d\ + \frac{1}{4}k_Y.\Delta d\right)\right].sen\left(k_Z.\frac{\Delta d}{2R}\right) = 0 \qquad (4.11)$$

O procedimento usado para obter a equação (4.11), a partir da equação de diferença finita para o campo magnético H_1, é repetido para as outras sete equações de diferença finita da grade de prismas hexagonais (da seção 3.2), considerando os coeficientes $C_{H20} = C_{H30} = C_{HZ0} = C_{E10} = C_{E20} = C_{E30} = C_{EZ0} = 1$, obtendo-se a seguinte relação:

$$\boldsymbol{M}.\boldsymbol{v_0} = 0 \qquad (4.12)$$

A matriz $\boldsymbol{M}$ é dada por:

$$\boldsymbol{M} = \begin{vmatrix} -S_0.\boldsymbol{I}_{4\times4} & \boldsymbol{A} \\ \boldsymbol{B} & -S_0.\boldsymbol{I}_{4\times4} \end{vmatrix} \qquad (4.13)$$

onde $\boldsymbol{I_{4\times4}}$ é a matriz identidade; as matrizes $\boldsymbol{A}$ e $\boldsymbol{B}$ são definidas como:

$$\boldsymbol{A} = \begin{vmatrix} 0 & -A_1 & -A_2 & C_{HA}.S_1 \\ A_1 & 0 & -A_3 & -C_{HA}.S_2 \\ A_2 & A_3 & 0 & -C_{HA}.S_3 \\ -C_{HZ}.S_1 & C_{HZ}.S_2 & C_{HZ}.S_3 & 0 \end{vmatrix} \qquad (4.14)$$

$$\boldsymbol{B} = \begin{vmatrix} 0 & B_1 & B_2 & -C_{EA}.S_1 \\ -B_1 & 0 & B_3 & C_{EA}.S_2 \\ -B_2 & -B_3 & 0 & C_{EA}.S_3 \\ C_{EZ}.S_1 & -C_{EZ}.S_2 & -C_{EZ}.S_3 & 0 \end{vmatrix} \qquad (4.15)$$

Os termos nas matrizes $\boldsymbol{M}$, $\boldsymbol{A}$ e $\boldsymbol{B}$ são dados por:

$$S_0 = \ \text{sen}\left(\frac{\omega.\Delta t}{2}\right) \qquad (4.16)$$

$$S_1 = \operatorname{sen}\left(\frac{k_Y.\Delta d}{2}\right) \tag{4.17}$$

$$S_2 = \operatorname{sen}\left(\frac{\sqrt{3}}{4}k_X.\Delta d - \frac{1}{4}k_Y.\Delta d\right) \tag{4.18}$$

$$S_3 = \operatorname{sen}\left(\frac{\sqrt{3}}{4}k_X.\Delta d + \frac{1}{4}k_Y.\Delta d\right) \tag{4.19}$$

$$S_Z = \operatorname{sen}\left(\frac{k_Z.\Delta d}{2R}\right) \tag{4.20}$$

$$A_1 = C_{HB}.exp\left[-J\left(\frac{\sqrt{3}}{12}k_X.\Delta d - \frac{1}{4}k_Y.\Delta d\right)\right].S_Z \tag{4.21}$$

$$A_2 = C_{HB}.exp\left[-J\left(\frac{\sqrt{3}}{12}k_X.\Delta d + \frac{1}{4}k_Y.\Delta d\right)\right].S_Z \tag{4.22}$$

$$A_3 = C_{HB}.exp\left[J\left(\frac{\sqrt{3}}{6}k_X.\Delta d\right)\right].S_Z \tag{4.23}$$

$$B_1 = C_{EB}.exp\left[J\left(\frac{\sqrt{3}}{12}k_X.\Delta d - \frac{1}{4}k_Y.\Delta d\right)\right].S_Z \tag{4.24}$$

$$B_2 = C_{EB}.exp\left[J\left(\frac{\sqrt{3}}{12}k_X.\Delta d + \frac{1}{4}k_Y.\Delta d\right)\right].S_Z \tag{4.25}$$

$$B_3 = C_{EB}.exp\left[-J\left(\frac{\sqrt{3}}{6}k_X.\Delta d\right)\right].S_Z \tag{4.26}$$

Os coeficientes C_{HA}, C_{HB}, C_{EA}, C_{EB}, C_{HZ} e C_{EZ} já foram definidos em equações anteriores (seção 3.2).

O determinante da matriz ***M*** pode ser calculado para um certo vetor número de onda ***k*** (k_X, k_Y, k_Z), e o resultado igualado a zero para obter S_0. Oito raízes (autovalores) são obtidas. Quatro dessas raízes são zero, duas raízes iguais negativas e duas raízes iguais positivas. A raiz positiva (S_{0+}) é a solução desejada, devido à convenção de sinais usada na equação (4.1). Neste ponto, é importante lembrar que ***k*** e ω são o vetor número de onda e a frequência angular, respectivamente, da grade de prismas hexagonais, e não do mundo real (físico). O

objetivo buscado é determinar as anisotropia e dispersão numéricas da grade; assim, por simplicidade, o módulo (*k*) do vetor número de onda ***k*** é aproximado por:

$$k \approx k_F \quad \text{e} \quad k_F = \frac{2\pi}{N_\lambda . \Delta d} \tag{4.27}$$

onde N_λ é o número de pontos por comprimento de onda. Assim, as componentes do número de onda k_F (k_X, k_Y, e k_Z), em coordenadas esféricas, conforme (PANARETOS; ABERLE; DÍAZ, 2006), são definidas como:

$$k_X = k_F . \cos \Phi \operatorname{sen} \theta \tag{4.28}$$

$$k_Y = k_F . \operatorname{sen} \Phi \operatorname{sen} \theta \tag{4.29}$$

$$k_Z = k_F . \cos \theta \tag{4.30}$$

O ângulo Φ é determinado a partir do eixo x (positivo) no plano x-y e o ângulo θ a partir do eixo z (positivo). Devido à aproximação usada no número de onda (k_F), a frequência angular encontrada não será necessariamente a da grade (ω). Esta frequência angular obtida será denominada de ω^*, de forma que, a partir da equação (4.16), tem-se:

$$\omega^* = \frac{2}{\Delta t} \operatorname{sen}^{-1}(S_{0+}) \tag{4.31}$$

A partir da equação do número de Courant da seção 3.2 (equação (3.31)), o incremento no tempo Δt é definido como:

$$\Delta t = \frac{S_{CFL} . \Delta d}{c} \tag{4.32}$$

onde *c* é a velocidade da onda no meio físico, como já definido na seção 3.1 (equação (3.7)). Logo, é possível definir uma velocidade c^*, que não será necessariamente a velocidade de propagação (v_{pn}) na grade, dada por:

$$c^* = \frac{\omega^*}{k_F} \tag{4.33}$$

Assim, a velocidade normalizada (c^* / c), aplicando as equações (4.27), (4.31) e (4.32) na equação (4.33) e dividindo pela velocidade de fase física (c), é definida como:

$$\frac{c^*}{c} = \frac{N_\lambda}{\pi . S_{CFL}} \text{sen}^{-1}(S_{0+}) \tag{4.34}$$

Usando a equação (4.34) com a equação (4.13) é possível determinar a velocidade normalizada (c^* / c) em função dos parâmetros N_λ, S_{CFL}, Δd, R e para qualquer direção desejada na grade, ou seja, em função dos ângulos Φ e θ, e, consequentemente, as anisotropia e dispersão numéricas da grade.

4.2 ANISOTROPIA E DISPERSÃO DE VELOCIDADE DE FASE NUMÉRICAS DA GRADE DE HEXAEDROS DO MÉTODO FDTD YEE

Uma análise de anisotropia e dispersão numéricas semelhante àquela desenvolvida para a grade de prismas hexagonais, pode ser aplicada para o método FDTD Yee. Utiliza-se uma equação semelhante à equação (4.1), que é aplicada nas equações de diferença finita do método FDTD Yee da subseção 2.3.7, como:

$$\boldsymbol{v_y} = \boldsymbol{v_{0y}} \exp[\, J(\omega . n . \Delta t - k_X . i . \Delta x - k_Y . j . \Delta y - k_Z . k . \Delta z)\,] \tag{4.35}$$

Os incrementos no espaço, obtidos das equações (2.110a) e (2.110b), são da seguinte forma:

$$\Delta y = \frac{\Delta x}{R_Y} \tag{4.36}$$

$$\Delta z = \frac{\Delta x}{R_Z} \tag{4.37}$$

Os vetores $\boldsymbol{v_y}$ (componentes dos campos) e $\boldsymbol{v_{0y}}$ (valores de pico constantes), na equação (4.35), são definidos como:

$$\boldsymbol{v_y} = [\, H_X \;\; H_Y \;\; H_Z \;\; E_X \;\; E_Y \; E_Z \,]^T \tag{4.38}$$

$$\boldsymbol{v_{0y}} = [\, H_{X0} \;\; H_{Y0} \;\; H_{Z0} \;\; E_{X0} \;\; E_{Y0} \;\; E_{Z0} \,]^T \tag{4.39}$$

O procedimento usado para obter a equação (4.11), a partir da equação de diferença finita para campo magnético H_1 (da grade de prismas hexagonais), é, também, aplicado às seis equações de diferença finita da grade de hexaedros do método FDTD Yee (da subseção 2.3.7), mas usando-se a equação (4.35). Assim, obtém-se a seguinte relação:

$$\boldsymbol{N}.\boldsymbol{v_{0y}} = 0 \tag{4.40}$$

A matriz ***N*** é dada por:

$$\boldsymbol{N} = \begin{vmatrix} -S_{0y}.\boldsymbol{I}_{3\times3} & \boldsymbol{C} \\ \boldsymbol{D} & -S_{0y}.\boldsymbol{I}_{3\times3} \end{vmatrix} \tag{4.41}$$

onde $\boldsymbol{I_{3\times3}}$ é a matriz identidade, e as matrizes ***C*** e ***D*** são definidas como:

$$\boldsymbol{C} = \begin{vmatrix} 0 & -C_{HZ}.S_Z & C_{HY}.S_Y \\ C_{HZ}.S_Z & 0 & -C_{HX}.S_X \\ -C_{HY}.S_Y & C_{HX}.S_X & 0 \end{vmatrix} \tag{4.42}$$

$$\boldsymbol{D} = \begin{vmatrix} 0 & C_{EZ}.S_Z & -C_{EY}.S_Y \\ -C_{EZ}.S_Z & 0 & C_{EX}.S_X \\ C_{EY}.S_Y & -C_{EX}.S_X & 0 \end{vmatrix} \tag{4.43}$$

Os termos nas matrizes ***N***, ***C*** e ***D*** são dados por:

$$S_{0y} = \operatorname{sen}\left(\frac{\omega.\Delta t}{2}\right) \tag{4.44}$$

$$S_X = \operatorname{sen}\left(\frac{k_X.\Delta x}{2}\right) \tag{4.45}$$

$$S_Y = \operatorname{sen}\left(\frac{k_Y.\Delta y}{2}\right) \tag{4.46}$$

$$S_Z = \operatorname{sen}\left(\frac{k_Z.\Delta z}{2}\right) \tag{4.47}$$

As equações dos coeficientes C_{HX}, C_{HY}, C_{HZ}, C_{EX}, C_{EY} e C_{EZ} já foram definidas na subseção 2.3.6 e são repetidas aqui, por conveniência:

$$C_{HX} = \frac{S_{CFL}}{Z} \tag{4.48}$$

$$C_{HY} = R_Y \frac{S_{CFL}}{Z} \tag{4.49}$$

$$C_{HZ} = R_Z \frac{S_{CFL}}{Z} \tag{4.50}$$

$$C_{EX} = S_{CFL}.Z \tag{4.51}$$

$$C_{EY} = R_Y\, S_{CFL}.Z \tag{4.52}$$

$$C_{EZ} = R_Z\, S_{CFL}.Z \tag{4.53}$$

Por simplicidade, foram usados nas equações de diferença finita do método FDTD Yee (da subseção 2.3.7) os coeficientes: $C_{EX0} = C_{EY0} = C_{EZ0} = C_{HX0} = C_{HY0} = C_{HZ0} = 1$.

O determinante da matriz ***N*** pode ser calculado para um certo vetor número de onda ***k*** (k_X, k_Y, k_Z), e o resultado igualado a zero para obter S_{0y}. Seis raízes (autovalores) são obtidas. Duas dessas raízes são zero, duas raízes iguais negativas e duas raízes iguais positivas. A raiz positiva (S_{0y+}) é a solução desejada, devido à convenção de sinais usada na equação (4.35). De uma forma semelhante ao procedimento feito para a grade de prismas hexagonais, da seção anterior, por simplicidade o módulo (k) do vetor número de onda ***k*** é aproximado por:

$$k \approx k_{FY} \quad \text{e} \quad k_{FY} = \frac{2\pi}{N_\lambda.\Delta x} \tag{4.54}$$

onde N_λ é o número de pontos por comprimento de onda, e o módulo do número de onda (k_{FY}) é específico ao método FDTD Yee. A partir da equação (4.54) a dedução

é a mesma realizada nas equações (4.28) a (4.34), obtendo-se a velocidade de fase numérica normalizada (c^* / c), para a grade do método FDTD Yee, como:

$$\frac{c^*}{c} = \frac{N_\lambda}{\pi . S_{CFL}} \text{sen}^{-1}\left(S_{0y+}\right) \tag{4.55}$$

Usando a equação (4.55) com a equação (4.41) é possível determinar a velocidade normalizada (c^* / c) em função dos parâmetros N_λ, S_{CFL}, Δx, R_Y, R_Z e para qualquer direção desejada na grade de células hexaédricas, ou seja, em função dos ângulos Φ e θ, e, consequentemente, as anisotropia e dispersão numéricas desta grade.

Ao invés de usar o procedimento descrito nesta seção, a velocidade normalizada poderia ser obtida diretamente da equação analítica, deduzida em (TAFLOVE; HAGNESS, 2005) para o método FDTD Yee, como:

$$\left[\frac{1}{c.\Delta t} sen\left(\frac{\omega.\Delta t}{2}\right)\right]^2 = \left[\frac{1}{\Delta x} sen\left(\frac{k_X.\Delta x}{2}\right)\right]^2 + \left[\frac{1}{\Delta y} sen\left(\frac{k_Y.\Delta y}{2}\right)\right]^2 + \left[\frac{1}{\Delta z} sen\left(\frac{k_Z.\Delta z}{2}\right)\right]^2 \tag{4.56}$$

Os resultados obtidos usando o procedimento descrito nas equações (4.35) a (4.55) desta seção são idênticos aos resultados da mais simples e concisa equação analítica (4.56). Desta forma confirma-se, também, a validade da análise de Fourier aplicada às equações de diferença finita da grade de prismas hexagonais como descrito na seção 4.1. Esta validação é importante, pois não foi possível obter uma equação analítica simples para a grade de prismas hexagonais, mesmo usando-se um solucionador analítico de equações (biblioteca *Sympy* para linguagem de programação *Python*). Isto se deve ao fato da matriz ***M***, usada na grade de prismas hexagonais, ter uma dimensão (8x8), tendo assim um número de termos no determinante da ordem de 56 vezes maior que o número de termos do determinante da matriz ***N***, usada na grade de células hexaédricas, cuja dimensão é (6x6).

4.3 ANÁLISE DE ESTABILIDADE APLICADA À GRADE DE PRISMAS HEXAGONAIS

A condição de estabilidade para o caso geral tridimensional do método FDTD Yee foi analisada na subseção 2.3.5. O número de Courant foi definido (mas não deduzido) na subseção 2.3.5 (equação (2.112)) como:

$$S_{CFL} \leq \frac{c.\Delta t}{\Delta x} = \frac{1}{\sqrt{1 + {R_Y}^2 + {R_Z}^2}} \tag{4.57}$$

onde: $R_Y = \Delta x / \Delta y$ e $R_Z = \Delta x / \Delta z$.

A rigor, o número de Courant da equação (4.57) deve ser deduzido a partir da equação (4.56), vista na seção anterior. Para fazer esta dedução, considera-se que a frequência angular numérica (ω) da equação (4.56) é um número complexo dado por:

$$\omega = \omega_{real} + J\omega_{imag} \tag{4.58}$$

Solucionando a equação (4.56) para a frequência angular numérica (ω) obtém-se:

$$\omega = \frac{2}{\Delta t}\text{sen}^{-1}(\xi) \tag{4.59}$$

com o argumento ξ definido como:

$$\xi = c.\Delta t\sqrt{\frac{1}{(\Delta x)^2}\text{sen}^2\left(\frac{k_X.\Delta x}{2}\right) + \frac{1}{(\Delta y)^2}\text{sen}^2\left(\frac{k_Y.\Delta y}{2}\right) + \frac{1}{(\Delta z)^2}\text{sen}^2\left(\frac{k_Z.\Delta z}{2}\right)} \tag{4.60}$$

Para que a condição de estabilidade seja alcançada, conforme (TAFLOVE; HAGNESS, 2005), é necessário que $\omega_{imag} = 0$, ou seja, $\omega = \omega_{real}$. O que é conseguido quando o argumento ξ da equação (4.59) estiver na faixa: $0 \leq \xi \leq 1$. O valor limite máximo do argumento ξ é obtido quando todos os termos $\text{sen}^2(...)$ sob a raiz quadrada da equação (4.60) alcançam simultaneamente o valor igual a um. Isto ocorre para as seguintes componentes do vetor número de onda $\boldsymbol{k}$:

$$k_X = \pm\frac{\pi}{\Delta x}\ ; \qquad k_Y = \pm\frac{\pi}{\Delta y}\ ; \qquad k_Z = \pm\frac{\pi}{\Delta z} \tag{4.61}$$

Logo, aplicando os valores da equação (4.61) na equação (4.60) obtém-se o seguinte resultado:

$$\xi = 1 = c.\Delta t\sqrt{\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2} + \frac{1}{(\Delta z)^2}} \tag{4.62}$$

Rearranjando os termos da equação (4.62), obtém-se o número de Courant como:

$$S_{CFL} \leq \frac{c.\Delta t}{\Delta x} = \frac{1}{\sqrt{1+\left(\frac{\Delta x}{\Delta y}\right)^2+\left(\frac{\Delta x}{\Delta z}\right)^2}} = \frac{1}{\sqrt{1 + {R_Y}^2 + {R_Z}^2}} \tag{4.63}$$

O número de Courant (S_{CFL}) para o método FDTD Yee obtido com a equação (4.63) é o mesmo da equação (4.57). Este mesmo raciocínio pode ser aplicado aos termos presentes na equação (4.13) da matriz ***M*** (método FDTD com grade de prismas hexagonais). Assim, utiliza-se as seguintes componentes do vetor número de onda ***k***:

$$k_X = \pm\frac{\sqrt{3}.\pi}{\Delta d}\ ; \qquad k_Y = \pm\frac{\pi}{\Delta d}\ ; \qquad k_Z = \pm\frac{R.\pi}{\Delta d} \tag{4.64}$$

Aplicando estes valores de número de onda nos termos da matriz ***M*** (equações (4.17) a (4.26)) obtém-se:

$$S_1 = S_2 = S_3 = S_Z = 1 \tag{4.65a}$$

$$A_1 = C_{HB}\ ;\quad A_2 = -JC_{HB}\ ;\quad A_3 = JC_{HB} \tag{4.65b}$$

$$B_1 = C_{EB}\ ;\quad B_2 = JC_{EB}\ ;\quad B_3 = -JC_{EB} \tag{4.65c}$$

Os coeficientes C_{HA}, C_{HB}, C_{HZ}, C_{EA}, C_{EB} e C_{EZ}, já foram definidos na seção 3.2 e são repetidos aqui, por conveniência, como:

$$C_{HA} = \frac{S_{CFL}}{Z}\ ; \qquad C_{HB} = \frac{R}{\sqrt{3}}\frac{S_{CFL}}{Z}\ ; \qquad C_{HZ} = \frac{2}{3}\frac{S_{CFL}}{Z} \tag{4.66a}$$

$$C_{EA} = S_{CFL}.Z\ ;\quad C_{EB} = \frac{R}{\sqrt{3}}S_{CFL}.Z\ ;\quad C_{EZ} = \frac{2}{3}S_{CFL}.Z \tag{4.66b}$$

Aplicam-se os valores obtidos nas equações (4.65) nas matrizes ***A*** e ***B*** das equações (4.14) e (4.15), respectivamente. Estas matrizes ***A*** e ***B*** são simplificadas para as seguintes formas:

$$\boldsymbol{A} = \begin{vmatrix} 0 & -C_{HB} & J.C_{HB} & C_{HA} \\ C_{HB} & 0 & -J.C_{HB} & -C_{HA} \\ -J.C_{HB} & J.C_{HB} & 0 & -C_{HA} \\ -C_{HZ} & C_{HZ} & C_{HZ} & 0 \end{vmatrix} \tag{4.67}$$

$$\boldsymbol{B} = \begin{vmatrix} 0 & C_{EB} & JC_{EB} & -C_{EA} \\ -C_{EB} & 0 & -JC_{EB} & C_{EA} \\ -J.C_{EB} & JC_{EB} & 0 & C_{EA} \\ C_{EZ} & -C_{EZ} & -C_{EZ} & 0 \end{vmatrix} \tag{4.68}$$

Aplicando estas formas simplificadas das matrizes ***A*** e ***B*** na matriz ***M***, e usando um solucionador simbólico de equações (como por exemplo: a biblioteca *Sympy* para linguagem de programação *Python*) obtém-se, para a matriz ***M***, oito autovalores simbólicos que são as soluções para a variável S_0:

$$S_{01} = S_{02} = \left\{ \frac{1}{2} \left[(-2S_{CFL}^2 - S_{CFL}^2 R^2)^2 - \frac{8}{9} S_{CFL}^4 R^2 \right]^{\frac{1}{2}} + S_{CFL}^2 + \frac{1}{2} S_{CFL}^2 R^2 \right\}^{\frac{1}{2}} \tag{4.69a}$$

$$S_{03} = S_{04} = -S_{01} \tag{4.69b}$$

$$S_{05} = S_{06} = \left\{ -\frac{1}{2} \left[(-2S_{CFL}^2 - S_{CFL}^2 R^2)^2 - \frac{8}{9} S_{CFL}^4 R^2 \right]^{\frac{1}{2}} + S_{CFL}^2 + \frac{1}{2} S_{CFL}^2 R^2 \right\}^{\frac{1}{2}} \tag{4.69c}$$

$$S_{07} = S_{08} = -S_{05} \tag{4.69d}$$

Os autovalores corretos são os quatro primeiros (S_{01} a S_{04}), pois produziram um número de Courant $S_{CFL} < 1$, enquanto os quatro últimos autovalores (S_{05} a S_{08}) não satisfazem a condição de estabilidade, pois produziram um número de Courant $S_{CFL} > 1$. Para que haja estabilidade numérica a seguinte condição deve ser satisfeita:

$$|S_{01}| \leq 1 \quad \Rightarrow \quad (\pm S_{01})^2 = (\pm 1)^2 \quad \Rightarrow \quad (S_{01})^2 = 1 \tag{4.70}$$

A seguir, soluciona-se a equação (4.70) analiticamente para a variável S_{CFL} (número de Courant), obtendo-se uma equação definida como:

$$S_{CFL} \leq \frac{c.\Delta t}{\Delta d} = \sqrt{\frac{6}{\sqrt{9R^4 + 28R^2 + 36} + 3R^2 + 6}} \tag{4.71}$$

na qual a razão é: $R = \Delta d / \Delta z$. A equação analítica (4.71), para o número de Courant da grade de prismas hexagonais, é mais complexa que a equação (4.63) para o número de Courant do método FDTD Yee. Observa-se, também, que o número de Courant da equação (4.71), reduz-se para o número de Courant da grade bidimensional de hexágonos (LIU, 1996) quando a razão $R = 0$ ($\Delta z \rightarrow \infty$), ou seja: $S_{CFL} = 1 / \sqrt{2}$. Este resultado está correto e coerente com a formulação do método FDTD para grade bidimensional de hexágonos.

É possível comparar o número de Courant da grade de prismas hexagonais com aquele da grade de hexaedros (método Yee). É suficiente definir a razão $R_Y = 1$ na equação (4.63) e considerar também na equação (4.63) a razão $R_Z = R$, onde R é a razão usada na grade de prismas hexagonais. Os números de Courant (S_{CFL}) das equações (4.63) e (4.71) são plotados em função da razão R, como mostrado na Figura 4.1.

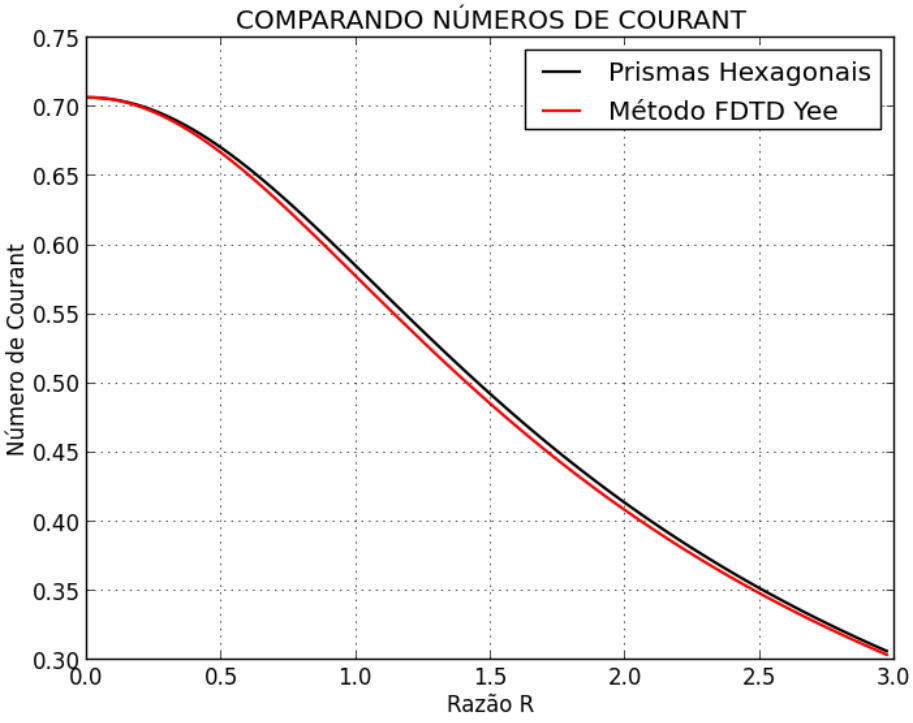


Figura 4.1 - Comparação dos números de Courant de ambos os métodos FDTD.

É observado na Figura 4.1, que a grade de prismas hexagonais tem um número máximo de Courant levemente maior que aquele da grade de hexaedros. Assim, é também possível usar para a grade de prismas hexagonais uma equação mais simples para número de Courant, a partir da equação (4.63), definida como:

$$S_{CFL} \leq \frac{c.\Delta t}{\Delta d} \approx \frac{1}{\sqrt{2+ R^2}} \tag{4.72}$$

4.4 COMPARAÇÃO DE ANISOTROPIA NUMÉRICA DE AMBOS OS MÉTODOS FDTD USANDO ANÁLISE DE FOURIER

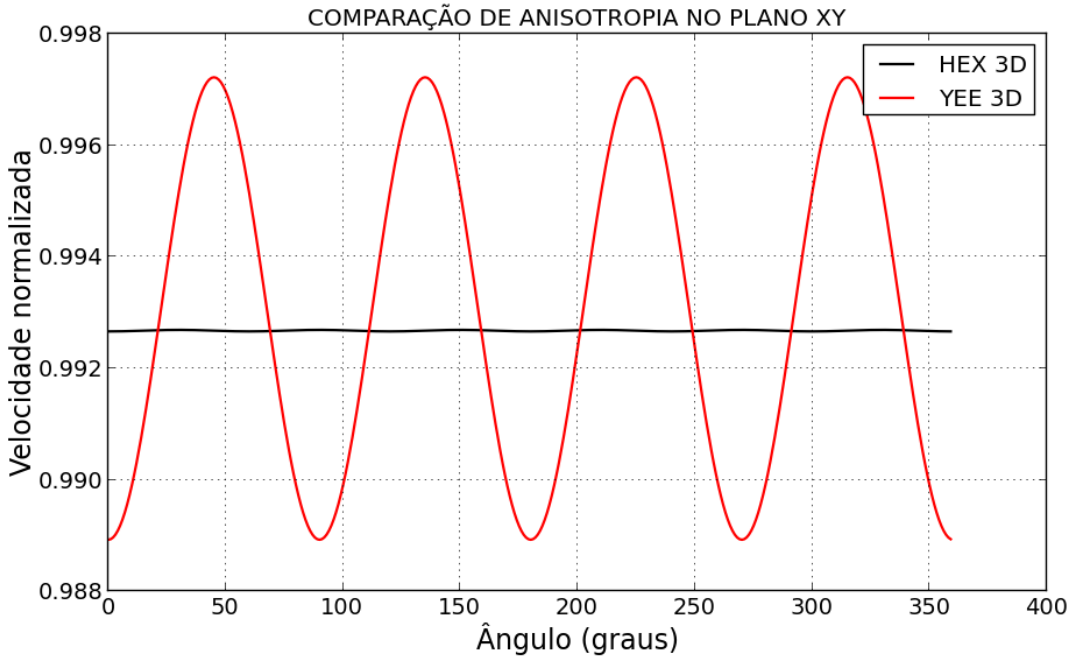
Para que comparações corretas de anisotropia e dispersão numéricas possam ser feitas entre ambos os métodos FDTD é necessário usar $\Delta x = \Delta y = \Delta z = \Delta$ na grade Yee, $\Delta d = \Delta$ na grade de prismas hexagonais e o mesmo parâmetro N_λ em ambos os métodos FDTD tal que os seus números de onda aproximados sejam iguais, isto é $k_F = k_{FY}$ (LIU, 1996).

A determinação da anisotropia de velocidade de fase numérica pode ser realizada para a grade de prismas hexagonais, usando a equação (4.34) da velocidade normalizada, juntamente com a determinação da raiz (ou autovalor) S_{0+} da matriz ***M***, dada pela equação (4.13). Da mesma forma, pode ser realizado o cálculo da anisotropia de velocidade de fase numérica, para grade de células cúbicas, usando a equação (4.55) da velocidade normalizada, juntamente com a determinação da raiz (ou autovalor) S_{0y+} da matriz ***N***, dada pela equação (4.41). A anisotropia numérica máxima é definida em função das velocidades normalizadas máxima e mínima como:

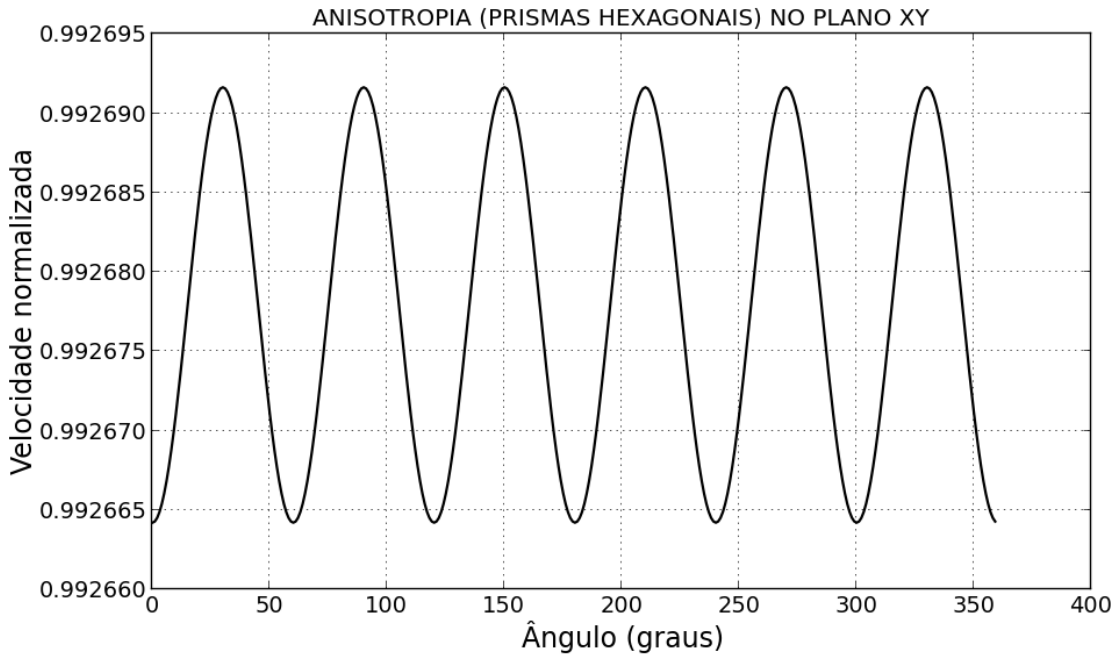
$$\Delta v_{anisotropia}(\%) = \frac{max[c^*(\phi)/c] - min[c^*(\phi)/c]}{min[c^*(\phi)/c]}.100 \tag{4.73}$$

A anisotropia numérica máxima depende principalmente do número de pontos por comprimento de onda (N_λ) e muito fracamente do número de Courant (S_{CFL}) (TAFLOVE; HAGNESS, 2005). A seguir compara-se as curvas de velocidade numérica normalizada para os dois métodos FDTD, para os planos x-y, x-z e y-z, como mostrado nas Figuras 4.2, 4.3 e 4.4, respectivamente. Utiliza-se, para ambos os métodos FDTD, número de pontos por comprimento de onda $N_\lambda = 10$. Para grade de prismas hexagonais usa-se: número de Courant S_{CFL} = 1 / √3,33333209;

$R = \Delta d / \Delta z = 1{,}1547$ e $\Delta d = 1$ m. Para o método FDTD Yee usa-se: número de Courant $S_{CFL} = 1 / \sqrt{3}$; $\Delta x = \Delta y = \Delta z = 1$ m.



(a)



(b)

Figura 4.2 - Comparação de curvas de velocidade numérica normalizada no plano x-y ($N_\lambda = 10$): (a) prismas hexagonais e o método FDTD Yee; (b) Unicamente grade de prismas hexagonais.

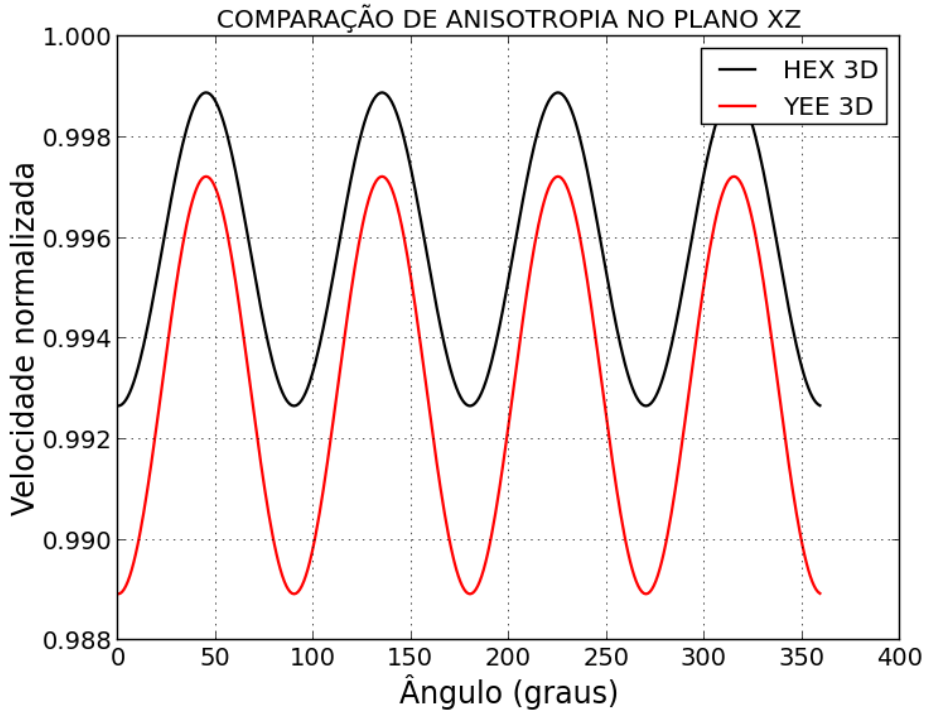


Figura 4.3 - Comparação de curvas de velocidade numérica normalizada no plano x-z ($N_\lambda = 10$): prismas hexagonais e o método FDTD Yee.

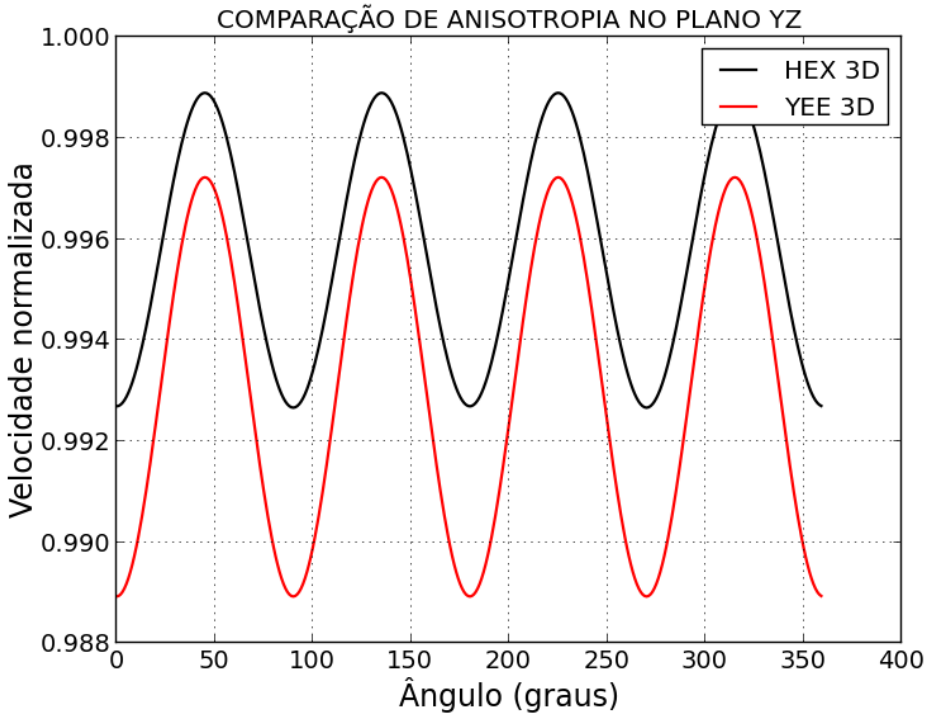


Figura 4.4 - Comparação de curvas de velocidade numérica normalizada no plano y-z ($N_\lambda = 10$): prismas hexagonais e o método FDTD Yee.

Observa-se na Figura 4.2, que a anisotropia de velocidade de fase numérica da curva de velocidade normalizada no plano x-y da grade de prismas hexagonais é da ordem de 300 vezes menor que aquela do método FDTD Yee. Nas Figuras 4.3 e 4.4 a diferença entre a anisotropia numérica, nos planos x-z ou y-z, de ambos os métodos FDTD é bem menor. No entanto, será provado a seguir que a anisotropia numérica máxima do método FDTD com prismas hexagonais, nos planos x-z ou y-z, é cerca de 25% menor que aquela do método FDTD Yee. Para provar isto, plota-se curvas de anisotropia numérica máxima (em %) em função da razão $R = \Delta d / \Delta z$, para grade de prismas hexagonais, e em função da razão $R = \Delta x / \Delta z$ (com $\Delta y = \Delta x$) para grade de hexaedros, respectivamente. Estas curvas de anisotropia numérica máxima são plotadas para os planos x-z e y-z nas Figuras 4.5 e 4.6, respectivamente, usando número de pontos por comprimento de onda: $N_\lambda = 10$.

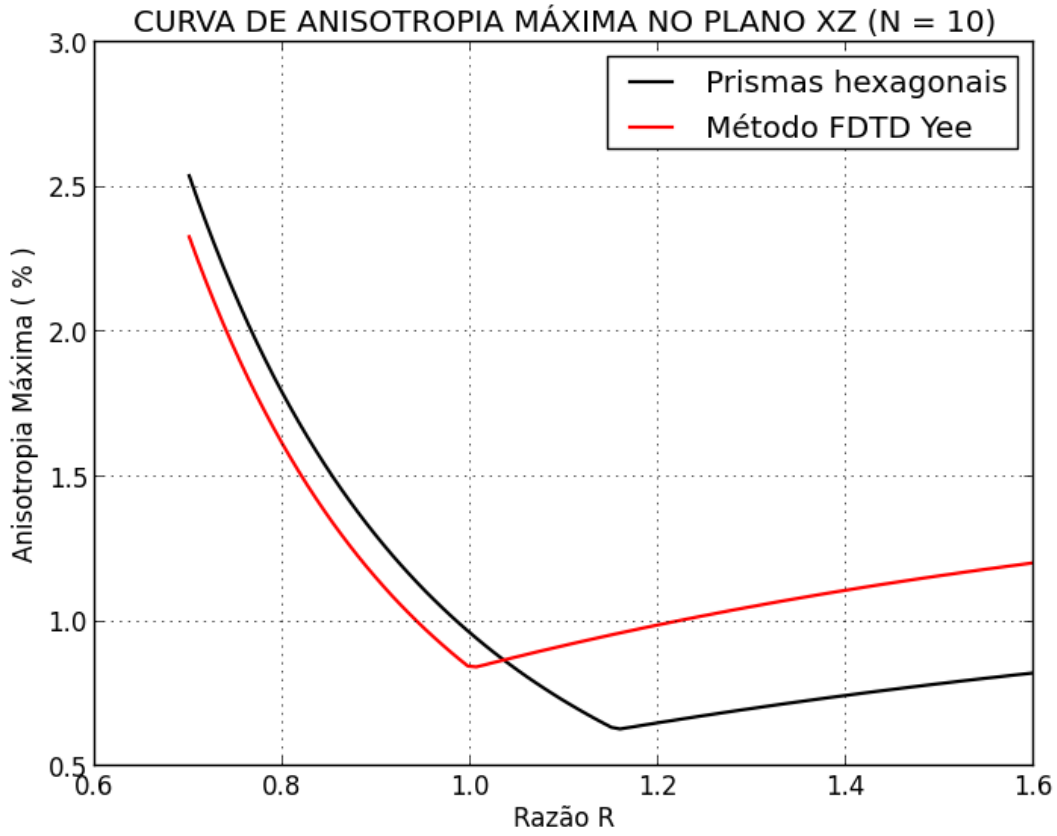


Figura 4.5 - Comparação de curvas de anisotropia numérica máxima (em %) no plano x-z: prismas hexagonais e o método FDTD Yee.

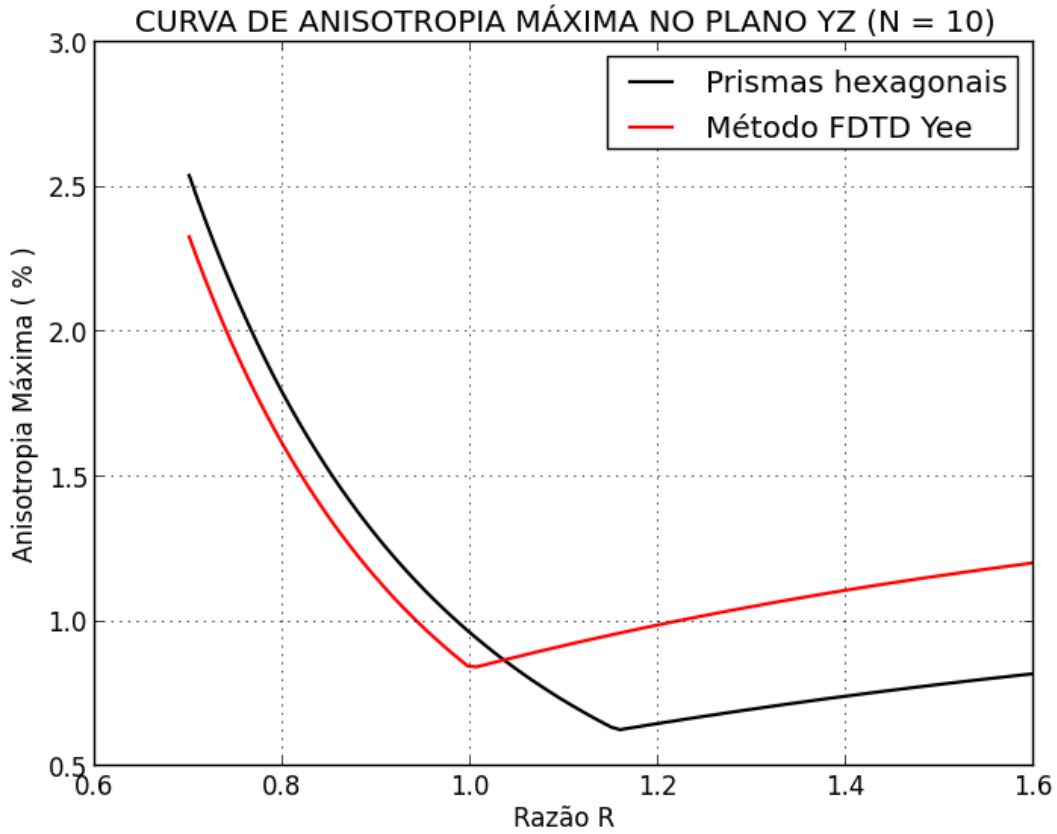


Figura 4.6 - Comparação de curvas de anisotropia numérica máxima (em %) no plano y-z: prismas hexagonais e o método FDTD Yee.

Observa-se nas Figuras 4.5 ou 4.6 que a grade de prismas hexagonais tem um mínimo de anisotropia máxima para uma razão $R = \Delta d / \Delta z \approx 1{,}16$; enquanto que, para a grade de hexaedros, esta razão é $R = \Delta x / \Delta z = 1$. Comparando os dois pontos de mínimo verifica-se que a grade de prismas hexagonais tem uma anisotropia máxima cerca de 25% menor que aquela da grade do método FDTD Yee nos planos x-z ou y-z. É importante observar que as curvas de anisotropia de cada método FDTD são iguais nos dois planos x-z e y-z.

O valor da razão $R = \Delta d / \Delta z \approx 1{,}16$ que minimiza a anisotropia numérica máxima para o plano x-z ou y-z da grade de prismas hexagonais, tem um significado geométrico: corresponde ao incremento de passo (Δz) na direção do eixo z, definido como:

$$\Delta z = \Delta x = \Delta d . \cos 30° \;\Rightarrow\; R = \frac{\Delta d}{\Delta z} = \frac{1}{\cos 30°} = \frac{2}{\sqrt{3}} \approx 1{,}1547 \approx 1{,}16 \qquad (4.74)$$

Ou seja, as células no plano x-z da grade de prismas hexagonais devem ter formato exatamente quadrado ($\Delta z = \Delta x$), para minimizar a anisotropia máxima neste plano. Isto também ocorre para o método FDTD Yee com: $R = \Delta x / \Delta z = 1$ e $\Delta y = \Delta x$.

Neste ponto, é útil comparar os resultados de anisotropia numérica utilizando a análise de Fourier com as fórmulas aproximadas de anisotropia numérica máxima, para o método FDTD Yee com grade bidimensional (TAFLOVE; HAGNESS, 2005), e para o método FDTD com grade bidimensional de hexágonos (LIU, 1996), definidas respectivamente como:

$$\Delta v_{YEE\_2D}(\%) \approx \left(\frac{\pi}{N_\lambda}\right)^2 \frac{100}{12} \tag{4.75}$$

$$\Delta v_{HEX\_2D}(\%) \approx \left(\frac{\pi}{N_\lambda}\right)^4 \frac{100}{360} \tag{4.76}$$

É possível, assim, comparar a anisotropia numérica máxima dos planos x-y, x-z e y-z: para o método FDTD com prismas hexagonais, o método FDTD Yee, e as fórmulas aproximadas dadas pelas equações (4.75) e (4.76). Isto é feito na Tabela 4.1. Considera-se para grade de prismas hexagonais $R = \Delta d / \Delta z = 1{,}1547$ e $\Delta d = 1$ m; número de Courant $S_{CFL} = 1 / \sqrt{3{,}33333209}$. Para o método FDTD Yee usa-se: $\Delta x = \Delta y = \Delta z = 1$ m; número de Courant $S_{CFL} = 1 / \sqrt{3}$. Considera-se, para ambos os métodos FDTD, número de pontos por comprimento de onda $N_\lambda = 10$.

TABELA 4.1 - COMPARAÇÃO DE ANISOTROPIAS MÁXIMAS ($N_\lambda = 10$)

Anisotropia Máxima (%)	Prismas Hexagonais	Método FDTD Yee	Formula para Hexágonos Bidimensionais	Formula para FDTD Yee Bidimensional
Plano x-y	0,0027615	0,8391132	0,0027058	0,8224670
Plano x-z	0,6272352	0,8391132	----------	0,8224670
Plano y-z	0,6275807	0,8391132	----------	0,8224670

Observa-se na Tabela 4.1 que a anisotropia máxima da grade de prismas hexagonais no plano x-y é apenas 2,06% maior que o valor obtido com a fórmula aproximada da equação (4.76); e a anisotropia máxima do método FDTD Yee, nos três planos, é apenas 2,02% maior que o valor obtido com a fórmula aproximada da equação (4.75). A anisotropia máxima no plano x-y da grade de prismas hexagonais é

aproximadamente 300 vezes menor que aquela do método FDTD Yee. A anisotropia máxima para grade de prismas hexagonais nos planos x-z e y-z, têm valores 25,25% e 25,20% menores que aqueles do método FDTD Yee, respectivamente. Uma outra tabela (Tabela 4.2) é feita, apenas alterando o número de pontos por comprimento de onda (N_λ) para 20.

TABELA 4.2 - COMPARAÇÃO DE ANISOTROPIAS MÁXIMAS (N_λ = 20)

Anisotropia Máxima (%)	Prismas Hexagonais	Método FDTD Yee	Formula para Hexágonos Bidimensionais	Formula para FDTD Yee Bidimensional
Plano x-y	0,0001699	0,2066484	0,0001691	0,2056167
Plano x-z	0,1548566	0,2066484	----------	0,2056167
Plano y-z	0,1548778	0,2066484	----------	0,2056167

Observa-se na Tabela 4.2 que a anisotropia máxima da grade de prismas hexagonais no plano x-y é apenas 0,51% maior que o valor obtido com a fórmula aproximada da equação (4.76); e que a anisotropia máxima do método FDTD Yee, nos três planos, é apenas 0,5% maior que o valor obtido com a fórmula aproximada da equação (4.75). A anisotropia máxima no plano x-y da grade de prismas hexagonais é aproximadamente 1200 vezes menor que aquela do método FDTD Yee. A anisotropia máxima para grade de prismas hexagonais nos planos x-z e y-z, tem valores 25,06% e 25,05% menores que aqueles do método FDTD Yee, respectivamente; estes valores são semelhantes aos obtidos na Tabela 4.1. Observa-se nas Tabelas 4.1 e 4.2 que a anisotropia numérica, para o método FDTD com prismas hexagonais, reduz-se de dezesseis vezes (no plano x-y) e de quatro vezes (nos planos x-z e y-z) quando dobra-se o número de pontos por comprimento de onda (N_λ). Para o método FDTD Yee, a anisotropia numérica reduz-se de quatro vezes (nos planos x-y, x-z e y-z) quando dobra-se o número de pontos por comprimento de onda (N_λ).

4.5 COMPARAÇÃO DE DISPERSÃO NUMÉRICA DE AMBOS OS MÉTODOS FDTD USANDO ANÁLISE DE FOURIER

A dispersão numérica é calculada em função da velocidade numérica mínima exata (min[v_{pn}]) e a velocidade física (c) da onda. Nesta tese, sem perda significativa de precisão para parâmetro $N_\lambda \geq 10$ (TAFLOVE; HAGNESS, 2005), a velocidade

numérica exata (v_{pn}) pode ser substituída pela velocidade numérica aproximada (c^*). Consequentemente, a dispersão numérica é definida como:

$$\Delta v_{dispersão}(\%) = \frac{min[c^*(\phi)] - c}{c}.100 = (min[c^*(\phi)/c] - 1).100 \quad (4.77)$$

A dispersão numérica, para ambos os métodos FDTD, é calculada em função da velocidade normalizada mínima usando a equação (4.77), e ela depende principalmente do parâmetro N_λ; mas deve ser usado o número máximo de Courant (S_{CFL}) de seu respectivo método FDTD para produzir dispersão numérica mínima para um certo parâmetro N_λ (TAFLOVE; HAGNESS, 2005). Em cada método FDTD, as velocidades normalizadas mínimas nos planos x-y, x-z e y-z, para um certo parâmetro N_λ são iguais com precisão de pelo menos sete casas decimais (para $N_\lambda \geq 10$). É importante lembrar que na propagação de um pulso em grades 2D ou 3D com resolução espacial definida, quanto maior a frequência de uma certa componente espectral deste pulso, menor é o parâmetro específico N_λ desta componente e, portanto, maior a dispersão numérica associada com esta componente. Na Tabela 4.3 é feita uma comparação de dispersão numérica entre ambos os métodos FDTD. Nesta comparação considera-se para grade de prismas hexagonais $R = \Delta d / \Delta z = 1{,}1547$ e $\Delta d = 1$ m; número de Courant $S_{CFL} = 1 / \sqrt{3{,}24193735}$ usando a equação exata (4.71). Para o método FDTD Yee usa-se: $\Delta x = \Delta y = \Delta z = 1$ m; número de Courant $S_{CFL} = 1 / \sqrt{3}$. Considera-se, para ambos os métodos FDTD, números de pontos por comprimento de onda $N_\lambda = 10$, 20 e 40.

TABELA 4.3 - COMPARAÇÃO DE DISPERSÃO NUMÉRICA (%) EM FUNÇAO DO PARÂMETRO N_λ.

Dispersão numérica (%) em função do parâmetro N_λ	Prismas Hexagonais	Método FDTD Yee
10	- 0,733579	- 1,107391
20	- 0,182031	- 0,274831
40	- 0,045422	- 0,068581

É observado na Tabela 4.3 que quanto maior o número de pontos por comprimento de onda N_λ, menor é a dispersão numérica. É também observado que a grade de prismas hexagonais tem em torno de 33% menos dispersão numérica que a grade de hexaedros (método Yee) para qualquer parâmetro N_λ. No entanto, em ambos os

métodos FDTD a dispersão numérica em função do parâmetro N_λ tem precisão de 2ª ordem, isto é, se o número de pontos por comprimento de onda (N_λ) dobrar, a dispersão numérica é reduzida em aproximadamente quatro vezes.

4.6 COMPROVAÇÃO DO USO DE COEFICIENTES CORRETOS NAS EQUAÇÕES DE DIFERENÇA FINITA DA GRADE DE PRISMAS HEXAGONAIS

Nesta seção será provado que a relação usada na equação (3.26b), entre as componentes de campo magnético H_{1E2} e H_{1E3} com o campo magnético H_1, é a melhor de todas. Para fazer esta comprovação é inserido o fator *f* na equação (3.26b), tal que:

$$\left.\frac{\partial \mu H_1}{\partial t}\right|_{i,j-1,k}^{n} = \frac{f}{\sqrt{3}}\left(\left.\frac{\partial \mu H_{1E2}}{\partial t}\right|_{i+\frac{1}{6},j-\frac{3}{2},k}^{n} + \left.\frac{\partial \mu H_{1E3}}{\partial t}\right|_{i+\frac{1}{6},j-\frac{1}{2},k}^{n}\right) \tag{4.78}$$

Este fator *f* multiplica os coeficientes C_{HA}, C_{HB}, C_{EA} e C_{EB} em seis das oito equações de diferença finita da grade de prismas hexagonais. É usado N_λ = 10, R = 1,1547, Δd = 1 m e número de Courant S_{CFL} = 1 / √3,33333209. A curva de anisotropia numérica máxima no plano x-z em função do fator *f* é plotada como mostrado na Figura 4.7.

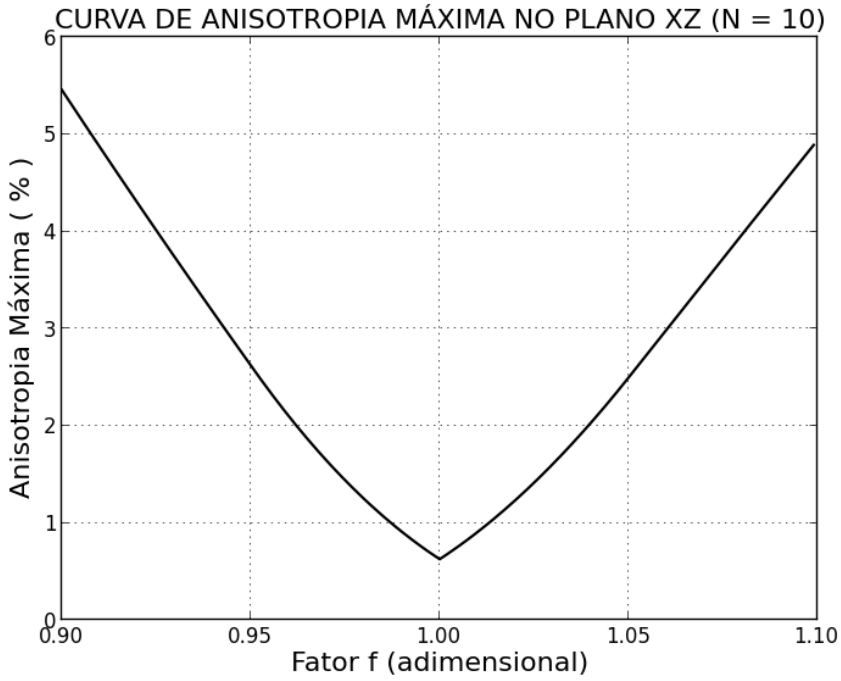


Figura 4.7 – Curva de anisotropia numérica máxima no plano x-z em função do fator *f* (N_λ = 10).

É observado nesta figura que o mínimo da curva de anisotropia numérica máxima no plano x-z é obtido para fator *f* = 1, assim confirmando a relação correta dada pela equação (3.26b). As curvas de velocidades normalizadas mínima e máxima no plano x-z em função do fator *f*, com os mesmos valores usados para plotar a Figura 4.7, são mostradas na Figura 4.8.

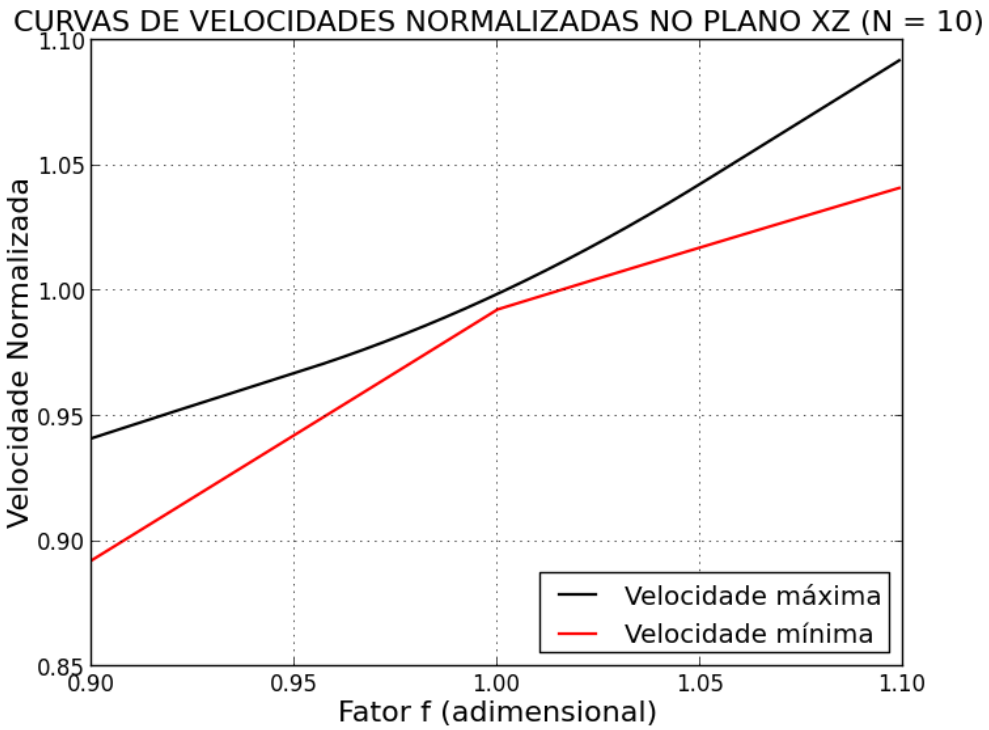


Figura 4.8 – Curvas de velocidades normalizadas mínima e máxima em função do fator *f* (N_λ = 10).

É observado na Figura 4.8 que quanto maior o fator *f* é em relação a um, mais dispersão numérica positiva (avanço de fase) é produzida; e quanto menor o fator *f* é em relação a um, mais dispersão numérica negativa (atraso de fase) é produzida. No entanto, os valores corretos de velocidades normalizadas mínima e máxima no plano x-z são obtidos para fator *f* = 1, que corresponde ao mínimo da curva de anisotropia numérica máxima (Figura 4.7). Consequentemente, é produzida uma leve dispersão numérica negativa pela curva de velocidade normalizada mínima para fator *f* = 1, como mostrado na Figura 4.8.

4.7 MEDIDAS DE ANISOTROPIA NA GRADE COM PRISMAS HEXAGONAIS

Nesta seção far-se-á as comparações entre os resultados teóricos de anisotropia numérica (obtidos da seção 4.1) e as medidas de anisotropia numérica realizadas nas simulações do método FDTD com grade de prismas hexagonais. Nas simulações com esse método FDTD, utilizou-se como fonte um sinal senoidal levemente atenuado (α_T = 1,8) no início da simulação (ver Figura 4.9), para minimizar oscilações e transitórios na frente de onda. Esta fonte senoidal é do tipo *hard*, e foi aplicada no centro da grade tridimensional para gerar o campo elétrico E_Z.

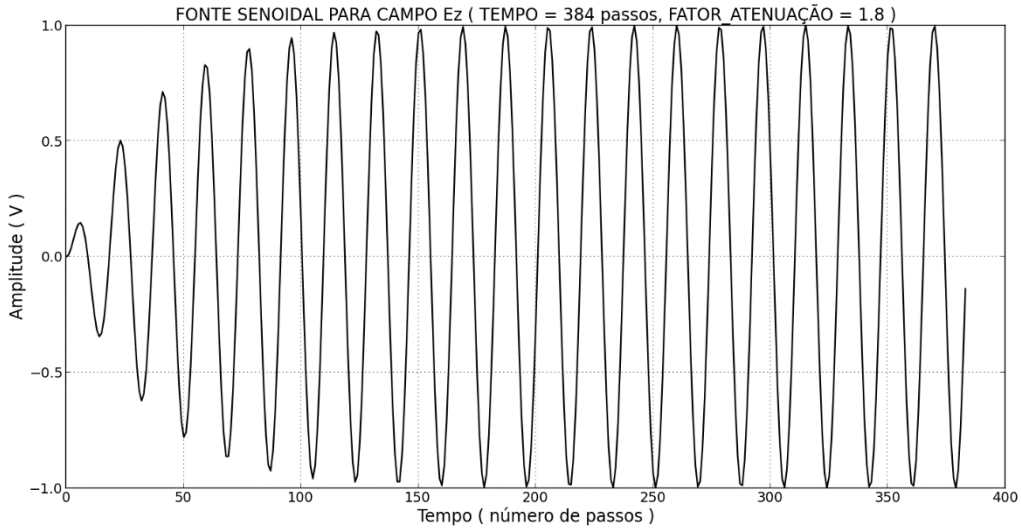


Figura 4.9 - Fonte de sinal senoidal para gerar o campo elétrico E_Z no centro da grade tridimensional de prismas hexagonais.

Nesta seção usa-se R = 1,1547, com S_{CFL} = 1 / √3,3333321 e também, por simplicidade, Δd = 1,0 m. Para fazer as medidas de velocidade de fase numérica da onda eletromagnética na grade foram escolhidos dois planos: o plano x-y que é paralelo aos hexágonos, e o plano x-z perpendicular aos hexágonos. As simulações foram feitas num espaço relativamente grande, com dimensões iguais nas três coordenadas cartesianas ($L_X = L_Y = L_Z \approx$ 540 m) e com número de pontos N_X = 625, N_Y = 1081 e N_Z = 625. O tempo de simulação deve estar abaixo de um certo valor (nº de passos × Δt), para que a onda senoidal não alcance o final do espaço de simulação e sofra reflexão.

Para o plano x-y, a partir do ponto central, são plotados gráficos do campo elétrico E_Z nas direções positivas x_1 (0°), y_1 (90°), d_1 (30°) e para o plano x-z, também

a partir do ponto central, nas direções positivas x_2 (0°), z_2 (90°) e d_2 (45°). Os gráficos são plotados para um certo tempo (384 passos) e para $N_\lambda = 10$. É escolhido para cada gráfico (x_1, y_1, d_1, x_2, z_2, d_2) dois pontos de nulo diferentes (posições A e B), como mostrados nas Figuras 4.10 e 4.11. A posição do ponto de nulo à direita (B), deve estar suficientemente atrás da frente de onda (alguns ciclos), para que variações de fase na frente de onda não produzam erros nas medidas (TAFLOVE; HAGNESS, 2005); e o ponto de nulo à esquerda (A), deve estar suficientemente afastado da origem da fonte, para maior precisão nas medidas.

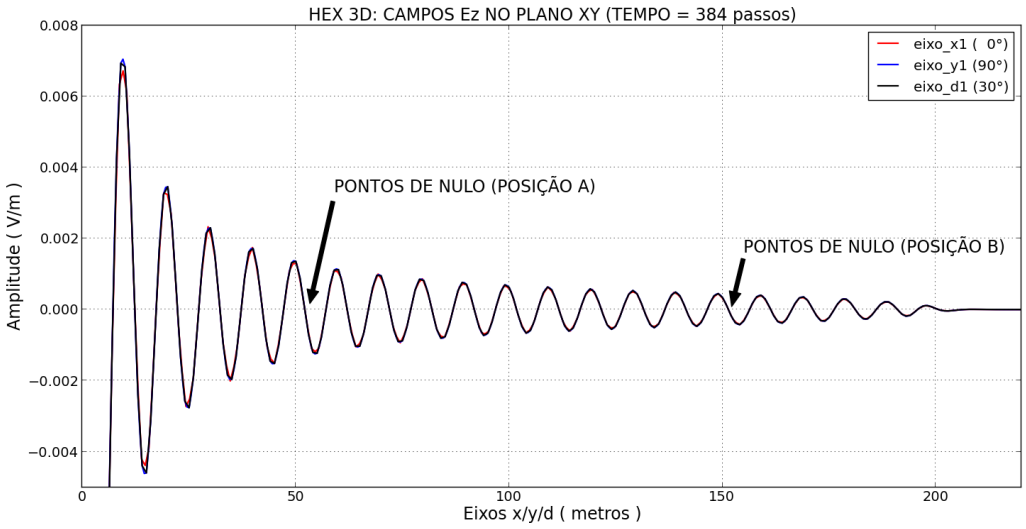


Figura 4.10 - Gráficos do campo elétrico E_Z no plano x-y para um tempo de 384 passos.

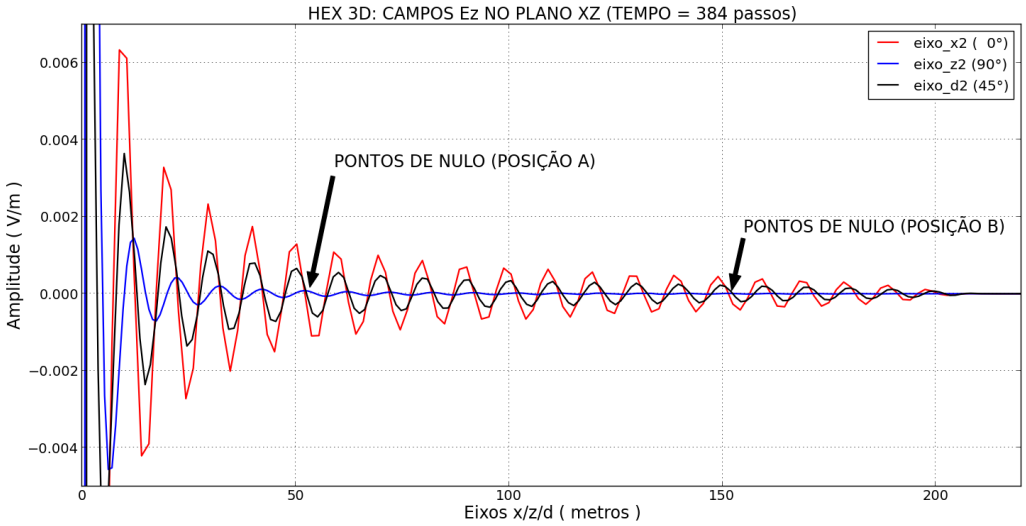


Figura 4.11 - Gráficos do campo elétrico E_Z no plano x-z para um tempo de 384 passos.

Para cada direção (x_1, y_1, d_1, x_2, z_2, d_2) um algoritmo numérico automaticamente calcula as posições A e B dos pontos de nulo do gráfico naquela direção. A anisotropia numérica (Δv_w) é a diferença (em %) entre a velocidade da onda na direção w (y_1, d_1 na Figura 4.10 ou z_2, d_2 na Figura 4.11) e a velocidade na direção de referência x (x_1 na Figura 4.10, ou x_2 na Figura 4.11) definida por:

$$\Delta v_w(\%) = \frac{(v_w - v_x).100}{v_x} = \frac{(\Delta w/\Delta t - \Delta x/\Delta t).100}{\Delta x/\Delta t} \tag{4.79}$$

A partir de (4.79), simplificando a variável tempo (Δt) implícita nos valores das velocidades v_w e v_x, obtém-se uma equação prática para o cálculo da anisotropia numérica (Δv_w) dada por:

$$\Delta v_w(\%) = \frac{[(w_B - w_A) - (x_B - x_A)].100}{(x_B - x_A)} \tag{4.80}$$

onde w_A e w_B indicam as posições A e B, respectivamente, dos pontos de nulo do gráfico na direção w. O mesmo raciocínio aplica-se a x_A e x_B na direção x.

A Tabela 4.4 compara a anisotropia numérica medida com o método FDTD na grade com prismas hexagonais utilizando a equação (4.80), com a formulação teórica da seção 4.1.

TABELA 4.4 - COMPARAÇÃO DE ANISOTROPIAS (N_λ = 10).

Direções	**Medida FDTD (%)**	**Análise de Fourier (%)**
Plano x-y: y_1 (90°)	0,0036819	0,0027615
Plano x-y: d_1 (30°)	0,0036134	0,0027615
Plano x-z: z_2 (90°)	- 0,0550591	- 0,0000011
Plano x-z: d_2 (45°)	0,6032132	0,6272340

Na Tabela 4.4 as medidas de anisotropia numérica com o método FDTD aplicado à grade de prismas hexagonais têm nas direções y_1 (90°) e d_1 (30°) valores que são 33,39% e 30,85% maiores, respectivamente, que aqueles obtidos com a análise de Fourier. No plano x-z a medida de anisotropia na direção z_2 (90°) é praticamente zero, se comparada com a direção d_2 (45°), e está de acordo com o valor da análise de Fourier, que teoricamente deve ser exatamente zero; na direção d_2 (45°) a medida de

anisotropia é apenas 3,83% menor que aquela obtida com a análise de Fourier. No entanto, apesar das medidas feitas com o método FDTD aplicado à grade de prismas hexagonais no plano x-y serem em torno de 30% maiores que aqueles da análise de Fourier, os valores medidos no plano x-y são muito menores que aqueles do método Yee (Tabela 4.1). Estas diferenças entre medidas práticas (método FDTD) e teóricas (análise de Fourier) são devido à necessidade de um espaço de simulação maior e, também, a uma diferença maior entre as posições A e B dos pontos de nulo de cada gráfico, para produzir resultados mais precisos; este comportamento foi observado nas simulações realizadas com menores espaços de simulação e menores diferenças entre as posições A e B dos pontos de nulo de cada gráfico.

5 CAMADAS DE ABSORÇÃO PARA O MÉTODO FDTD APLICADO À GRADE DE PRISMAS HEXAGONAIS

5.1 INTRODUÇÃO

O método FDTD é um método de diferenças finitas com marcha no tempo aplicado nas equações de Maxwell de forma a simular a propagação de ondas eletromagnéticas na grade tridimensional (3D). Para simular a propagação de onda em espaço aberto, o método FDTD necessita de uma forma de emular um espaço aberto ou infinito. Há basicamente duas formas principais de simular um espaço aberto. A primeira forma usa condições de contorno analíticas (ou matemáticas) nos contornos externos de grades 2D ou 3D e são conhecidas como ABCs (*Analytical Absorbing Boundary Conditions*) (TAFLOVE; HAGNESS, 2005). A rigor, estas ABCs são condições de contorno de radiação e não de absorção (INAN; MARSHALL, 2011). A segunda forma é inspirada no mundo real onde existem materiais que absorvem as ondas eletromagnéticas, os quais são muito usados em aplicações práticas como: blindagens, engenharia de antenas, câmaras anecóicas, bioengenharia, etc. Estes materiais possuem normalmente condutividade elétrica alta e espessura suficiente para atenuar (ou absorver) um certo espectro desejado de frequências.

Este mesmo conceito pode ser aplicado no método FDTD. Assim, por exemplo são usadas seis camadas finas de material com perdas em torno da grade FDTD 3D, que usualmente tem formato hexaédrico. A função dessas camadas será absorver as ondas incidentes e reduzir a um nível desprezível possíveis ondas refletidas na direção do interior da grade (TAFLOVE; HAGNESS, 2005). A primeira dificuldade encontrada, baseada na teoria eletromagnética (JIN, 2010), é que na interface entre um meio dielétrico (que pode corresponder ao interior da grade FDTD) e um meio com perdas, mas homogêneo e isotrópico, uma reflexão zero de uma onda plana incidente poderá ocorrer unicamente para incidência normal nesta interface. Para ângulos de incidência diferentes de zero em relação à normal à interface as reflexões de onda seriam de níveis não-desprezíveis, o que inviabiliza o uso deste simples material (homogêneo e isotrópico) com perdas como uma ABC para grades 2D ou 3D, embora funcione bem para grade 1D, mas que é de pouco interesse prático. No entanto, esta primeira dificuldade foi contornada por J. P. Bérenger em 1994 (BÉRENGER, 1994), que desenvolveu uma camada de absorção artificial (não-física) com características

anisotrópicas para condutividades elétrica e magnética, que teoricamente permite reflexão igual a zero para todos os ângulos de incidência válidos ($0° \leq \theta_i < 90°$) independentemente da frequência usada e absorção efetiva da onda dentro da camada artificial. Usando uma espessura suficiente para esta camada e valores adequados de condutividade elétrica e magnética para garantir casamento de impedância perfeito (reflexão zero) na interface com a grade FDTD, é possível garantir que a onda que penetra na camada seja suficiente atenuada até o contorno externo (final), onde sofrerá reflexão total, e novamente atenuada no seu trajeto de volta em direção à interface, que será atravessada sem reflexão, mas com nível muito reduzido e desprezível no interior da grade FDTD. Devido à propriedade de casamento de impedância perfeito (reflexão igual a zero) nas interfaces entre as camadas de absorção e a grade FDTD, que independe do ângulo de incidência e da frequência da onda, estas camadas de absorção são conhecidas em inglês como PMLs (*Perfectly Matched Layers*). A partir da bem-sucedida formulação PML de Bérenger, outras PMLs foram desenvolvidas, sendo as mais relevantes a *Uniaxial* PML (UPML) (GEDNEY, 1996) e a *Convolutional* PML (CPML) (RODEN; GEDNEY, 2000). Nesta tese será analisada, de forma breve para o método FDTD Yee, a formulação PML com campos divididos desenvolvida por Bérenger (BÉRENGER, 1994) e a formulação CPML. Atualmente, a formulação CPML está entre as melhores PMLs, e é obtida a partir das coordenadas estiradas (CHEW; WEEDON, 1994) derivadas da formulação de Bérenger, e usa uma eficiente convolução recursiva no tempo para o método FDTD. Esta base teórica será útil para desenvolver a formulação CPML aplicada à grade de prismas hexagonais.

5.2 FORMULAÇÃO PML DE BÉRENGER

O método de Bérenger para desenvolver uma PML consiste em usar campos divididos nas equações de Maxwell e escolher seletivamente valores diferentes de condutividades elétrica (σ) e magnética (σ_m). Considera-se uma interface ($x = 0$) no plano x-y composta de duas regiões, como mostrado na Figura 1. A região 1 (para $x < 0$) é composta por um material dielétrico, mas sem perdas. A região 2 ($x > 0$) é a PML com valores de perdas elétricas e magnéticas que não precisam ter significado físico real, mas cujos valores são escolhidos para maximizar o desempenho da PML.

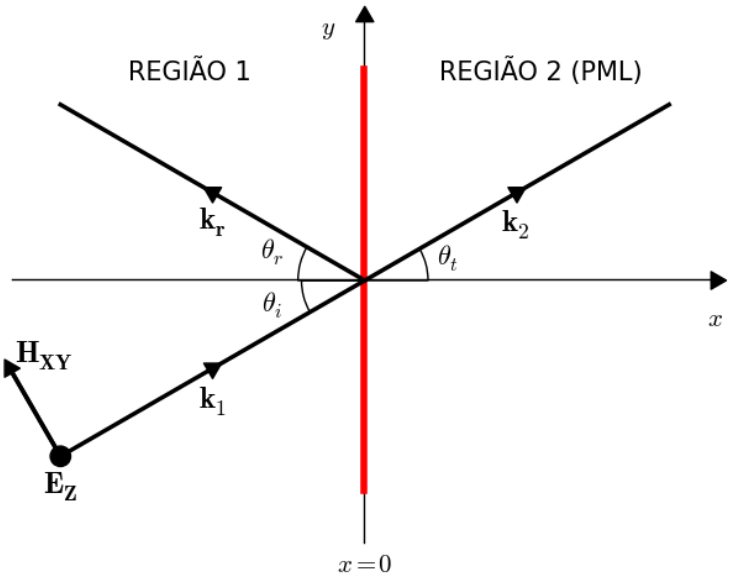


Figura 5.1 - Interface de incidência de campos elétrico e magnético entre regiões 1 e 2 (PML).

Na região 1, por exemplo, para grade 2D com o modo TM_Z (composto apenas de campos H_X, H_Y e E_Z) o campo elétrico incidente E^i_Z na forma fasorial (ver subseção 2.2.3), é dado por:

$$\boldsymbol{E^i} = \boldsymbol{k}.E_{Z0}.e^{-j\boldsymbol{k_1}.\boldsymbol{r}} \tag{5.1}$$

onde o vetor número de onda ($\boldsymbol{k_1}$) é definido para o ângulo de incidência (θ_i) como:

$$\boldsymbol{k_1} = \boldsymbol{i}.k_1\cos\theta_i + \boldsymbol{j}.k_1\,\mathrm{sen}\,\theta_i = \boldsymbol{i}.k_{1x} + \boldsymbol{j}.k_{1y} \tag{5.2}$$

e o vetor distância $\boldsymbol{r}$ é dado por:

$$\boldsymbol{r} = \boldsymbol{i}.x + \boldsymbol{j}.y \tag{5.3}$$

Aplicando equações (5.2) e (5.3) em (5.1) é encontrado:

$$\boldsymbol{E^i} = \boldsymbol{k}.E_{Z0}.e^{-jk_{1x}.x - jk_{1y}.y} \tag{5.4}$$

Na interface (x = 0) os ângulos de incidência (θ_i) e reflexão (θ_r) devem ser iguais tal que o campo elétrico refletido E^r_Z, se existir, é dado por:

$$\boldsymbol{E^r} = \boldsymbol{k}.\Gamma.E_{Z0}.e^{jk_{1x}.x - jk_{1y}.y} \tag{5.5}$$

onde Γ é o coeficiente de reflexão para campo elétrico ($0 \leq |\Gamma| \leq 1$) e o sinal trocado indica propagação da onda refletida na direção -x. Somando os campos elétricos incidente e refletido é obtido o campo elétrico total na região 1 como:

$$\boldsymbol{E} = \boldsymbol{k}.(1 + \Gamma e^{j2.k_{1x}.x}).E_{Z0}.e^{-jk_{1x}.x - jk_{1y}.y} \tag{5.6}$$

O campo magnético total $\boldsymbol{H}$ na região 1 a partir da equação rotacional de Maxwell (2.37), considerando $E_X = E_Y = 0$ e $\partial/\partial z = 0$, é dado por:

$$\boldsymbol{H} = \frac{-1}{j\omega\mu_1}\nabla \times \boldsymbol{E} = \boldsymbol{i}.\frac{k_{1y}}{\omega\mu_1}\left(1 + \Gamma e^{j2.k_{1x}.x}\right).E_{Z0}.e^{-jk_{1x}.x - jk_{1y}.y} + \boldsymbol{j}.\frac{k_{1x}}{\omega\mu_1}\left(-1 + \Gamma e^{j2.k_{1x}.x}\right).E_{Z0}.e^{-jk_{1x}.x - jk_{1y}.y} \tag{5.7}$$

O campo magnético total $\boldsymbol{H}$ na região 1 possui componentes H_X e H_Y.

Na região 2 ter-se-á os campos transmitidos que devem ter continuidade com os campos da região 1 na interface (x = 0) da Figura 5.1, de acordo com as condições de contorno analisadas na subseção 2.2.6. Contudo, antes de aplicar estas condições de contorno será desenvolvida a formulação de Bérenger na região 2. São consideradas as equações (2.105) para o modo TM_Z como definidas na subseção 2.3.4, mas no domínio da frequência (em forma fasorial), tal que na região 2 (PML) tem-se para os campos magnéticos H_X e H_Y:

$$j\omega\mu_2 H_Y + \sigma_{mx}H_Y = \frac{\partial E_Z}{\partial x} \tag{5.8a}$$

$$j\omega\mu_2 H_X + \sigma_{my}H_X = -\frac{\partial E_Z}{\partial y} \tag{5.8b}$$

onde σ_{mx} e σ_{my} são condutividades magnéticas associadas com as direções *x* e *y*, respectivamente, tal que $\sigma_{mx} = \sigma_{mx}(x)$ e $\sigma_{my} = \sigma_{my}(y)$, e não com as componentes H_X ou H_Y do campo magnético. Assim na direção *x*, os campos não-zero são H_Y e E_Z. E na direção *y*, os campos não-zero são H_X e E_Z. Para o campo elétrico E_Z, também a partir da equação (2.105c), mas no domínio da frequência (em forma fasorial) tem-se:

$$j\omega\varepsilon_2 E_z + \sigma E_z = \frac{\partial H_Y}{\partial x} - \frac{\partial H_X}{\partial y} \tag{5.9}$$

Nesta equação existe uma condutividade elétrica (σ) única. No entanto, J. P. Bérenger conseguiu obter comportamento anisotrópico de condutividade elétrica dividindo o campo E_Z tal que:

$$E_Z = E_{ZX} + E_{ZY} \tag{5.10}$$

E assim aplicando a equação (5.10) de forma a modificar a equação (5.9) para:

$$j\omega\varepsilon_2 E_{ZX} + \sigma_x E_{ZX} = \frac{\partial H_Y}{\partial x} \tag{5.11a}$$

$$j\omega\varepsilon_2 E_{ZY} + \sigma_y E_{ZY} = -\frac{\partial H_X}{\partial y} \tag{5.11b}$$

Observe que se for escolhido $\sigma_x = \sigma_y = \sigma$ e somando as equações (5.11a) e (5.11b) é obtida a equação original (5.9). Pode-se observar das equações (5.8) e (5.11) que se $\sigma_y = \sigma_{my} = 0$ e $\sigma_x \neq 0$, $\sigma_{mx} \neq 0$ somente as ondas com componentes H_Y e E_{ZX} são atenuadas na direção *x*; mas se $\sigma_y \neq 0$, $\sigma_{my} \neq 0$ e $\sigma_x = \sigma_{mx} = 0$ somente as ondas com componentes H_X e E_{ZY} são atenuadas na direção *y*. O uso seletivo de valores de condutividades elétricas e magnéticas nas direções *x* e *y* permite criar uma eficiente PML, como será demonstrado na sequência. A partir da teoria desenvolvida na subseção 2.2.5 e aplicada nas equações (5.8) e (5.11) tem-se:

$$S_w = 1 + \frac{\sigma_w}{j\omega\varepsilon_2} \tag{5.12a}$$

$$S_{mw} = 1 + \frac{\sigma_{mw}}{j\omega\mu_2} \tag{5.12b}$$

onde ε_2 e μ_2 são valores específicos para a região 2, o índice $w \in \{x, y\}$ e S_w, S_{mw} são condutividades normalizadas tal que as equações (5.8) e (5.11) são reescritas como:

$$j\omega\varepsilon_2 S_x E_{ZX} = \frac{\partial H_Y}{\partial x} \tag{5.13}$$

$$j\omega\varepsilon_2 S_y E_{ZY} = -\frac{\partial H_X}{\partial y} \tag{5.14}$$

$$j\omega\mu_2 S_{mx} H_Y = \frac{\partial}{\partial x}(E_{ZX} + E_{ZY}) \tag{5.15}$$

$$j\omega\mu_2 S_{my} H_X = -\frac{\partial}{\partial y}(E_{ZX} + E_{ZY}) \tag{5.16}$$

Tomando a derivada parcial em relação a x do campo H_Y em (5.15), rearranjando e substituindo no lado direito de (5.13), é obtido:

$$-\omega^2\mu_2\varepsilon_2 S_x S_{mx} E_{ZX} = \frac{\partial^2}{\partial x^2}(E_{ZX} + E_{ZY}) \tag{5.17}$$

Tomando a derivada parcial em relação a direção y do campo H_X em (5.16), rearranjando e substituindo no lado direito de (5.14), é obtido:

$$-\omega^2\mu_2\varepsilon_2 S_y S_{my} E_{ZY} = \frac{\partial^2}{\partial x^2}(E_{ZX} + E_{ZY}) \tag{5.18}$$

Dividindo as equações (5.17) e (5.18) pelos seus respectivos termos S_w, S_{mw} e somando as duas equações é obtido:

$$-\omega^2\mu_2\varepsilon_2(E_{ZX} + E_{ZY}) = \left(\frac{1}{S_x S_{mx}}\frac{\partial^2}{\partial x^2} + \frac{1}{S_y S_{my}}\frac{\partial^2}{\partial y^2}\right)(E_{ZX} + E_{ZY}) \tag{5.19}$$

Aplicando a equação (5.10) dos campos divididos para E_Z em (5.19) é encontrado:

$$\left(\frac{1}{S_x S_{mx}}\frac{\partial^2}{\partial x^2} + \frac{1}{S_y S_{my}}\frac{\partial^2}{\partial y^2} + \omega^2\mu_2\varepsilon_2\right)E_Z = 0 \tag{5.20}$$

Para satisfazer a equação (5.20) o campo transmitido E^t_Z na região 2 (PML) deve ser dado por:

$$E_Z^t = T.E_{Z0}.\exp\left(-j\sqrt{S_xS_{mx}}\,.k_{2x}.x\; - j\sqrt{S_yS_{my}}\,.k_{2y}.y\right) \tag{5.21}$$

onde T é o coeficiente de transmissão para campo elétrico, e as componentes x e y do vetor número de onda ($\boldsymbol{k_2}$) na região 2 obedecem à seguinte relação:

$$k_{2x}^2 + \; k_{2y}^2 = \; k_2^2 = \; \omega^2\mu_2\varepsilon_2 \tag{5.22}$$

Usando a equação (5.21) na equação (5.15) a componente y do campo magnético na PML é dada por:

$$H_Y^t = \; \frac{1}{j\omega\mu_2 S_{mx}}\frac{\partial E_Z^t}{\partial x} = -\frac{\sqrt{S_xS_{mx}}\;\; k_{2x}}{\omega\mu_2 S_{mx}}E_Z^t = \; -\frac{.k_{2x}}{\omega\mu_2}\sqrt{\frac{S_x}{S_{mx}}}E_Z^t \tag{5.23}$$

Aplicando a condição de contorno para campos tangenciais elétricos como definido pela equação (2.60), isto é, igualando a equação (5.6) com a equação (5.21) para x = 0, tem-se:

$$(1 + \; \Gamma)E_{Z0}.\exp\left(\; -jk_{1y}.y\right) = T.E_{Z0}.\exp\left(\; -j\sqrt{S_yS_{my}}\,.k_{2y}.y\right) \tag{5.24}$$

Para que a equação (5.24) seja válida para todo y é necessário fazer $S_y = S_{my} = 1$ (ou $\sigma_y = \sigma_{my} = 0$) e $k_{2y} = k_{1y}$, assim obtendo:

$$1 + \; \Gamma = T \tag{5.25}$$

Aplicando a condição de contorno para campos magnéticos tangenciais, como definido na equação (2.62), isto é, igualando a componente H_Y da equação (5.7) com a equação (5.23), onde o campo elétrico transmitido (E^t_Z) é definido pela equação (5.21), para x = 0 tem-se:

$$\frac{k_{1x}}{\omega\mu_1}(-1 + \; \Gamma)E_{Z0}.\exp\left(\; -jk_{1y}.y\right) = -\frac{k_{2x}}{\omega\mu_2}\sqrt{\frac{S_x}{S_{mx}}}.T.E_{Z0}.\exp\left(\; -j\sqrt{S_yS_{my}}\,.k_{2y}.y\right) \tag{5.26}$$

Aplicando na equação (5.26) as mesmas condições da equação (5.24), ou seja, $S_{my} = S_y = 1$ e $k_{2y} = k_{1y}$, é obtido:

$$1 - \Gamma = \frac{\mu_1 k_{2x}}{\mu_2 k_{1x}} \sqrt{\frac{S_x}{S_{mx}}} . T \tag{5.27}$$

Usando a equação (5.27) com a equação (5.25) é possível encontrar os valores dos coeficientes de transmissão (T) e reflexão (Γ) como:

$$T = \frac{2.\frac{k_{1x}}{\mu_1}}{\frac{k_{1x}}{\mu_1} + \frac{k_{2x}}{\mu_2}\sqrt{\frac{S_x}{S_{mx}}}} \tag{5.28}$$

$$\Gamma = \frac{\frac{k_{1x}}{\mu_1} - \frac{k_{2x}}{\mu_2}\sqrt{\frac{S_x}{S_{mx}}}}{\frac{k_{1x}}{\mu_1} + \frac{k_{2x}}{\mu_2}\sqrt{\frac{S_x}{S_{mx}}}} \tag{5.29}$$

Para que o coeficiente de reflexão (Γ) seja zero é suficiente escolher $\mu_1 = \mu_2$, $k_{1x} = k_{2x}$ e $S_{mx} = S_x$. Esta última condição é conseguida fazendo $\sigma_{mx} / \mu_2 = \sigma_x / \varepsilon_2$. Dessa forma o conjunto completo de condições de casamento perfeito na interface com $x = 0$ é:

$$\mu_1 = \mu_2; \quad \varepsilon_1 = \varepsilon_2; \quad \sigma_y = \sigma_{my} = 0; \quad \frac{\sigma_{mx}}{\mu_2} = \frac{\sigma_x}{\varepsilon_2}; \quad k_{1x} = k_{2x}; \quad k_{1y} = k_{2y} \tag{5.30}$$

Sob essas condições a propagação dos campos elétrico (E_Z) e magnéticos (H_X ou H_Y) na região 2 (PML), segundo o termo exponencial da equação (5.21), é determinada por:

$$\exp\left(-jS_x \,.\, k_{1x}.x \; - jk_{1y}.y\right) = \; \exp\left(-\frac{k_{1x}}{\omega\varepsilon_1}.\sigma_x.x\right) \exp\left(-jk_{1x}.x \; - jk_{1y}.y\right) \tag{5.31}$$

onde $k_{1x} = k_1.\cos\theta_i$. Esta equação mostra no 1º termo exponencial do lado direito que existe um decaimento exponencial na direção x, mas a fase é a mesma da região 1 (2º termo exponencial do lado direito) e com ângulo de refração ($\theta_t = \theta_i$), ou seja, sem reflexão da onda incidente, como mostrado na Figura 5.1. A atenuação da onda na

região 2 (PML) depende do valor da condutividade (σ_x), da distância *x* e do ângulo de incidência (θ_i).

5.3 FORMULAÇÃO CPML PARA O MÉTODO FDTD YEE

Na seção anterior foi observado que para a condição de casamento de impedância ($\Gamma = 0$) tem-se $S_{mw} = S_w$ onde $w \in \{x, y\}$. Logo é possível definir um único termo genérico S_w tal que:

$$S_w = 1 + \frac{\sigma_w}{j\omega\varepsilon_0} = 1 + \frac{\sigma_{mw}}{j\omega\mu_0} \tag{5.32}$$

onde os termos ε_0 e μ_0 são definidos para o vácuo (ver subseção 2.2.5). É possível generalizar a equação (5.32) de forma a permitir que a região PML seja eficiente para ondas evanecentes de frequência baixa geradas por espalhamento de onda em objetos próximos à região PML ou por campos gerados por antenas também próximas à região PML (TAFLOVE; HAGNESS, 2005). Desta forma a equação (5.32) é modificada para:

$$S_w = \kappa_w + \frac{\sigma_w}{a_w + j\omega\varepsilon_0} \tag{5.33}$$

O termo a_w na equação (5.33) melhora o desempenho da PML para frequências baixas mantendo o coeficiente de reflexão baixo na interface da PML com a grade FDTD (TAFLOVE; HAGNESS, 2005). O termo κ_w permite que a permissividade relativa possa mudar na PML. Este termo será útil, por exemplo, quando trabalhando com meios com perdas adjacentes à PML, o qual aumentará a atenuação da onda dentro da PML (TAFLOVE; HAGNESS, 2005). O uso de permissividade elétrica (ε_0) do vácuo ao invés da permissividade elétrica (ε) do meio adjacente à PML na equação (5.33) permite que a condutividade (σ_w) seja constante na interface e assim bem adaptada à presença de meios não-homogêneos (TAFLOVE; HAGNESS, 2005). É bom lembrar que os termos σ_w, a_w e κ_w não precisam ter significado físico real, mas são ajustados para maximizar o desempenho da PML. Para o caso 3D é admitido $w \in \{x, y, z\}$. Usando o único termo S_w às equações (5.13-16) do modo TM_Z da seção anterior, estas tornam-se:

$$j\omega\varepsilon E_{ZX} = \frac{1}{S_x}\frac{\partial H_Y}{\partial x} \tag{5.34}$$

$$j\omega\varepsilon E_{ZY} = -\frac{1}{S_y}\frac{\partial H_X}{\partial y} \tag{5.35}$$

$$j\omega\mu H_Y = \frac{1}{S_x}\frac{\partial E_Z}{\partial x} \tag{5.36}$$

$$j\omega\mu H_X = -\frac{1}{S_y}\frac{\partial E_Z}{\partial y} \tag{5.37}$$

Somando as equações (5.34) e (5.35) é obtido:

$$j\omega\varepsilon E_Z = \frac{1}{S_x}\frac{\partial H_Y}{\partial x} - \frac{1}{S_y}\frac{\partial H_X}{\partial y} \tag{5.38}$$

Nas equações acima é observado que o termo S_w está sempre ligado a uma derivada parcial em relação à direção *w*. Assim, é possível definir um novo operador del ($\boldsymbol{\nabla_s}$) tal que:

$$\boldsymbol{\nabla_s} = \boldsymbol{i}\frac{1}{S_x}\frac{\partial}{\partial x} + \boldsymbol{j}\frac{1}{S_y}\frac{\partial}{\partial y} + \boldsymbol{k}\frac{1}{S_z}\frac{\partial}{\partial z} \tag{5.39}$$

Usando este operador $\boldsymbol{\nabla_s}$ nas equações (2.37) e (2.39) de Maxwell na forma rotacional, respectivamente, tem-se:

$$-j\omega\mu\boldsymbol{H} = \boldsymbol{\nabla_s} \times \boldsymbol{E} \tag{5.40}$$

$$j\omega\varepsilon\boldsymbol{E} = \boldsymbol{\nabla_s} \times \boldsymbol{H} \tag{5.41}$$

As equações (5.40) e (5.41) são válidas para o caso geral 3D. Esta formulação é conhecida como formulação PML com coordenadas estiradas (CHEW; WEEDON, 1994). Usando as equações (5.40) e (5.41) é possível desenvolver uma formulação PML sem a divisão de campos conhecida como *Convolutional* PML (CPML) (RODEN; GEDNEY, 2000). Portanto, é possível obter equações escalares a partir do caso 3D

geral dado pelas equações (5.40) e (5.41). Como um exemplo, pode ser obtida a equação escalar para a componente E_X do vetor campo elétrico $\boldsymbol{E}$, em forma fasorial, a partir da equação (5.41), tal que:

$$j\omega\varepsilon E_X = \frac{1}{S_y}\frac{\partial H_Z}{\partial y} - \frac{1}{S_z}\frac{\partial H_Y}{\partial z} \tag{5.42}$$

Convertendo essa equação para o domínio do tempo é produzido:

$$\varepsilon\frac{\partial E_X}{\partial t} = \hat{S}_y * \frac{\partial H_Z}{\partial y} - \hat{S}_z * \frac{\partial H_Y}{\partial z} \tag{5.43}$$

onde o símbolo " $*$ " indica convolução no tempo e o termo $\hat{S}_w = \hat{S}_w(t)$ é a transformada inversa de Fourier de $(1 / S_w)$. O termo $\hat{S}_w(t)$ tem a seguinte resposta impulso:

$$\hat{S}_w(t) = \frac{\delta(t)}{\kappa_w} - u(t).\frac{\sigma_w}{\varepsilon_0.\kappa_w^2}.\exp\left(-t\left[\frac{a_w}{\varepsilon_0} + \frac{\sigma_w}{\kappa_w.\varepsilon_0}\right]\right) = \frac{\delta(t)}{\kappa_w} + \varphi_w(t) \tag{5.44}$$

onde $\delta(t)$ e $u(t)$ são as funções impulso e passo no tempo, respectivamente. Aplicando a equação (5.44) na equação (5.43) é obtido:

$$\varepsilon\frac{\partial E_X}{\partial t} = \frac{1}{\kappa_y}\frac{\partial H_Z}{\partial y} - \frac{1}{\kappa_z}\frac{\partial H_Y}{\partial z} + \varphi_y(t) * \frac{\partial H_Z}{\partial y} - \varphi_z(t) * \frac{\partial H_Y}{\partial z} \tag{5.45}$$

A resposta impulso discreta de $\varphi_w(t)$ é:

$$\begin{aligned}\varphi_w(n) &= \int_{\tau=n.\Delta t}^{(n+1).\Delta t} \varphi_w(\tau).d\tau \\ &= \frac{-\sigma_w}{\varepsilon_0\kappa_w^2}\int_{\tau=n.\Delta t}^{(n+1).\Delta t} \exp\left(-\left[\frac{a_w}{\varepsilon_0} + \frac{\sigma_w}{\kappa_w.\varepsilon_0}\right].\tau\right).d\tau = C_w(b_w)^n\end{aligned} \tag{5.46}$$

onde b_w e C_w são valores escalares para o passo de tempo Δt, definidos como:

$$b_w = \exp\left(-\left[\frac{a_w}{\varepsilon_0} + \frac{\sigma_w}{\kappa_w.\varepsilon_0}\right].\Delta t\right) \tag{5.47}$$

$$C_w = \frac{\sigma_w}{\sigma_w\kappa_w + \kappa_w^2 a_w}(b_w - 1) \tag{5.48}$$

Logo a equação (5.45) na forma de diferenças finitas, usando como base a equação de diferenças finitas convencional (2.125a) do método FDTD Yee, torna-se:

$$E_X\big|_{i+\frac{1}{2},j,k}^{n+1} = C_{EX0}E_X\big|_{i+\frac{1}{2},j,k}^{n} + C_{EY}\frac{1}{\kappa_y(j)}\left(H_Z\big|_{i+\frac{1}{2},j+\frac{1}{2},k}^{n+1/2} - H_Z\big|_{i+\frac{1}{2},j-\frac{1}{2},k}^{n+1/2}\right) - C_{EZ}\frac{1}{\kappa_z(k)}\left(H_Y\big|_{i+\frac{1}{2},j,k+\frac{1}{2}}^{n+\frac{1}{2}} - H_Y\big|_{i+\frac{1}{2},j,k-\frac{1}{2}}^{n+\frac{1}{2}}\right) + C_{EY}.\varphi_{Ex,y}\big|_{i+\frac{1}{2},j,k}^{n+1/2} - C_{EZ}.\varphi_{Ex,z}\big|_{i+\frac{1}{2},j,k}^{n+1/2} \quad (5.49)$$

Os coeficientes C_{EX0}, C_{EY} e C_{EZ} são definidos na subseções 2.3.6, 2.3.7 e 2.3.8; e os termos $\varphi_{Ex,w}$ na equação (5.49) são obtidos recursivamente no tempo como:

$$\varphi_{Ex,y}\big|_{i+\frac{1}{2},j,k}^{n+1/2} = b_y(j).\varphi_{Ex,y}\big|_{i+\frac{1}{2},j,k}^{n-\frac{1}{2}} + C_y(j).\left(H_Z\big|_{i+\frac{1}{2},j+\frac{1}{2},k}^{n+1/2} - H_Z\big|_{i+\frac{1}{2},j-\frac{1}{2},k}^{n+1/2}\right) \quad (5.50a)$$

$$\varphi_{Ex,z}\big|_{i+\frac{1}{2},j,k}^{n+1/2} = b_z(k).\varphi_{Ex,z}\big|_{i+\frac{1}{2},j,k}^{n-1/2} + C_z(k).\left(H_Y\big|_{i+\frac{1}{2},j,k+\frac{1}{2}}^{n+\frac{1}{2}} - H_Y\big|_{i+\frac{1}{2},j,k-\frac{1}{2}}^{n+\frac{1}{2}}\right) \quad (5.50b)$$

onde os termos b_y e b_z são obtidos da equação (5.47) e os termos C_y e C_z são obtidos da equação (5.48). Embora a formulação CPML pareça a princípio complexa, sua aplicação é relativamente simples. A convolução no tempo é recursiva e necessita-se conhecer unicamente os valores das variáveis $\varphi_{Eu,w}$ ou $\varphi_{Hu,w}$ (com $u \neq w$ e $u, w \in \{x, y, z\}$) no tempo anterior, portanto, sendo facilmente adaptada ao método FDTD convencional que também necessita dos valores dos campos unicamente no tempo anterior. As equações de diferença finita necessitam apenas pequenas modificações e algumas variáveis extras, que precisam ser unicamente armazenadas para as posições dentro das camadas PML. A formulação CPML funciona bem para materiais não-homogêneos, com perdas, anisotrópicos, não-lineares e dispersivos (TAFLOVE; HAGNESS, 2005).

5.4 GRADUAÇÃO DA CONDUTIVIDADE NA PML

A PML de campo dividido de Bérenger ou a CPML analisadas neste texto requerem dois princípios básicos para funcionar corretamente: primeiro, deve haver reflexão zero na interface e, segundo, deve haver absorção da onda incidente que se propaga dentro da PML. Para uma condutividade σ_x variando com a direção *x* o fator de reflexão $R(\theta_i)$ estimado (INAN; MARSHALL, 2011) em função do ângulo de incidência (θ_i) e espessura (*d*) da PML é:

$$R(\theta_i) = exp\left(-2.\eta.\cos\theta_i \int_0^d \sigma_x(x).dx\right) \tag{5.51}$$

onde η é a impedância na PML. Assim, a segunda dificuldade que surge é que, devido à discretização finita do espaço na grade, a reflexão não será zero, pois embora a impedância (η) na interface da PML esteja casada com a grade, os campos elétrico e magnético estão intervalados por meio passo espacial na grade e encontram o contorno da PML diferentemente. Para resolver este problema Bérenger propôs graduar suavemente a condutividade na PML de um valor zero (na interface) até um valor máximo no contorno final da PML (BÉRENGER, 1994). Duas formas de graduação de condutividade têm sido usadas com sucesso: variação polinomial e variação geométrica (INAN; MARSHALL, 2011). Neste texto analisar-se-á apenas a variação polinomial. A graduação polinomial do perfil de condutividade é dada por:

$$\sigma_x(x) = \sigma_{x,max}\left(\frac{x}{d}\right)^m \tag{5.52a}$$

$$\kappa_x(x) = 1 + \left(\kappa_{x,max} - 1\right)\left(\frac{x}{d}\right)^m \tag{5.52b}$$

$$a_x(x) = a_{x,max}\left(\frac{d-x}{d}\right)^{ma} \tag{5.52c}$$

onde $0 \leq x \leq d$, *m* ou *ma* determinam as ordens dos polinômios (não precisam ser números inteiros). Os termos κ_x e a_x (como resumidamente analisados na seção anterior) são usados em algumas PMLs para otimizar os parâmetros S_w, com $\kappa_{x,max} \geq 1$ e $a_{x,max} \geq 0$; para maiores detalhes consultar (TAFLOVE; HAGNESS, 2005).

O termo *d* é a espessura da PML assim $\sigma_x(d) = \sigma_{x,max}$. Usando esses parâmetros e aplicando a equação (5.52a) na equação (5.51), o fator de reflexão é avaliado como:

$$R(\theta_i) = exp\left(-\frac{2}{m+1}.\eta.\sigma_{x,max}.d.\cos\theta_i\right) \quad (5.53)$$

Resolvendo para $\sigma_{x,max}$ com ângulo de incidência $\theta_i = 0°$ é encontrado:

$$\sigma_{x,max} = -\frac{(m+1)}{2.\eta.d}\ln[R(0°)] \quad (5.54)$$

Onde *R*(0°) é a reflexão desejada para incidência normal, sendo um valor típico da ordem de 10^{-6}. Uma espessura típica para o método FDTD Yee é $d = 10.\Delta x$, e $3 \leq m \leq 4$. A impedância (η) é a mesma na interface da PML, ou seja, igual ao material da grade próxima à interface; se este material for o espaço livre ter-se-á $\eta = 120.\pi \approx 377$ ohms. É prudente verificar que a escolha de $R(0°) = 10^{-6}$ na equação (5.54) não garante que esta especificação será alcançada; na prática, normalmente, os valores reais de *R*(0°) serão maiores. A dedução da equação (5.54) é baseada num tratamento analítico da PML em meio contínuo com perdas; no entanto, o processo de discretização limita o desempenho da PML na grade FDTD. Para ilustrar: um exemplo de simulação em (INAN; MARSHALL, 2011) mediu um valor médio no tempo para o coeficiente de reflexão como $R(0°) \approx 3.\ 10^{-3}$ para $d = 10.\Delta x$ ao invés do valor teórico $R(0°) = 10^{-6}$, e aumentando a espessura da PML para $d = 20.\Delta x$ foi encontrado um valor médio no tempo $R(0°) \approx 10^{-4}$, mas também acima do valor teórico $R(0°) = 10^{-6}$.

5.5 FORMULAÇÃO CPML PARA A GRADE DE PRISMAS HEXAGONAIS

5.5.1 Formulação de Bérenger para a grade 2D de hexágonos

Antes de desenvolver a formulação CPML para a grade de prismas hexagonais, foi desenvolvida também a formulação de Bérenger com campos divididos no modo TM_Z para grade 2D de hexágonos, pois não se encontrou artigos sobre este assunto. Aqui, é necessário utilizar um sistema de coordenadas adequado, como mostrado na Figura 5.2.

Na Figura 5.2 os vetores unitários $\boldsymbol{a_{n1}}$, $\boldsymbol{a_{n2}}$ e $\boldsymbol{a_{n3}}$ são normais às direções dos campos magnéticos H_1, H_2 e H_3, respectivamente. É possível relacionar estes vetores unitários com os vetores unitários do sistema de coordenadas cartesiano, tal que:

$$\boldsymbol{a_{n1}} = \boldsymbol{j} \tag{5.55}$$

$$\boldsymbol{a_{n2}} = -\boldsymbol{i}.\cos 30° + \boldsymbol{j}.\operatorname{sen} 30° = -\boldsymbol{i}\frac{\sqrt{3}}{2} + \boldsymbol{j}\frac{1}{2} \tag{5.56}$$

$$\boldsymbol{a_{n3}} = -\boldsymbol{i}.\cos 30° - \boldsymbol{j}.\operatorname{sen} 30° = -\boldsymbol{i}\frac{\sqrt{3}}{2} - \boldsymbol{j}\frac{1}{2} \tag{5.57}$$

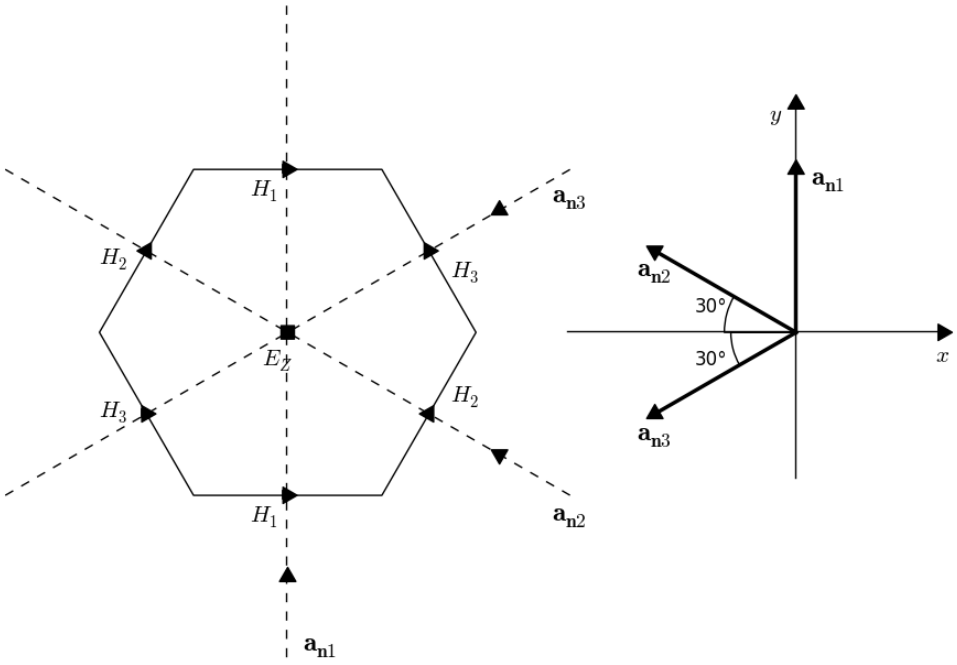


Figura 5.2 - Sistema de coordenadas para a grade 2D de hexágonos.

Também é possível definir um vetor posição $\boldsymbol{r}$ e um vetor número de onda $\boldsymbol{k_1}$ (na região 1) neste novo sistema de coordenadas como:

$$\boldsymbol{r} = \boldsymbol{a_{n1}}.n_1 + \boldsymbol{a_{n2}}.n_2 + \boldsymbol{a_{n3}}.n_3 = (-n_2 - n_3)\frac{\sqrt{3}}{2}.\boldsymbol{i} + \left(n_1 + \frac{n_2}{2} - \frac{n_3}{2}\right).\boldsymbol{j} \tag{5.58}$$

$$\boldsymbol{k_1} = \boldsymbol{a_{n1}}.k_{11} + \boldsymbol{a_{n2}}.k_{12} + \boldsymbol{a_{n3}}.k_{13} \tag{5.59}$$

Na equação (5.58) o vetor posição $\boldsymbol{r}$ é definido em função das posições n_1, n_2 e n_3 que são normais aos campos H_1, H_2 e H_3, respectivamente. Neste novo sistema de coordenadas os vetores unitários não são linearmente independentes (ARFKEN; WEBER, 2007). Assim, para que se possa definir vetores corretos de incidência ($\boldsymbol{r_i}$) e reflexão ($\boldsymbol{r_r}$) nas interfaces dos campos H_1, H_2 e H_3, usando a equação (5.58), é necessário que os ângulos de incidência (θ_i) e reflexão (θ_r) sejam iguais, como mostrado na Figura 5.3.

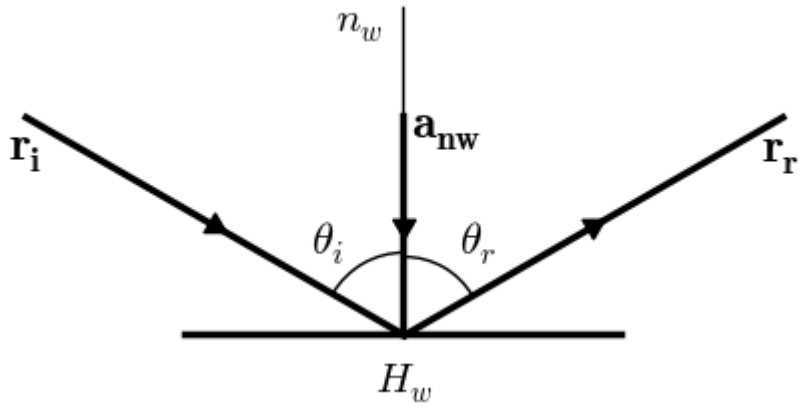


Figura 5.3 - Vetores de incidência (r_i) e reflexão (r_r) na interface dos campos H_1, H_2 ou H_3.

Na Figura 5.3 o índice $w \in \{1, 2, 3\}$. Assim, na interface do campo magnético H_1, para se ter $\theta_i = \theta_r$ é necessário que seja satisfeita a seguinte condição:

$$\boldsymbol{r_i} \,.\, \boldsymbol{a_{n1}} = -\boldsymbol{r_r} \,.\, \boldsymbol{a_{n1}} \tag{5.60}$$

Aplicando as equações (5.55) e (5.58) na equação (5.60) e trocando o sinal de n_1 para o vetor de reflexão ($\boldsymbol{r_r}$) é obtido:

$$\left[(-n_2 - n_3)\frac{\sqrt{3}}{2}\boldsymbol{i} + \left(n_1 + \frac{n_2}{2} - \frac{n_3}{2}\right)\boldsymbol{j}\right].\boldsymbol{j} = -\left[(-n_2 - n_3)\frac{\sqrt{3}}{2}\boldsymbol{i} + \left(-n_1 + \frac{n_2}{2} - \frac{n_3}{2}\right)\boldsymbol{j}\right].\boldsymbol{j}$$

e, então:

$$n_3 = n_2 \tag{5.61}$$

O resultado obtido ($n_3 = n_2$) é aplicado nos vetores $\boldsymbol{r_i}$ e $\boldsymbol{r_r}$ obtendo-se a seguinte igualdade $|\boldsymbol{r_i}| = |\boldsymbol{r_r}|$ o que indica que a restrição obtida está correta. Um procedimento similar, a partir da Figura 5.3, é aplicado nas interfaces dos campos H_2 e H_3, obtendo-se as seguintes restrições:

$$\boldsymbol{r_i} \,.\, \boldsymbol{a_{n2}} = -\boldsymbol{r_r} \,.\, \boldsymbol{a_{n2}} \quad \Rightarrow \quad n_3 = -n_1 \tag{5.62}$$

$$\boldsymbol{r_i} \,.\, \boldsymbol{a_{n3}} = -\boldsymbol{r_r} \,.\, \boldsymbol{a_{n3}} \quad \Rightarrow \quad n_2 = n_1 \tag{5.63}$$

Assim, aplicando este novo sistema de coordenadas na região 1 que faz interface com a região 2 (PML) ter-se-á o campo elétrico incidente (E^i_Z) como:

.

$$\boldsymbol{E^i} = \boldsymbol{k}.\, E_{Z0}.\, e^{-j\boldsymbol{k_1}.\boldsymbol{r}} \tag{5.64a}$$

$$\boldsymbol{E^i} = \boldsymbol{k}.\, E_{Z0}.\, e^{-jk_{11}.n_1 - jk_{12}.n_2 - jk_{13}.n_3} \tag{5.64b}$$

O campo H_1 da equação (3.13) da grade 2D de hexágonos, em forma fasorial (no domínio da frequência), substituindo $\partial/\partial t \rightarrow j\omega$ torna-se:

$$j\omega\mu H_1 = \frac{\partial E_Z}{\partial n_1} \tag{5.65}$$

onde a derivada parcial do campo E_Z é tomada na direção n_1, normal ao campo H_1. De forma similar os campos H_2 e H_3 a partir das equações (3.16) e (3.17), respectivamente, tornam-se:

$$j\omega\mu H_2 = \frac{\partial E_Z}{\partial n_2} \tag{5.66}$$

$$j\omega\mu H_3 = \frac{\partial E_Z}{\partial n_3} \tag{5.67}$$

onde n_2 e n_3 indicam direções normais aos campos H_2 e H_3, respectivamente. A equação para campo E_Z a partir da equação (3.10) aplicando procedimento similar,

mas multiplicando o lado direito pela dimensão do lado do hexágono maior (Δ*d*) para compensar as derivadas explicitas nas direções n_1, n_2 e n_3, torna-se:

$$j\omega\varepsilon E_Z = \frac{\Delta b.\Delta d}{A_H}\left(\frac{\partial H_1}{\partial n_1} + \frac{\partial H_2}{\partial n_2} + \frac{\partial H_3}{\partial n_3}\right) = \frac{2}{3}\left(\frac{\partial H_1}{\partial n_1} + \frac{\partial H_2}{\partial n_2} + \frac{\partial H_3}{\partial n_3}\right) \tag{5.68}$$

onde Δ*b* é o tamanho do lado do hexágono menor (ver Figura 3.1) e A_H corresponde à área do hexágono menor definida por $A_H = (\sqrt{3}/2)\,(\Delta d)^2$. Sabendo que $\Delta b = \Delta d/\sqrt{3}$, logo comprova-se que o termo constante na equação (5.68) é:

$$\frac{\Delta b.\Delta d}{A_H} = \frac{(\Delta d/\sqrt{3}).\Delta d}{(\sqrt{3}/2)(\Delta d)^2} = \frac{2}{3} \tag{5.69}$$

No método FDTD Yee, com células em formato hexaédrico, as interfaces das PMLs são perpendiculares às direções *x*, *y* e *z*. Todavia, no método FDTD aplicado à grade de prismas hexagonais as interfaces das PMLs são perpendiculares às direções n_1, n_2, n_3 e *z*. Particularmente no plano x-y o contorno da PML pode ser definido conforme mostrado na Figura 5.4, ou seja, na direção *y* o contorno PML segue os campos H_2 e H_3, tendo um formato de dente de serra. Na direção *x* o contorno segue os campos H_1 ou E_1 na forma de uma linha reta. Na direção *z* a forma do contorno é também de uma linha reta.

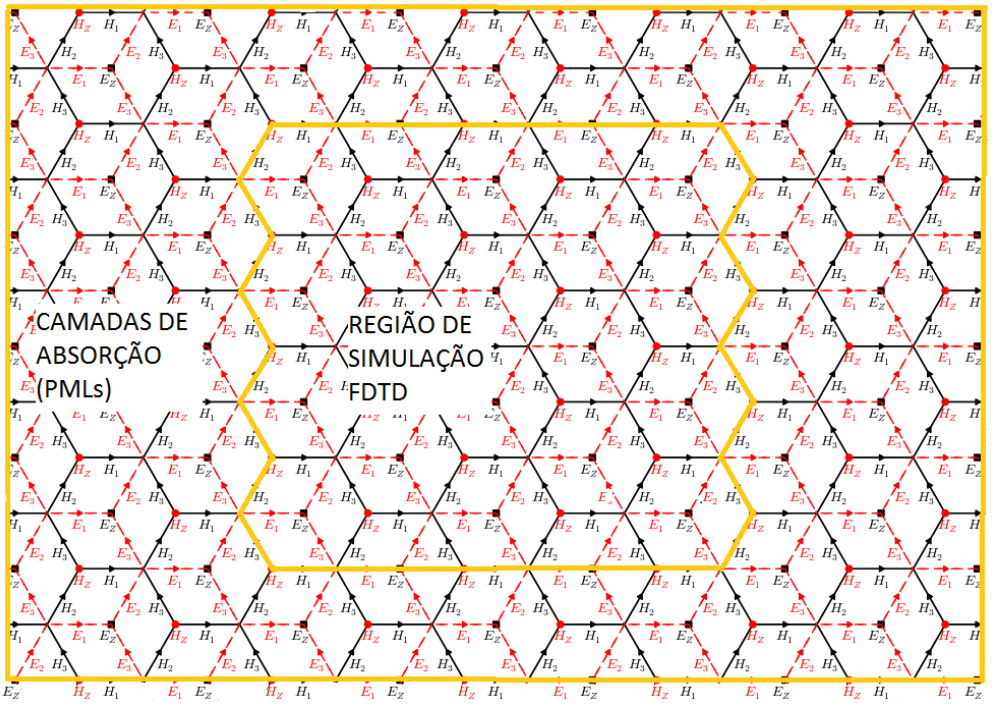


Figura 5.4 - Formato do contorno entre a região de simulação e as PMLs.

Na região 1, para o campo magnético incidente H^i_1 na interface *y* constante (na direção n_1), aplicando a equação (5.65) na equação (5.64b), tem-se:

$$H_1^i = -\frac{k_{11}}{\omega,\mu_1} E_{Z0}.\, e^{-jk_{11}.n_1 - jk_{12}.n_2 - jk_{13}.n_3} \tag{5.70}$$

O campo refletido de H_1 na interface *y* constante será, trocando o sinal de n_1 e multiplicando pelo fator de reflexão de campo elétrico (Γ) com sinal trocado, escrito como:

$$H_1^r = \frac{k_{11}}{\omega,\mu_1} \Gamma.\, E_{Z0}.\, e^{jk_{11}.n_1 - jk_{12}.n_2 - jk_{13}.n_3} \tag{5.71}$$

De forma similar os campos magnéticos incidentes H^i_2 e H^i_3 na interface dente de serra (nas direções n_2 e n_3, respectivamente, da Figura 5.4), aplicando as equações (5.66) e (5.67), respectivamente, na equação (5.64b), são dados por:

$$H_2^i = -\frac{k_{12}}{\omega,\mu_1} E_{Z0}.\, e^{-jk_{11}.n_1 - jk_{12}.n_2 - jk_{13}.n_3} \tag{5.72}$$

$$H_3^i = -\frac{k_{13}}{\omega,\mu_1} E_{Z0}.\, e^{-jk_{11}.n_1 - jk_{12}.n_2 - jk_{13}.n_3} \tag{5.73}$$

Os campos refletidos de H_2 e H_3 na interface dente de serra, trocando os sinais de n_2 e n_3, respectivamente e multiplicando pelo fator de reflexão com sinal trocado, são dados por:

$$H_2^r = \frac{k_{12}}{\omega,\mu_1} \Gamma.\, E_{Z0}.\, e^{-jk_{11}.n_1 + jk_{12}.n_2 - jk_{13}.n_3} \tag{5.74}$$

$$H_3^r = \frac{k_{13}}{\omega,\mu_1} \Gamma.\, E_{Z0}.\, e^{-jk_{11}.n_1 - jk_{12}.n_2 + jk_{13}.n_3} \tag{5.75}$$

Agora, aplicando a formulação de Bérenger para campos divididos nas equações (5.65), (5.66), (5.67) e (5.68) na região 2 (PML) é obtido:

$$j\omega\mu_2 S_{m1} H_1 = \frac{\partial(E_{Z1} + E_{Z2} + E_{Z3})}{\partial n_1} \tag{5.76a}$$

$$j\omega\mu_2 S_{m2} H_2 = \frac{\partial(E_{Z1} + E_{Z2} + E_{Z3})}{\partial n_2} \tag{5.76b}$$

$$j\omega\mu_2 S_{m3} H_3 = \frac{\partial(E_{Z1} + E_{Z2} + E_{Z3})}{\partial n_3} \tag{5.76c}$$

$$j\omega\varepsilon_2 S_1 E_{Z1} = \frac{2}{3}\frac{\partial H_1}{\partial n_1} \tag{5.76d}$$

$$j\omega\varepsilon_2 S_2 E_{Z2} = \frac{2}{3}\frac{\partial H_2}{\partial n_2} \tag{5.76e}$$

$$j\omega\varepsilon_2 S_3 E_{Z3} = \frac{2}{3}\frac{\partial H_3}{\partial n_3} \tag{5.76f}$$

Nas equações (5.76a-f) o campo E_Z é dividido em três componentes, tal que:

$$E_Z = E_{Z1} + E_{Z2} + E_{Z3} \tag{5.77}$$

Tomando a derivada do campo H_1 em relação à direção n_1 na equação (5.76a) e substituindo no lado direito da equação (5.76d), é obtido:

$$-\omega^2\mu_2\varepsilon_2 S_1 S_{m1} E_{Z1} = \frac{2}{3}\frac{\partial^2 E_Z}{\partial n_1^2} \tag{5.78}$$

Tomando a derivada do campo H_2 em relação à direção n_2 na equação (5.76b) e substituindo no lado direito da equação (5.76e), é obtido:

$$-\omega^2\mu_2\varepsilon_2 S_2 S_{m2} E_{Z2} = \frac{2}{3}\frac{\partial^2 E_Z}{\partial n_2^2} \tag{5.79}$$

Tomando a derivada do campo H_3 em relação à direção n_3 na equação (5.76c) e substituindo no lado direito da equação (5.76f), é obtido:

$$-\omega^2\mu_2\varepsilon_2 S_3 S_{m3} E_{Z3} = \frac{2}{3}\frac{\partial^2 E_Z}{\partial n_3^2} \tag{5.80}$$

Somando as equações (5.78), (5.79) e (5.80), usando a equação (5.77) e rearranjando os termos é encontrado:

$$\left(\frac{1}{S_1 S_{m1}}\frac{\partial^2}{\partial n_1^2} + \frac{1}{S_2 S_{m2}}\frac{\partial^2}{\partial n_2^2} + \frac{1}{S_3 S_{m3}}\frac{\partial^2}{\partial n_3^2} + \frac{3}{2}\omega^2\mu_2\varepsilon_2\right).E_Z = 0 \tag{5.81}$$

Para satisfazer esta equação o campo transmitido na região 2 (PML) é definido como:

$$E_Z^t = T.E_{Z0}.\exp\left(-j\sqrt{S_1 S_{m1}}\,.k_{21}.n_1 - j\sqrt{S_2 S_{m2}}\,.k_{22}.n_2 - j\sqrt{S_3 S_{m3}}\,.k_{23}.n_3\right) \tag{5.82}$$

onde a seguinte relação deve ser mantida para as componentes de vetor número de onda ***k₂*** (na região 2):

$$k_{21}^2 + k_{22}^2 + k_{23}^2 = k_2^2 = \frac{3}{2}\omega^2\mu_2\varepsilon_2 \tag{5.83}$$

Na interface com a região 2 (PML), paralela ao campo H_1, as equações (5.70) e (5.71) são somadas e reescritas como o campo H_1 total na região 1 dado por:

$$H_1 = H_1^i + H_1^r \tag{5.84a}$$

$$H_1 = -\frac{k_{11}}{\omega,\mu_1}E_Z^{i1} + \frac{k_{11}}{\omega,\mu_1}E_Z^{r1} \tag{5.84b}$$

$$H_1 = \frac{k_{11}}{\omega\mu_1}\left(-1 + \Gamma.\exp(2.jk_{11}.n_1)\right).E_{Z0}.\exp(-jk_{11}.n_1 - jk_{12}.n_2 - jk_{13}.n_3) \tag{5.84c}$$

As componentes incidente e refletida do campo elétrico E_z na interface paralela ao campo H_1 dadas na equação (5.84b) são definidas como:

$$E_Z^{i1} = E_{Z0}.e^{-jk_{11}.n_1 - jk_{12}.n_2 - jk_{13}.n_3} \tag{5.85a}$$

$$E_Z^{r1} = \Gamma.E_{Z0}.e^{jk_{11}.n_1 - jk_{12}.n_2 - jk_{13}.n_3} \tag{5.85b}$$

$$E_Z^t = E_Z^{i1} + E_Z^{r1} \tag{5.85c}$$

onde E^t_Z é o campo elétrico transmitido. O campo H_1 (na região 1) na interface com a região 2 (PML) dado pela equação (5.84) poderia de forma equivalente ser obtido aplicando a equação (5.65) no campo E^t_Z dado pela equação (5.85c). Como o campo elétrico transmitido (E^t_Z) na interface deve satisfazer às condições de contorno da subseção 2.2.6, o campo H_1 (na região 2) poderia também ser obtido aplicando a equação (5.76a) no campo elétrico transmitido (E^t_Z) dado pela equação (5.82) de forma que:

$$H_1 = \frac{1}{j\omega\mu_2 S_{m1}}\frac{\partial E_Z^t}{\partial n_1} = -\frac{k_{21}}{\omega\mu_2}\sqrt{\frac{S_1}{S_{m1}}}T.E_{Z0}.\exp\left(-j\sqrt{S_1S_{m1}}\,.k_{21}.n_1 - j\sqrt{S_2S_{m2}}\,.k_{22}.n_2 - j\sqrt{S_3S_{m3}}\,.k_{23}.n_3\right) \quad (5.86)$$

Como na interface entre a regiões 1 e 2 (PML) o campo H_1 é o mesmo, logo pode-se igualar as equações (5.84c) e (5.86) usando $n_1 = 0$ na interface e obtendo:

$$\frac{k_{11}}{\omega\mu_1}(-1+\Gamma).E_{Z0}.\exp(-jk_{12}.n_2 - jk_{13}.n_3) = -\frac{k_{21}}{\omega\mu_2}\sqrt{\frac{S_1}{S_{m1}}}T.E_{Z0}.\exp\left(-j\sqrt{S_2S_{m2}}\,.k_{22}.n_2 - j\sqrt{S_3S_{m3}}\,.k_{23}.n_3\right) \quad (5.87)$$

Para que a equação seja válida para todo n_2 e n_3 (a rigor $n_3 = n_2$ para que a direção de reflexão neste sistema de coordenadas seja correta segundo a equação (5.61)) é necessário usar:

$$S_{m2} = S_2 = 1; \quad S_{m3} = S_3 = 1; \quad k_{22} = k_{12}; \quad k_{23} = k_{13} \quad (5.88)$$

Para os campos elétricos incidente, refletido e transmitido a mesma relação dada pela equação (5.25) é válida e repetida aqui por conveniência:

$$1 + \Gamma = T \quad (5.89)$$

onde *T* é o coeficiente de transmissão para campo elétrico. Aplicando as equações (5.88) e (5.89) na equação (5.87), e solucionando para o coeficiente de reflexão, é obtido:

$$\Gamma = \frac{\frac{k_{11}}{\mu_1} - \frac{k_{21}}{\mu_2}\sqrt{\frac{S_1}{S_{m1}}}}{\frac{k_{11}}{\mu_1} + \frac{k_{21}}{\mu_2}\sqrt{\frac{S_1}{S_{m1}}}} \tag{5.90}$$

Assim, para se ter reflexão zero na interface do campo magnético H_1 é necessário ter:

$$S_{m1} = S_1; \quad k_{21} = k_{11}; \quad \mu_2 = \mu_1 \tag{5.91}$$

Usando procedimento semelhante ao campo H_1, na interface com a região PML, paralela ao campo H_2, as equações (5.72) e (5.74) são somadas para obter:

$$H_2 = H_2^i + H_2^r = \frac{k_{12}}{\omega\mu_1}\left(-1 + \Gamma.\exp(2.jk_{12}.n_2)\right).E_{Z0}.\exp(-jk_{11}.n_1 - jk_{12}.n_2 - jk_{13}.n_3) \tag{5.92}$$

Aplicando a equação (5.76b) no campo elétrico transmitido (E^t_Z) dado pela equação (5.82) é obtido:

$$H_2 = \frac{1}{j\omega\mu_2 S_{m2}}\frac{\partial E_Z^t}{\partial n_2} = -\frac{k_{22}}{\omega\mu_2}\sqrt{\frac{S_2}{S_{m2}}}T.E_{Z0}.\exp\left(-j\sqrt{S_1S_{m1}}.k_{21}.n_1 - j\sqrt{S_2S_{m2}}.k_{22}.n_2 - j\sqrt{S_3S_{m3}}.k_{23}.n_3\right) \tag{5.93}$$

Assim, igualando as equações (5.92) e (5.93) para $n_2 = 0$ na interface é encontrado:

$$\frac{k_{12}}{\omega\mu_1}(-1 + \Gamma).E_{Z0}.\exp(-jk_{11}.n_1 - jk_{13}.n_3) = -\frac{k_{22}}{\omega\mu_2}\sqrt{\frac{S_2}{S_{m2}}}T.E_{Z0}.\exp\left(-j\sqrt{S_1S_{m1}}.k_{21}.n_1 - j\sqrt{S_3S_{m3}}.k_{23}.n_3\right) \tag{5.94}$$

Para que a equação seja válida para todo n_1 e n_3 (a rigor $n_3 = -n_1$ para que a direção de reflexão neste sistema de coordenadas seja correta segundo a equação (5.62)) é necessário usar:

$$S_{m1} = S_1 = 1; \quad S_{m3} = S_3 = 1; \quad k_{21} = k_{11}; \quad k_{23} = k_{13} \tag{5.95}$$

Aplicando as equações (5.89) e (5.95) na equação (5.94) e solucionando para o coeficiente de reflexão, é obtido:

$$\Gamma = \frac{\frac{k_{12}}{\mu_1} - \frac{k_{22}}{\mu_2}\sqrt{\frac{S_2}{S_{m2}}}}{\frac{k_{12}}{\mu_1} + \frac{k_{22}}{\mu_2}\sqrt{\frac{S_2}{S_{m2}}}} \tag{5.96}$$

Para obter reflexão zero na interface do campo H_2 é necessário usar:

$$S_{m2} = S_2; \quad k_{22} = k_{12}; \quad \mu_2 = \mu_1 \tag{5.97}$$

E, finalmente, na interface com a região PML, paralela ao campo H_3, as equações (5.73) e (5.75) são somadas para obter:

$$H_3 = H_3^i + H_3^r = \frac{k_{13}}{\omega\mu_1}\left(-1 + \Gamma.\exp(2.jk_{13}.n_3)\right).E_{Z0}.\exp(-jk_{11}.n_1 - jk_{12}.n_2 - jk_{13}.n_3) \tag{5.98}$$

Aplicando a equação (5.76c) no campo elétrico transmitido (E^t_Z) dado pela equação (5.82), é obtido:

$$H_3 = \frac{1}{j\omega\mu_2 S_{m3}}\frac{\partial E_Z^t}{\partial n_3} = -\frac{k_{23}}{\omega\mu_2}\sqrt{\frac{S_3}{S_{m3}}}T.E_{Z0}.\exp\left(-j\sqrt{S_1S_{m1}}.k_{21}.n_1 - j\sqrt{S_2S_{m2}}.k_{22}.n_2 - j\sqrt{S_3S_{m3}}.k_{23}.n_3\right) \tag{5.99}$$

Igualando as equações (5.98) e (5.99) para $n_3 = 0$ na interface, é encontrado:

$$\frac{k_{13}}{\omega\mu_1}(-1 + \Gamma).E_{Z0}.\exp(-jk_{11}.n_1 - jk_{13}.n_2) =$$

$$-\frac{k_{23}}{\omega\mu_2}\sqrt{\frac{S_3}{S_{m3}}}T.E_{Z0}.\exp\left(-j\sqrt{S_1S_{m1}}\,.k_{21}.n_1-j\sqrt{S_2S_{m2}}\,.k_{22}.n_2\right) \tag{5.100}$$

Para que a equação (5.100) seja válida para todo n_1 e n_2 (a rigor $n_2 = n_1$ para que a direção de reflexão neste sistema de coordenadas seja correta segundo a equação (5.63)) é necessário usar:

$$S_{m1} = S_1 = 1; \quad S_{m2} = S_2 = 1; \quad k_{21} = k_{11}; \quad k_{22} = k_{12} \tag{5.101}$$

Aplicando a equação (5.89) e (5.101) na equação (5.100), e solucionando para o coeficiente de reflexão é obtido:

$$\Gamma = \frac{\frac{k_{13}}{\mu_1} - \frac{k_{23}}{\mu_2}\sqrt{\frac{S_3}{S_{m3}}}}{\frac{k_{13}}{\mu_1} + \frac{k_{23}}{\mu_2}\sqrt{\frac{S_3}{S_{m3}}}} \tag{5.102}$$

Para obter reflexão zero na interface do campo H_3 é necessário ter:

$$S_{m3} = S_3; \quad k_{23} = k_{13}; \quad \mu_2 = \mu_1 \tag{5.103}$$

5.5.2 Formulação CPML para grade de prismas hexagonais

Observa-se nas equações da subseção anterior que para a condição de casamento de impedância ($S_{mw} = S_w$) onde $w \in \{1, 2, 3, z\}$ e assim S_w é definido como na equação (5.33), lembrando que S_w varia apenas na direção n_w (ou z) tal que:

$$S_1 = S_1(n_1); \quad S_2 = S_2(n_2); \quad S_3 = S_3(n_3); \quad S_z = S_z(z) \tag{5.104}$$

O conceito de coordenadas estiradas pode, também, ser usado para as equações da grade 2D de hexágonos ou a grade 3D de prismas hexagonais. Assim, os oito campos na grade de prismas hexagonais, como definidos nas equações (3.18), (3.27), (3.32-

34) e (3.36-38), podem ter suas equações reescritas em coordenadas estiradas para o modo TM_Z como:

$$j\omega\varepsilon E_Z = \frac{2}{3}\left(\frac{1}{S_1}\frac{\partial H_1}{\partial n_1} + \frac{1}{S_2}\frac{\partial H_2}{\partial n_2} + \frac{1}{S_3}\frac{\partial H_3}{\partial n_3}\right) \quad (5.105a)$$

$$j\omega\mu H_1 = \frac{1}{S_1}\frac{\partial E_Z}{\partial n_1} + \frac{1}{\sqrt{3}}\left(\frac{1}{S_z}\frac{\partial E_2}{\partial z} + \frac{1}{S_z}\frac{\partial E_3}{\partial z}\right) \quad (5.105b)$$

$$j\omega\mu H_2 = \frac{1}{S_2}\frac{\partial E_Z}{\partial n_2} + \frac{1}{\sqrt{3}}\left(\frac{-1}{S_z}\frac{\partial E_1}{\partial z} + \frac{1}{S_z}\frac{\partial E_3}{\partial z}\right) \quad (5.105c)$$

$$j\omega\mu H_3 = \frac{1}{S_3}\frac{\partial E_Z}{\partial n_3} + \frac{1}{\sqrt{3}}\left(\frac{-1}{S_z}\frac{\partial E_1}{\partial z} - \frac{1}{S_z}\frac{\partial E_2}{\partial z}\right) \quad (5.105d)$$

e para o modo TE_Z como:

$$j\omega\mu H_Z = \frac{-2}{3}\left(\frac{1}{S_1}\frac{\partial E_1}{\partial n_1} + \frac{1}{S_2}\frac{\partial E_2}{\partial n_2} + \frac{1}{S_3}\frac{\partial E_3}{\partial n_3}\right) \quad (5.106a)$$

$$j\omega\varepsilon E_1 = \frac{-1}{S_1}\frac{\partial H_Z}{\partial n_1} + \frac{1}{\sqrt{3}}\left(\frac{-1}{S_z}\frac{\partial H_2}{\partial z} - \frac{1}{S_z}\frac{\partial H_3}{\partial z}\right) \quad (5.106b)$$

$$j\omega\varepsilon E_2 = \frac{-1}{S_2}\frac{\partial H_Z}{\partial n_2} + \frac{1}{\sqrt{3}}\left(\frac{1}{S_z}\frac{\partial H_1}{\partial z} - \frac{1}{S_z}\frac{\partial H_3}{\partial z}\right) \quad (5.106c)$$

$$j\omega\varepsilon E_3 = \frac{-1}{S_3}\frac{\partial H_Z}{\partial n_3} + \frac{1}{\sqrt{3}}\left(\frac{1}{S_z}\frac{\partial H_1}{\partial z} + \frac{1}{S_z}\frac{\partial H_2}{\partial z}\right) \quad (5.106d)$$

A seguir, dois exemplos são feitos, aplicando a formulação CPML nas equações (5.105a) e (5.105b), de forma a obter:

$$\varepsilon\frac{\partial E_Z}{\partial t} = \frac{2}{3}\left(\hat{S}_1 * \frac{\partial H_1}{\partial n_1} + \hat{S}_2 * \frac{\partial H_2}{\partial n_2} + \hat{S}_3 * \frac{\partial H_3}{\partial n_3}\right) \quad (5.107)$$

$$\mu\frac{\partial H_1}{\partial t} = \hat{S}_1 * \frac{\partial E_Z}{\partial n_1} + \frac{1}{\sqrt{3}}\left(\hat{S}_z * \frac{\partial E_2}{\partial z} + \hat{S}_z * \frac{\partial E_3}{\partial z}\right) \quad (5.108)$$

Onde o símbolo " $*$ " novamente indica convolução no tempo e o termo $\hat{S}_w = \hat{S}_w(t)$ é a transformada inversa de Fourier de $(1 / S_w)$. O procedimento é o mesmo da seção 5.3, mas com $w \in \{1, 2, 3, z\}$. As condutividades também serão graduadas usando o mesmo procedimento da seção 5.4. Desta forma, a partir da equação (5.107), aplicando a formulação CPML em forma discreta na equação (3.18), tem-se:

$$E_Z\big|_{i,j,k}^{n+1} = C_{EZ0}.E_Z\big|_{i,j,k}^{n} + C_{EZ}\frac{1}{\kappa_1(i,j)}\left(H_1\big|_{i,j-1,k}^{n+\frac{1}{2}} - H_1\big|_{i,j+1,k}^{n+\frac{1}{2}}\right) + C_{EZ}\frac{1}{\kappa_2(i,j)}\left(H_2\big|_{i+\frac{1}{2},j-\frac{1}{2},k}^{n+\frac{1}{2}} - H_2\big|_{i-\frac{1}{2},j+\frac{1}{2},k}^{n+\frac{1}{2}}\right) + C_{EZ}\frac{1}{\kappa_3(i,j)}\left(H_3\big|_{i+\frac{1}{2},j+\frac{1}{2},k}^{n+\frac{1}{2}} - H_3\big|_{i-\frac{1}{2},j-\frac{1}{2},k}^{n+\frac{1}{2}}\right) + C_{EZ}\left(\varphi_{Ez,h1}\big|_{i,j,k}^{n+1/2} + \varphi_{Ez,h2}\big|_{i,j,k}^{n+1/2} + \varphi_{Ez,h3}\big|_{i,j,k}^{n+1/2}\right) \quad (5.109a)$$

$$\varphi_{Ez,h1}\big|_{i,j,k}^{n+1/2} = b_1(i,j).\varphi_{Ez,h1}\big|_{i,j,k}^{n-1/2} + C_1(i,j).\left(H_1\big|_{i,j-1,k}^{n+\frac{1}{2}} - H_1\big|_{i,j+1,k}^{n+\frac{1}{2}}\right) \quad (5.109b)$$

$$\varphi_{Ez,h2}\big|_{i,j,k}^{n+1/2} = b_2(i,j).\varphi_{Ez,h2}\big|_{i,j,k}^{n-1/2} + C_2(i,j).\left(H_2\big|_{i+\frac{1}{2},j-\frac{1}{2},k}^{n+\frac{1}{2}} - H_2\big|_{i-\frac{1}{2},j+\frac{1}{2},k}^{n+\frac{1}{2}}\right) \quad (5.109c)$$

$$\varphi_{Ez,h3}\big|_{i,j,k}^{n+1/2} = b_3(i,j).\varphi_{Ez,h3}\big|_{i,j,k}^{n-1/2} + C_3(i,j).\left(H_3\big|_{i+\frac{1}{2},j+\frac{1}{2},k}^{n+\frac{1}{2}} - H_3\big|_{i-\frac{1}{2},j-\frac{1}{2},k}^{n+\frac{1}{2}}\right) \quad (5.109d)$$

Os termos b_1, b_2 e b_3 são obtidos da equação (5.47) e os termos C_1, C_2 e C_3 são obtidos da equação (5.48). Os termos κ_1, κ_2 e κ_3, se forem usados, são obtidos da equação (5.52b). É bom lembrar que os termos acima e as variáveis $\varphi_{Ez,hw}$ precisam ser armazenadas apenas nas PMLs. Para o campo H_1, a partir da equação (5.108), aplicando a formulação CPML em forma discreta na equação (3.27) tem-se:

$$H_1\big|_{i,j-1,k}^{n+\frac{1}{2}} = C_{H10}.H_1\big|_{i,j-1,k}^{n-\frac{1}{2}} + C_{HA}\frac{1}{\kappa_1(i,j)}\left(E_z\big|_{i,j-2,k}^{n} - E_z\big|_{i,j,k}^{n}\right) + C_{HB}\frac{1}{\kappa_z(k)}\left(E_2\big|_{i+\frac{1}{6},j-\frac{3}{2},k+\frac{1}{2}}^{n} - E_2\big|_{i+\frac{1}{6},j-\frac{3}{2},k-\frac{1}{2}}^{n}\right) + C_{HB}\frac{1}{\kappa_z(k)}\left(E_3\big|_{i+\frac{1}{6},j-\frac{1}{2},k+\frac{1}{2}}^{n} - E_3\big|_{i+\frac{1}{6},j-\frac{1}{2},k-\frac{1}{2}}^{n}\right) + C_{HA}.\varphi_{H1,ez}\big|_{i,j-1,k}^{n} + C_{HB}.\left(\varphi_{H1,e2}\big|_{i,j-1,k}^{n} + \varphi_{H1,e3}\big|_{i,j-1,k}^{n}\right) \quad (5.110a)$$

$$\varphi_{H1,ez}\big|_{i,j-1,k}^{n} = b_1(i,j).\varphi_{H1,ez}\big|_{i,j-1,k}^{n-1} + C_1(i,j).\left(E_z|_{i,j-2,k}^{n} - E_z|_{i,j,k}^{n}\right) \quad (5.110b)$$

$$\varphi_{H1,e2}\big|_{i,j-1,k}^{n} = b_z(k).\varphi_{H1,e2}\big|_{i,j-1,k}^{n-1} + C_z(k).\left(E_2|_{i+\frac{1}{6},j-\frac{3}{2},k+\frac{1}{2}}^{n} - E_2|_{i+\frac{1}{6},j-\frac{3}{2},k-\frac{1}{2}}^{n}\right) \quad (5.110c)$$

$$\varphi_{H1,e3}\big|_{i,j-1,k}^{n} = b_z(k).\varphi_{H1,e3}\big|_{i,j-1,k}^{n-1} + C_z(k).\left(E_3|_{i+\frac{1}{6},j-\frac{1}{2},k+\frac{1}{2}}^{n} - E_3|_{i+\frac{1}{6},j-\frac{1}{2},k-\frac{1}{2}}^{n}\right) \quad (5.110d)$$

Os termos b_1 e b_z são obtidos da equação (5.47) e os termos C_1 e C_z são obtidos da equação (5.48). Os termos κ_1 e κ_z, se forem usados, são obtidos da equação (5.52b). A implementação computacional da formulação CPML é relativamente simples, necessitando pequena quantidade de memória extra, principalmente quando o espaço de simulação é grande. Nas simulações realizadas usou-se sempre, por simplicidade, $\kappa_w = 1$ e $a_w = 0$ nos termos S_w.

5.5.3 Graduação otimizada de condutividade para as PMLs da grade de prismas hexagonais

No método FDTD Yee o termo $S_w = S_w(w)$ onde $w \in \{x, y, z\}$ é graduado de forma simples. Por exemplo, para uma interface perpendicular à direção w é suficiente graduar o termo S_w na direção w. No entanto, para o método FDTD aplicado a grade de prismas hexagonais o termo $S_w = S_w(n_w$ ou $w)$ onde $w \in \{1, 2, 3$ ou $z\}$ será graduado de forma simples apenas na direção z, que é ortogonal às outras três direções n_1, n_2, e n_3 que estão no plano x-y. Para que a graduação dos termos S_w seja atendida simultaneamente nas três direções (n_1, n_2, n_3), conforme a análise teórica realizada nas subseções 5.5.1 e 5.5.2, é necessário pensar numa graduação que seja adequada à geometria da grade de hexágonos dos modos TM_z e TE_z. A Figura 5.5 ilustra esta graduação que parece uma moldura de quadro, mas que usa ângulos de 60° (ao invés de 45°) nos cantos para produzir uma perfeita continuidade da graduação de condutividade paras todas as posições dos oitos campos nas PMLs no plano x-y. A moldura com ângulo de canto a 60° implica que na direção y a PML é 1,73 vezes mais larga que na direção x. Um detalhe importante é que as variáveis φ (usadas nas PMLs) na direção z são calculadas na posições (i, j, k) tal que as posições

(*i*, *j*) pertencem ao plano x-y total, mas restringindo o valor de *k* tal que $0 \leq k \leq k_{base}$ (onde k_{base} indica o final da PML na direção *z* na base ou parte inferior da grade 3D) e $k_{topo} \leq k \leq k_{max}$ (onde k_{topo} e k_{max} indicam o início e o final da PML na direção *z*, respectivamente, no topo ou parte superior da grade 3D). No entanto, as variáveis φ da formulação CPML nas direções n_1, n_2, e n_3 precisam ser calculadas em todas as posições (*i*, *j, k*) tal que as posições (*i*, *j*) pertencem unicamente a PML (moldura no plano x-y, como mostrado na Figura 5.5) e incluindo todo *k* na direção *z* tal que $k_{base} < k < k_{topo}$.

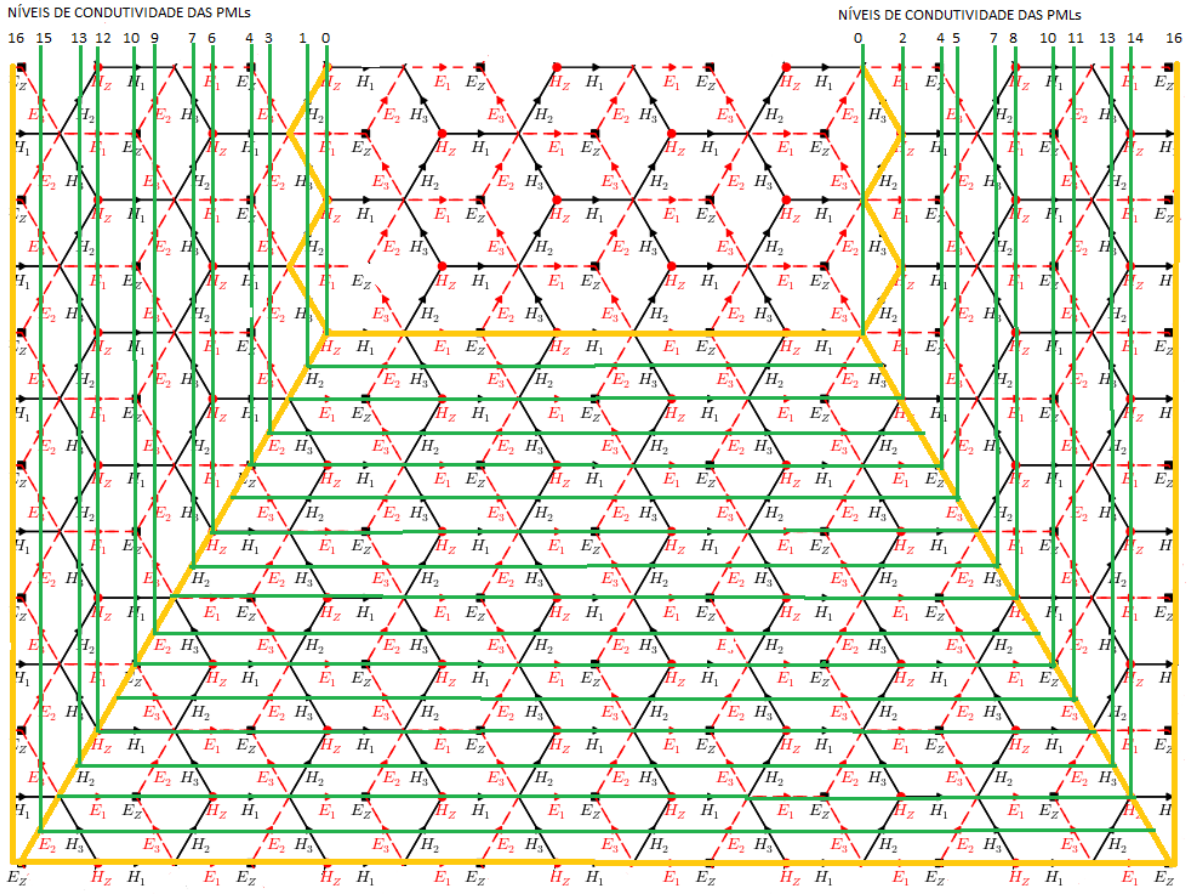


Figura 5.5 - Graduação das condutividades nas PMLs no plano x-y da grade de primas hexagonais.

5.5.4 Simulações da formulação CPML para grade de prismas hexagonais

O objetivo desta subseção é demonstrar o funcionamento satisfatório da formulação CPML desenvolvida para a grade de primas hexagonais. Medidas de reflexão (INAN; MARSHALL, 2011) são feitas escolhendo quatro pontos (A, B, C, D) na grade de prismas hexagonais que são localizados próximos às PMLs. As posições destes pontos são escolhidas em relação à posição central onde se é aplicada a fonte

senoidal do tipo *soft* (ver subseção 3.3.1). Inicialmente o espaço de simulação é aumentado de forma que haja propagação da onda sem sofrer reflexão e, consequentemente, sem o uso das PMLs. O nível da onda se propagando em cada ponto (A, B, C, D) é gravado para todos os passos de tempo. A seguir o espaço de simulação é encurtado e utiliza-se as PMLs para absorver as ondas incidentes. Novamente o nível da onda em cada ponto (A, B, C, D), que mantem a mesma posição relativa à fonte senoidal da simulação sem PMLs, é gravado para o mesmo número de passos de tempo usado na simulação sem PMLs. O módulo do coeficiente de reflexão instantâneo em cada ponto (A, B, C, D) pode ser calculado utilizando-se a seguinte equação:

$$\Gamma(i) = \left| \frac{E_{Z,com\ PML}(i) \ - \ E_{Z,sem\ PML}(i)}{E_{Z,sem\ PML}(i)} \right| \tag{5.111}$$

onde o passo de tempo (*i*) varia de zero ao valor máximo (*n*). Para obter o coeficiente de reflexão médio é feita uma média para todos os *k* passos de tempo em que o coeficiente de reflexão instantâneo Γ(*i*) é diferente de zero. Assim, tem-se:

$$\Gamma_{med} = \frac{\left[\sum_{i=1}^{k} \Gamma(i) \right]}{k} \ \text{para todo:} \ \Gamma(i) > 0; \ \text{e com:} \ k < n \tag{5.112}$$

Simulações são feitas usando PMLs para a grade de prismas hexagonais com N_X = 185, N_Y = 369, N_Z = 185, número de pontos por comprimento de onda N_λ = 20, número de Courant S_{CFL} = 1/√3,3334, razão R = 1,1547, lado do hexágono maior Δd = 1 m e número de passos de tempo n = 257. A espessura da PML na direção x é $PMLx = n_x$. Δx = 20. Δx = 17,3 m. A espessura da PML na direção y é $PMLy = n_y$. Δy = (3.n_x + 2).Δy = 62. Δy = 31 m. A espessura da PML na direção z é $PMLz = n_x$.Δz = 20. Δz = 17,3 m. O coeficiente de reflexão teórico usado é Γ = 10^{-6} e a ordem do polinômio usado para graduação de condutividade das PMLs no plano x-y é m = 1,9 e na direção z é m = 3,9. Para a simulação sem PMLs é usado N_X = 465, N_Y = 801, N_Z = 465 e os demais parâmetros são iguais àqueles da simulação com PMLs. A Figura 5.6 mostra a onda senoidal sendo absorvida pelas PMLs no plano x-y. O nível da onda senoidal é reduzido por 3,0 10^{-4} para facilitar a sua visualização. As posições dos pontos A, B e C são indicadas nesta figura.

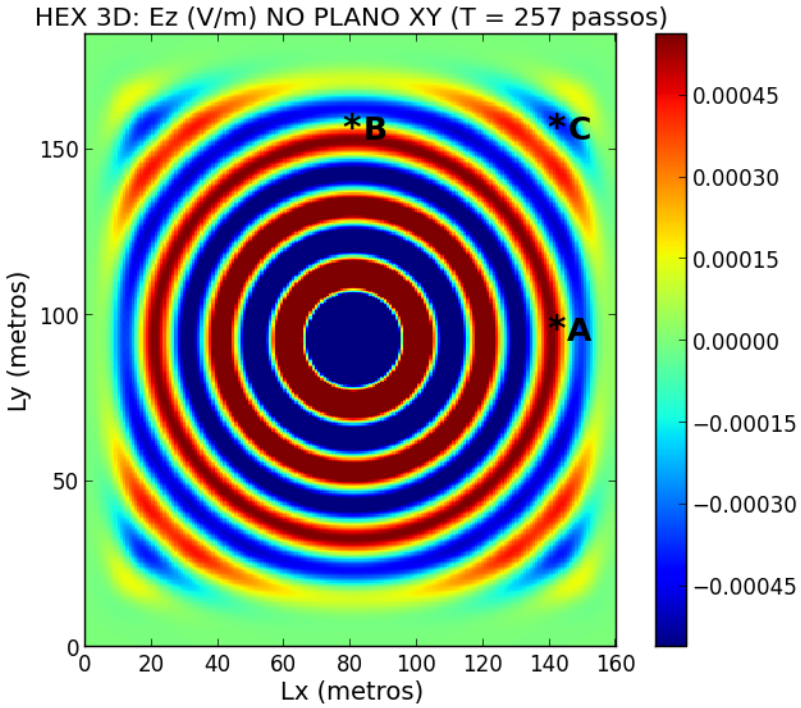


Figura 5.6 - Propagação da onda senoidal no plano x-y com pontos de medida de reflexão A, B e C.

A Figura 5.7 mostra os gráficos 1D nas direções positivas *x* (0°), *y* (90°) e *d* (30°) a partir da posição central do gráfico 2D da Figura 5.6. Observa-se uma absorção gradual e efetiva das ondas senoidais pelas PMLs.

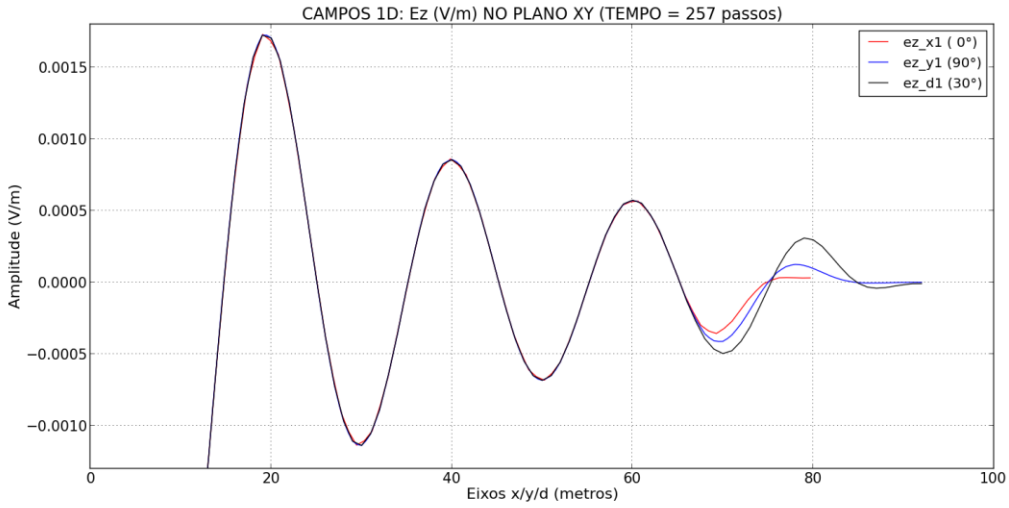


Figura 5.7 - Propagação da onda senoidal nas direções *x* (0°), *y* (90°) e *d* (30°) no plano x-y.

A Figura 5.8 mostra a onda senoidal sendo absorvida pelas PMLs no plano x-z. O nível da onda senoidal é reduzido por $3{,}0\ 10^{-4}$ para facilitar a sua visualização. A posição do ponto D é indicada nesta figura.

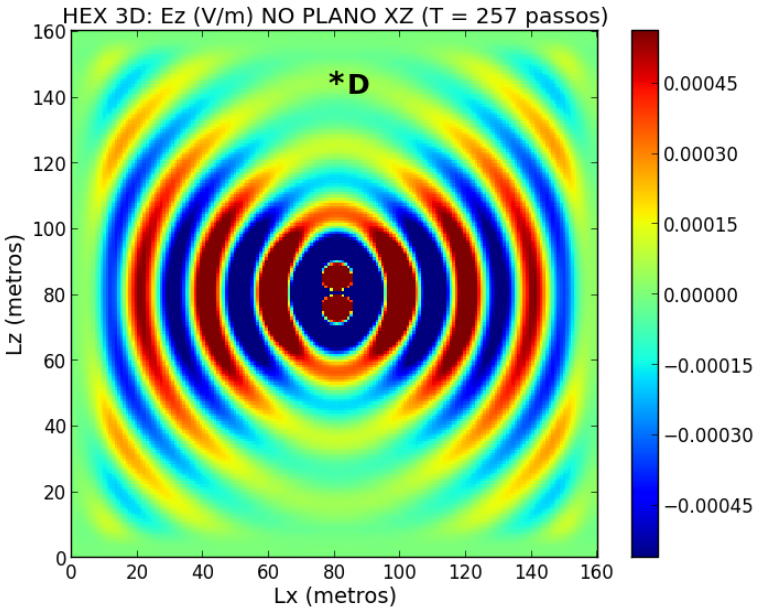


Figura 5.8 - Propagação da onda senoidal no plano x-z com ponto de medida de reflexão D.

A Figura 5.9 mostra os gráficos 1D nas direções positivas *x* (0°), *z* (90°) e *d* (45°) a partir da posição central do gráfico 2D da Figura 5.8. Observa-se, também, uma absorção gradual e efetiva das ondas senoidais pelas PMLs.

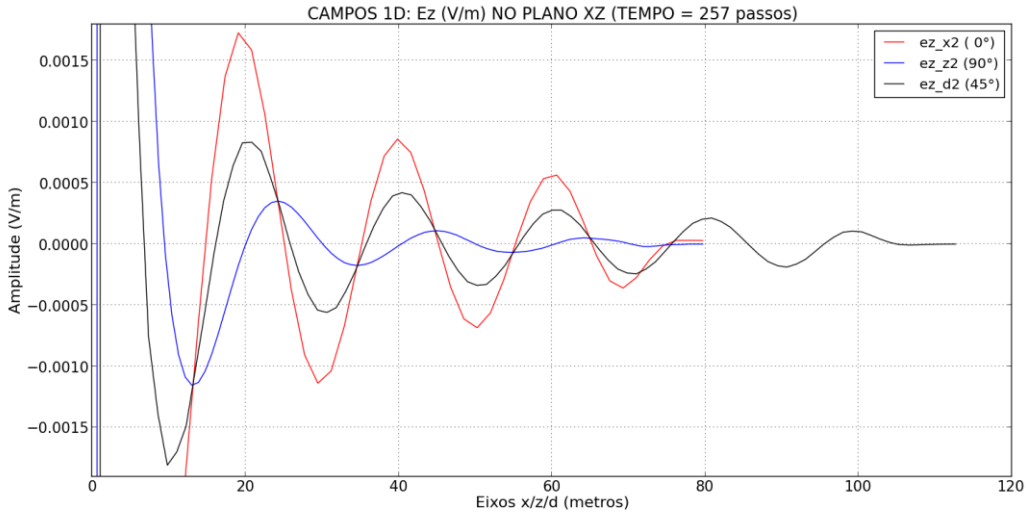
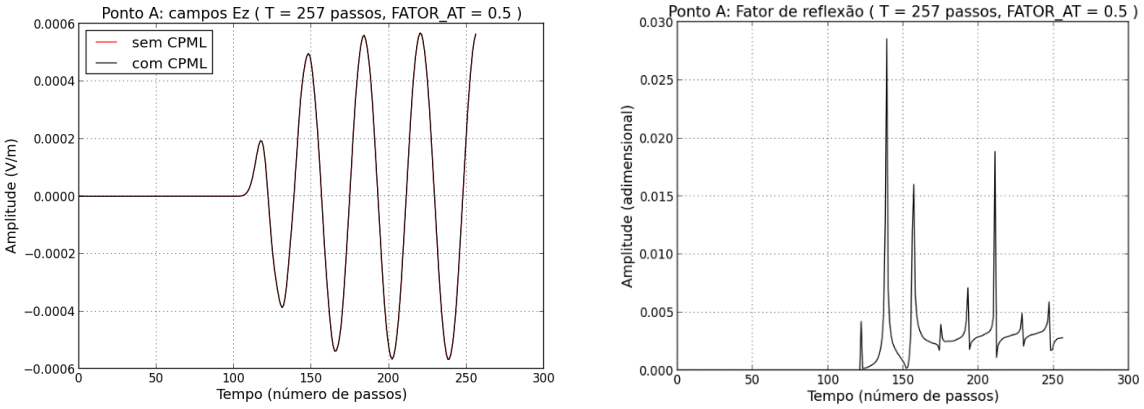


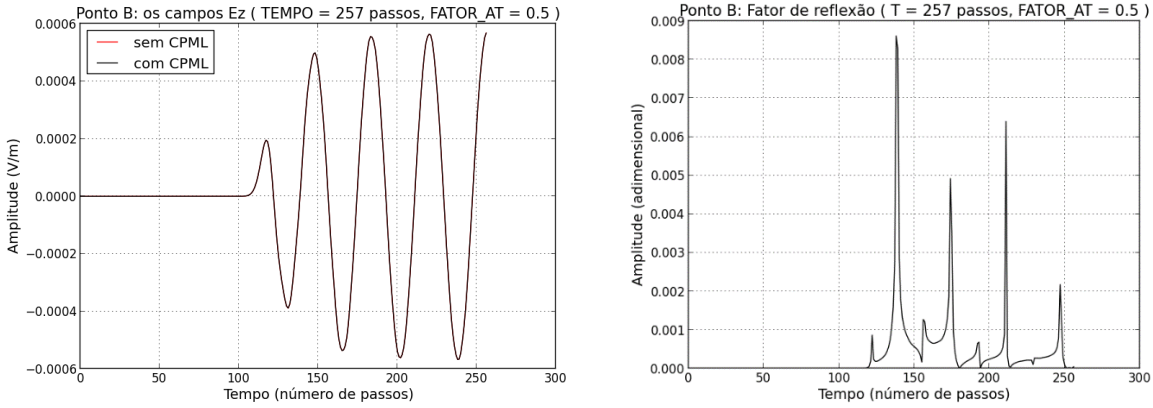
Figura 5.9 - Propagação da onda senoidal nas direções *x* (0°), *z* (90°) e *d* (45°) no plano x-z.

A seguir mostra-se gráficos no tempo (com superposição dos campos E_Z sem e com PMLs) e gráficos de coeficiente de reflexão instantâneo para os quatro pontos A, B, C e D.



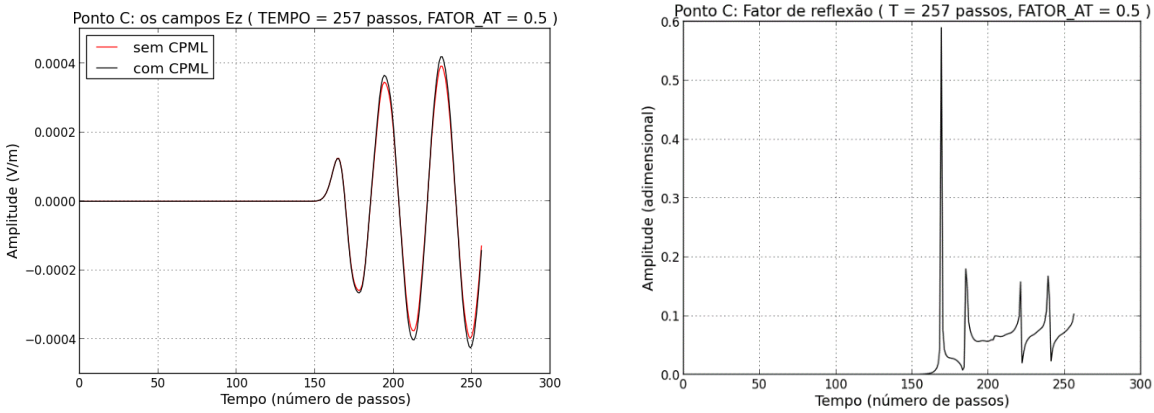
(a) (b)

Figura 5.10 - Ponto A: (a) campos E_Z com e sem PML; (b) coeficiente de reflexão instantâneo.



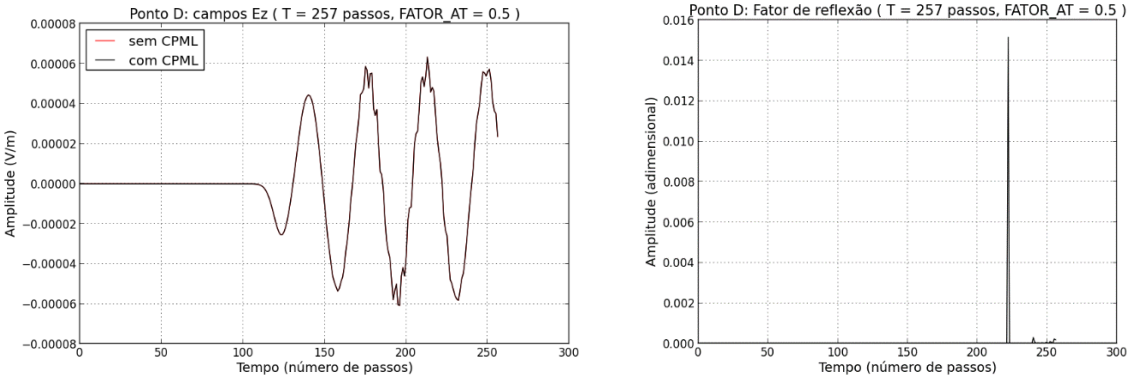
(a) (b)

Figura 5.11 - Ponto B: (a) campos E_Z com e sem PML; (b) coeficiente de reflexão instantâneo.



(a) (b)

Figura 5.12 - Ponto C: (a) campos E_Z com e sem PML; (b) coeficiente de reflexão instantâneo.



(a) (b)

Figura 5.13 - Ponto D: (a) campos E_Z com e sem PML; (b) coeficiente de reflexão instantâneo.

Os gráficos das Figuras 5.10 a 5.13 utilizam o valor ótimo da ordem do polinômio usado para graduação de condutividades nas PMLs no plano x-y, ou seja m = 1,9; para a direção z o valor usado é m = 3,9. Para determinar este valor ótimo foram feitas algumas simulações variando este parâmetro (m), e calculando o coeficiente de reflexão médio como mostrado na Tabela 5.1.

TABELA 5.1 - COMPARAÇÃO DE COEFICIENTES DE REFLEXÃO MÉDIO PARA GRADE DE PRISMAS HEXAGONAIS.

Coeficiente de reflexão médio	Ponto A	Ponto B	Ponto C	Ponto D
m = 1,5	$5{,}35\ 10^{-3}$	$1{,}35\ 10^{-3}$	$7{,}35\ 10^{-2}$	$8{,}34\ 10^{-3}$
m = 1,8	$2{,}78\ 10^{-3}$	$5{,}79\ 10^{-4}$	$4{,}48\ 10^{-2}$	$5{,}60\ 10^{-3}$
m = 1,9	$2{,}60\ 10^{-3}$	$5{,}23\ 10^{-4}$	$4{,}18\ 10^{-2}$	$4{,}50\ 10^{-3}$
m = 2,0	$2{,}88\ 10^{-3}$	$5{,}21\ 10^{-4}$	$4{,}11\ 10^{-2}$	$3{,}48\ 10^{-3}$
m = 2,5	$4{,}71\ 10^{-3}$	$4{,}94\ 10^{-4}$	$4{,}69\ 10^{-2}$	$4{,}89\ 10^{-4}$
m = 3,0	$5{,}84\ 10^{-3}$	$7{,}50\ 10^{-4}$	$4{,}88\ 10^{-2}$	$2{,}48\ 10^{-4}$
m = 3,9	$5{,}31\ 10^{-3}$	$6{,}01\ 10^{-4}$	$4{,}80\ 10^{-2}$	$1{,}25\ 10^{-4}$

Na Tabela 5.1 verifica-se que o ponto B é aquele que mede o menor coeficiente de reflexão médio no plano x-y, pois está posicionado na direção y onde a espessura da PML é 1,73 vezes maior que a espessura da PML na direção x. O ponto C é aquele que produz as piores medidas, pois está localizado próximo ao canto superior direito do contorno das PMLs no plano x-y e recebe ondas refletidas com intensidade não desprezível das direções x e y (ver a Figura 5.12a). O ponto A localizado próximo à PML na direção x produz um coeficiente de reflexão médio que é menor que o do

ponto C, mas maior que aquele do ponto B, pois possui uma menor espessura de PML. No entanto, este ponto A, por ter um peso maior no desempenho global das PMLs, será a referência escolhida para minimizar a reflexão. Assim, para o ponto A o mínimo de reflexão será produzido para um valor de ordem de polinômio aproximadamente igual a 1,9. O ponto D para as PMLs na direção *z* apresentou um comportamento diferente das PMLs no plano x-y, podendo-se escolher uma ordem de polinômio mais alta como m = 3,0 ou 3,9 que minimiza o coeficiente de reflexão médio.

Para comparar o desempenho das PMLs usadas na grade de prismas hexagonais com as PMLs da grade de hexaedros (método Yee), um procedimento similar é aplicado na grade Yee com medidas em quatro pontos (A, B, C, D) com posições aproximadamente iguais àquelas utilizadas na grade de prismas hexagonais. Simulações são feitas usando PMLs para a grade Yee com $N_X = N_Y = N_Z = 159$, número de pontos por comprimento de onda $N_\lambda = 20$, número de Courant $S_{CFL} = 1/\sqrt{3}$, $\Delta x = \Delta y = \Delta z = 1$ m e número de passos de tempo $n = 243$. As espessura das PMLs nas direções *x, y* e *z* são dadas por $PMLx = PMLy = PMLz = n_x$. $\Delta x = 18$. $\Delta x = 18$ m. Esta espessura é próxima àquela usada pela grade de prismas hexagonais na direções *x* e *z*. O coeficiente de reflexão teórico usado é $\Gamma = 10^{-6}$ e a ordem do polinômio usado para graduação de condutividade das PMLs nas direções *x, y* e *z* é m = 3,9. Para a simulação sem PMLs é usado $N_X = N_Y = N_Z = 401$ e os demais parâmetros são os mesmos da simulação com PMLs.

A Figura 5.14 mostra a onda senoidal sendo absorvida pelas PMLs no plano x-y e as posições dos pontos A, B e C. O nível da onda senoidal nesta figura é reduzido por $3{,}0\ 10^{-4}$ para facilitar a sua visualização. A Figura 5.15 mostra a onda senoidal sendo absorvida pelas PMLs no plano x-z e a posição do ponto D. O nível da onda senoidal nesta figura é também reduzido por $3{,}0\ 10^{-4}$ para facilitar a sua visualização.

Os gráficos das Figuras 5.14 e 5.15 utilizam a ordem do polinômio ótimo m = 3,9 que minimiza os coeficientes de reflexão médio nos pontos A, B, C e D. Para determinar este valor ótimo foi feita a Tabela 5.2 variando a ordem do polinômio (m).

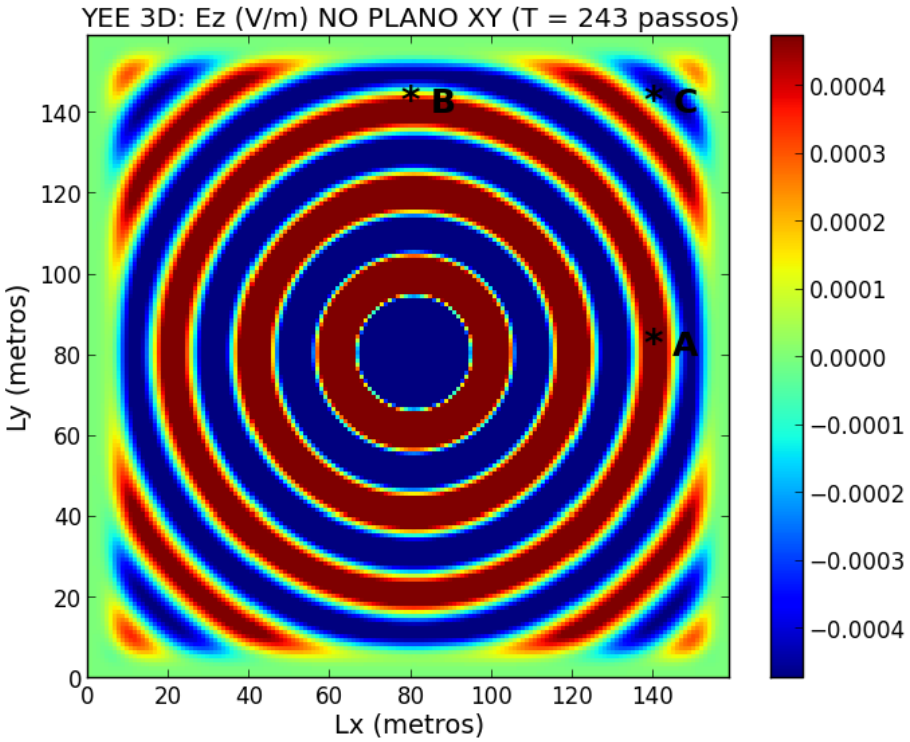


Figura 5.14 - Propagação da onda senoidal no plano x-y com pontos de medida de reflexão A, B e C.

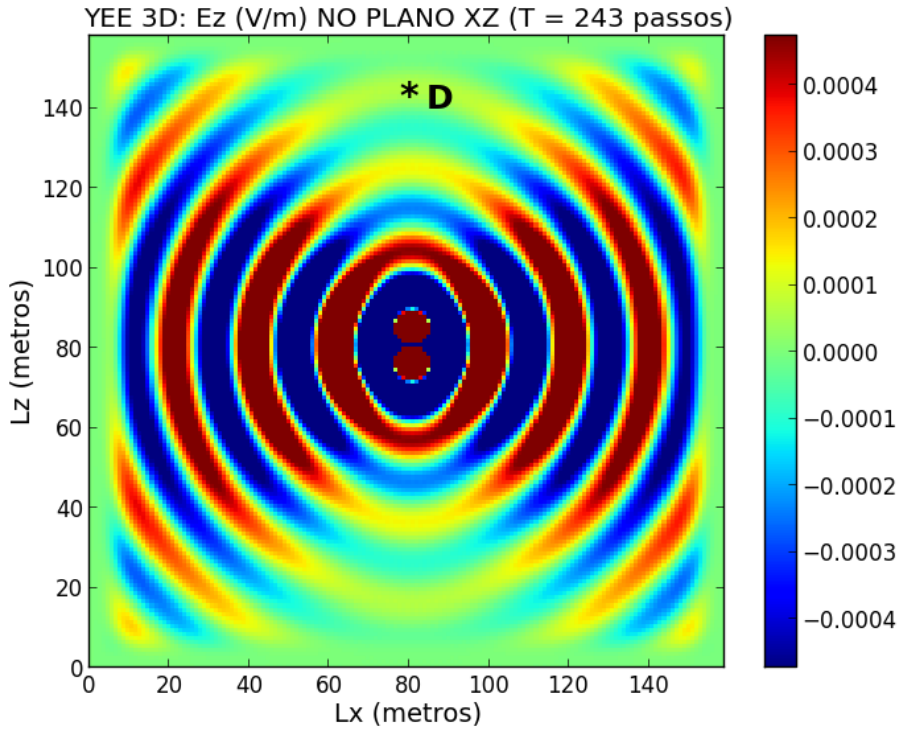


Figura 5.15 - Propagação da onda senoidal no plano x-z com ponto de medida de reflexão D.

TABELA 5.2 - COMPARAÇÃO DE COEFICIENTES DE REFLEXÃO MÉDIO PARA GRADE YEE.

Coeficiente de reflexão médio	Ponto A	Ponto B	Ponto C	Ponto D
m = 1,9	$7{,}94\ 10^{-4}$	$7{,}94\ 10^{-4}$	$5{,}59\ 10^{-4}$	$2{,}36\ 10^{-3}$
m = 3,0	$3{,}14\ 10^{-4}$	$3{,}14\ 10^{-4}$	$4{,}45\ 10^{-4}$	$1{,}70\ 10^{-3}$
m = 3,8	$1{,}35\ 10^{-4}$	$1{,}35\ 10^{-4}$	$3{,}82\ 10^{-4}$	$9{,}81\ 10^{-4}$
m = 3,9	$1{,}31\ 10^{-4}$	$1{,}31\ 10^{-4}$	$3{,}64\ 10^{-4}$	$8{,}31\ 10^{-4}$
m = 4,0	$1{,}54\ 10^{-4}$	$1{,}54\ 10^{-4}$	$3{,}51\ 10^{-4}$	$9{,}48\ 10^{-4}$
m = 4,2	$2{,}12\ 10^{-4}$	$2{,}12\ 10^{-4}$	$3{,}17\ 10^{-4}$	$1{,}17\ 10^{-3}$
m = 4,5	$2{,}91\ 10^{-4}$	$2{,}91\ 10^{-4}$	$2{,}85\ 10^{-4}$	$1{,}44\ 10^{-3}$

Na Tabela 5.2 as posições A e B produzem o mesmo coeficiente de reflexão em função da ordem do polinômio (m), pois as espessuras das PMLs são iguais nas direções x e y. Na posição C os valores medidos do coeficiente de reflexão são também baixos. A posição D foi a que produziu as piores medidas de reflexão. O menor valor de coeficiente de reflexão medido para as posições A e B foi para m = 3,9; os valores medidos nas posições C e D, para esta mesma ordem do polinômio (m) são 2,78 e 6,34 vezes maiores, respectivamente, que aquele da posições A ou B. Na comparação da Tabela 5.1 (grade de prismas hexagonais) com a Tabela 5.2 (grade de hexaedros) é evidente o melhor desempenho das PMLs no método FDTD Yee. No entanto, os resultados obtidos na grade de prismas hexagonais são satisfatórios para muitas aplicações práticas, e podem ser aperfeiçoados com o custo de se usar PMLs mais espessas.

6 FORMULAÇÃO FDTD COM REGIÕES DE CAMPOS TOTAL E ESPALHADO NA GRADE DE PRISMAS HEXAGONAIS

6.1 FORMULAÇÃO FDTD COM REGIÕES DE CAMPOS TOTAL E ESPALHADO

Em muitas aplicações práticas a fonte de irradiação (antena) está muito distante do objeto em que a onda eletromagnética incide. Assim, pode ser conveniente definir uma onda incidente no espaço de simulação, de forma a emular uma fonte de irradiação externa (distante). Normalmente, esta onda incidente é considerada plana e pode atingir o espaço de simulação com as características desejadas pelo usuário como: forma de onda arbitrária, duração, ângulo de incidência e ângulo de polarização. Embora existam algumas formas diferentes de criar esta onda plana (TAFLOVE; HAGNESS, 2005), neste texto será usada apenas a formulação considerada a mais eficiente e fácil de usar, conhecida em inglês como *Total-Field / Scattered-Field* (TF/SF). A formulação TF/SF é baseada na linearidade das equações de Maxwell. A linearidade significa que os parâmetros ε, μ e σ não dependem das intensidades dos campos elétrico ou magnético. Isto é, normalmente, válido na maioria das aplicações práticas. Esta linearidade admite que o campo elétrico total ($\boldsymbol{E_{total}}$) e o campo magnético total ($\boldsymbol{H_{total}}$) pode ser decompostos como:

$$\boldsymbol{E_{total}} = \boldsymbol{E_{inc}} + \boldsymbol{E_{esp}} \tag{6.1a}$$

$$\boldsymbol{H_{total}} = \boldsymbol{H_{inc}} + \boldsymbol{H_{esp}} \tag{6.1b}$$

Nas equações acima os campos elétrico incidente ($\boldsymbol{E_{inc}}$) e magnético incidente ($\boldsymbol{H_{inc}}$) são considerados conhecidos para todos os pontos na grade 2D ou 3D e também para todos os tempos. Estes campos são os que existem no vácuo, na ausência de qualquer material. Os campos elétrico espalhado ($\boldsymbol{E_{esp}}$) e magnético espalhado ($\boldsymbol{H_{esp}}$) são os campos não conhecidos e resultam da iteração dos campos incidentes ($\boldsymbol{E_{inc}}$ e $\boldsymbol{H_{inc}}$) com quaisquer materiais presentes na grade. O espaço de simulação (grade 2D ou 3D) é dividido em duas regiões, como mostrado na Figura 6.1. Na região 1 estão presente os campos totais e a estrutura (objetos formados de um ou mais materiais) que interagem com os campos incidentes e geram os campos espalhados. Na região

2 existem apenas os campos espalhados. A região 2 é truncada com camadas de absorção (PMLs) que emulam um espaço infinito, como já analisado no capitulo 5. As regiões 1 e 2 são separadas por uma interface virtual (não-física) que serve para conectar os campos em cada região e, consequentemente, gerar a onda incidente.

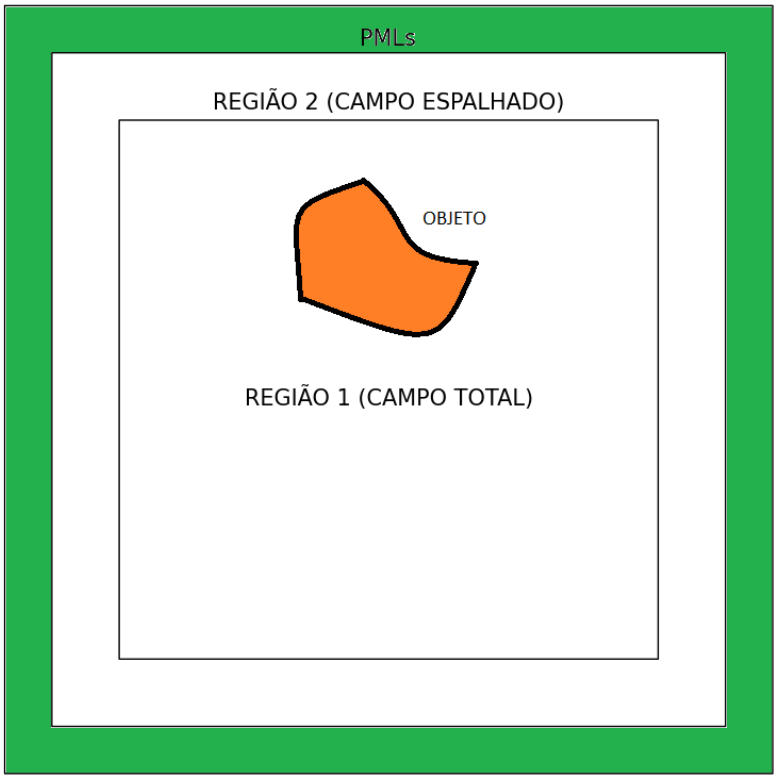


Figura 6.1 - Interface virtual entre regiões de campo total e campo espalhado.

Esta interface virtual constitui a interface entre regiões 1 e 2, as quais contêm os campos *E* e *H* que para serem calculados, na formulação FDTD, usam diferenças finitas espaciais com componentes adjacentes de campos (*H* ou *E*) num processo de marcha no tempo. Quando a diferença finita espacial requerida é tomada através da interface virtual um problema de consistência surge. Na região 1 (do lado adjacente à interface virtual) é admitido serem totais os campos elétricos ou magnéticos, enquanto na região 2 (do outro lado adjacente a interface virtual) é admitido serem espalhados os campos elétricos ou magnéticos. Assim, seria inconsistente realizar uma diferença finita espacial entre um valor de campo total (na região 1) com um valor de campo espalhado (na região 2). Para ilustrar este problema, considere uma grade 1D formada de componentes E_Z e H_Y propagando-se na direção $+x$ como mostrado na Figura 6.2.

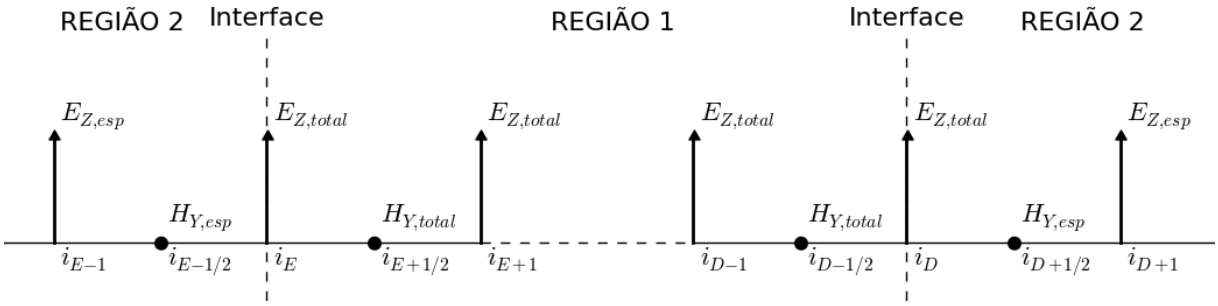


Figura 6.2 - Grade 1D formada de componentes E_Z e H_Y propagando-se na direção +*x*.

O campo elétrico E_Z é definido pela seguinte equação de diferença finita:

$$E_Z|_i^{n+1} = E_Z|_i^n + \frac{\Delta t}{\varepsilon_0 \Delta x}\left(H_Y|_{i+1/2}^{n+1/2} - H_Y|_{i-1/2}^{n+1/2}\right) \tag{6.2}$$

A equação (6.2) pode ser usada de forma consistente quando a componente E_Z e as duas componentes H_Y estiverem totalmente na região 1 ou totalmente na região 2. No entanto, uma inconsistência surgirá para a componente E_Z na posição i_E (ver Figura 6.2), assim que a equação (6.2) será reescrita como:

$$E_{Z,total}\Big|_{i_E}^{n+1} = E_{Z,total}\Big|_{i_E}^{n} + \frac{\Delta t}{\varepsilon_0 \Delta x}\left(H_{Y,total}\Big|_{i_E+1/2}^{n+1/2} - H_{Y,esp}\Big|_{i_E-1/2}^{n+1/2}\right) \tag{6.3}$$

Para corrigir o termo inconsistente ($H_{Y,\,esp}$) na equação (6.3) é suficiente somar o campo incidente ($H_{Y,inc}$), admitido conhecido, na posição (i_E – 1/2). Logo, tem-se:

$$E_{Z,total}\Big|_{i_E}^{n+1} = E_{Z,total}\Big|_{i_E}^{n} + \frac{\Delta t}{\varepsilon_0 \Delta x}\left(H_{Y,total}\Big|_{i_E+1/2}^{n+1/2} - H_{Y,esp}\Big|_{i_E-1/2}^{n+1/2} - H_{Y,inc}\Big|_{i_E-1/2}^{n+1/2}\right) \tag{6.4}$$

Um raciocínio semelhante aplica-se ao campo E_Z na posição i_D. Para o campo magnético H_Y é definida uma equação de diferença finita dada por:

$$H_Y|_{i-1/2}^{n+1/2} = H_Y|_{i-1/2}^{n-1/2} + \frac{\Delta t}{\mu_0 \Delta x}(E_Z|_i^n - E_Z|_{i-1}^n) \tag{6.5}$$

A equação (6.5) pode ser usada de forma consistente quando a componente H_Y e as duas componentes E_Z estiverem totalmente na região 1 ou totalmente na região 2. No

entanto, uma inconsistência surgirá para a componente H_Y na posição (i_E – 1/2), assim que a equação (6.5) será reescrita como:

$$H_{Y,esp}\Big|_{i_E-1/2}^{n+1/2} = H_{Y,esp}\Big|_{i_E-1/2}^{n-1/2} + \frac{\Delta t}{\mu_0 \Delta x}\left(E_{Z,total}\Big|_{i_E}^{n} - E_{Z,esp}\Big|_{i_E-1}^{n}\right) \quad (6.6)$$

Para corrigir o termo inconsistente ($E_{Z,\,total}$) na equação (6.6) é suficiente subtrair o campo elétrico ($E_{Z,\,inc}$), admitido conhecido, na posição i_E, tal que:

$$H_{Y,esp}\Big|_{i_E-1/2}^{n+1/2} = H_{Y,esp}\Big|_{i_E-1/2}^{n-1/2} + \frac{\Delta t}{\mu_0 \Delta x}\left(E_{Z,total}\Big|_{i_E}^{n} - E_{Z,inc}\Big|_{i_E}^{n} - E_{Z,esp}\Big|_{i_E-1}^{n}\right) \quad (6.7)$$

Um raciocínio semelhante aplica-se ao campo magnético $H_{Y,\,esp}$ na posição (i_D + 1/2). O procedimento, usado neste simples exemplo (grade 1D), é também aplicável diretamente para grades 2D e 3D. Na grade 2D as interfaces virtuais entre regiões 1 e 2 serão formadas por quatro linhas, enquanto que na grade 3D as interfaces virtuais entre regiões 1 e 2 serão formadas por seis superfícies. É importante observar que as equações de diferença finita do método FDTD não precisam ser modificadas. Isto significa que os termos de correções (para campos totais ou espalhados) podem ser aplicados após a atualização no tempo das equações de diferença finita e unicamente para os campos adjacentes aos contornos entre regiões 1 e 2 que apresentarem inconsistência.

6.2 COMPARAÇÃO DA FORMULAÇÃO TF/SF ENTRE AMBOS OS MÉTODOS FDTD

Uma forma simples de implementar uma onda plana em grades 2D e 3D é utilizar uma grade 1D auxiliar. Para grade de hexaedros (método Yee) ou grade de prismas hexagonais, a grade auxiliar 1D é colocada na direção +*x* com componentes de campos elétrico (E_{Z1}) e magnético (H_{Y1}) cujas equações de diferença finita são definidas como:

$$E_{Z1}\Big|_{i}^{n+1} = E_{Z1}\Big|_{i}^{n} + C_{EZ1}\left(H_{Y1}\Big|_{i+1/2}^{n+1/2} - H_{Y1}\Big|_{i-1/2}^{n+1/2}\right) \quad (6.8)$$

$$H_{Y1}|_{i+1/2}^{n+1/2} = H_{Y1}|_{i+1/2}^{n-1/2} + C_{HY1}(E_{Z1}|_{i+1}^{n} - E_{Z1}|_{i}^{n}) \quad (6.9)$$

onde os coeficientes C_{EZ1} e C_{HY1} são dados por:

$$C_{EZ1} = S_{CFL}.Z.C_{ajuste} \quad (6.10)$$

$$C_{HY1} = \frac{S_{CFL}}{Z}.C_{ajuste} \quad (6.11)$$

A impedância Z é aquela do meio na interface virtual entre regiões 1 e 2, a qual é, normalmente, a impedância no vácuo (Z = 120.π ≈ 377 ohms). O número de Courant (S_{CFL}) deve ser igual àquele usado na simulação com grade de hexaedros (método Yee) ou aquele usado na grade de prismas hexagonais. É usado fator de ajuste C_{ajuste} = 1 para grade de hexaedros e fator de ajuste C_{ajuste} = 1 / cos(30°) para grade de prismas hexagonais. Os valores dos campos incidentes (na região 1) são definidos na grade de hexaedros em função dos campos da grade auxiliar 1D da seguinte forma:

$$E_{Z,inc}|_{i,j,k+\frac{1}{2}}^{n} = E_{Z1}|_{i}^{n} \quad (6.12a)$$

$$H_{Y,inc}|_{i+\frac{1}{2},j,k+\frac{1}{2}}^{n+1/2} = H_{Y1}|_{i+\frac{1}{2}}^{n+1/2} \quad (6.12b)$$

Na grade de prismas hexagonais os campos incidentes (na região 1) são definidos em função dos campos da grade auxiliar 1D da seguinte forma:

$$E_{Z,inc}|_{i,j,k}^{n} = E_{Z1}|_{i}^{n} \quad (6.13a)$$

$$H_{2,inc}|_{i+\frac{1}{2},j-\frac{1}{2},k}^{n+1/2} = H_{Y1}|_{i+\frac{1}{2}}^{n+1/2}.\cos 30^{\circ} \quad (6.13b)$$

$$H_{3,inc}|_{i+\frac{1}{2},j+\frac{1}{2},k}^{n+1/2} = H_{Y1}|_{i+\frac{1}{2}}^{n+1/2}.\cos 30^{\circ} \quad (6.13c)$$

A fonte de sinal (pulso Ricker ou senóide) é aplicada na posição i = 0 (lado inicial esquerdo da grade auxiliar 1D) para excitar o campo E_{Z1}. No lado direito da grade auxiliar 1D na posição final (N_X) é acrescentada uma camada de absorção (PML) com espessura suficiente (N_{PML}) para absorver a onda incidente, tal que o comprimento total da grade 1D é $M = N_X + N_{PML}$.

6.3 IMPLEMENTAÇÃO PRÁTICA DAS REGIÕES DE CAMPO TOTAL E CAMPO ESPALHADO NA GRADE DE PRISMAS HEXAGONAIS

A implementação das regiões de campo total (região 1) e campo espalhado (região 2) para o método FDTD Yee 3D é muito simples, pois o formato das superfícies virtuais entre regiões 1 e 2 é sempre retangular. No entanto, para o método FDTD com grade de prismas hexagonais a maior dificuldade foi encontrar o formato correto da interface virtual no plano x-y entre as regiões 1 e 2 de forma que fosse minimizado o vazamento do campo incidente da região 1 para a região 2. Testes iniciais usando grade 2D de hexágonos (para o modo TM_Z) demostraram que o formato ideal no plano x-y da interface virtual é aquele que segue os contornos dos hexágonos grandes (grade primária) como mostrado na Figura 6.3.

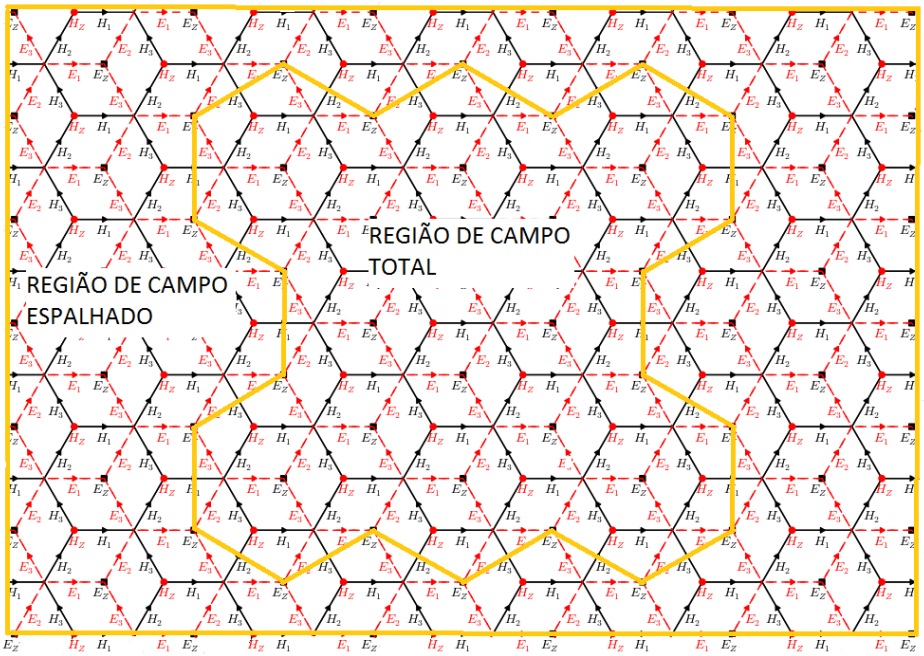
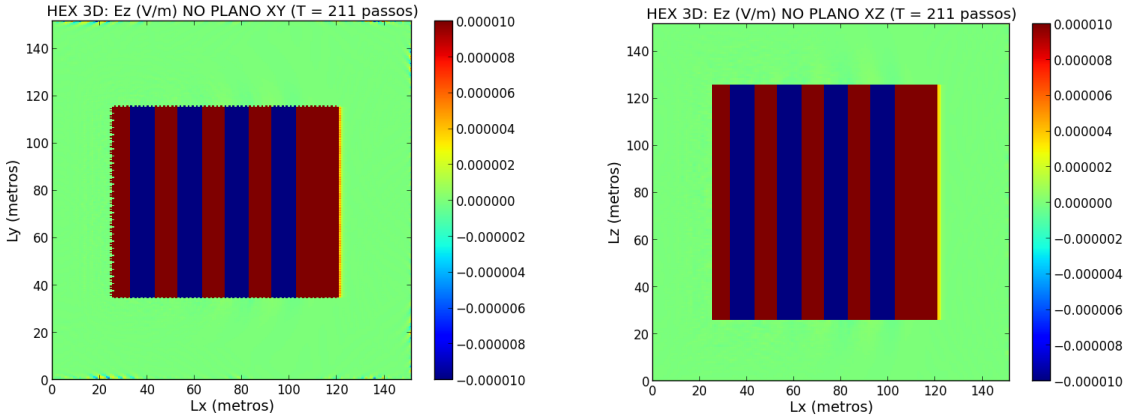


Figura 6.3 - Contorno virtual entre regiões de campo total e campo espalhado no plano x-y para a grade 2D de hexágonos.

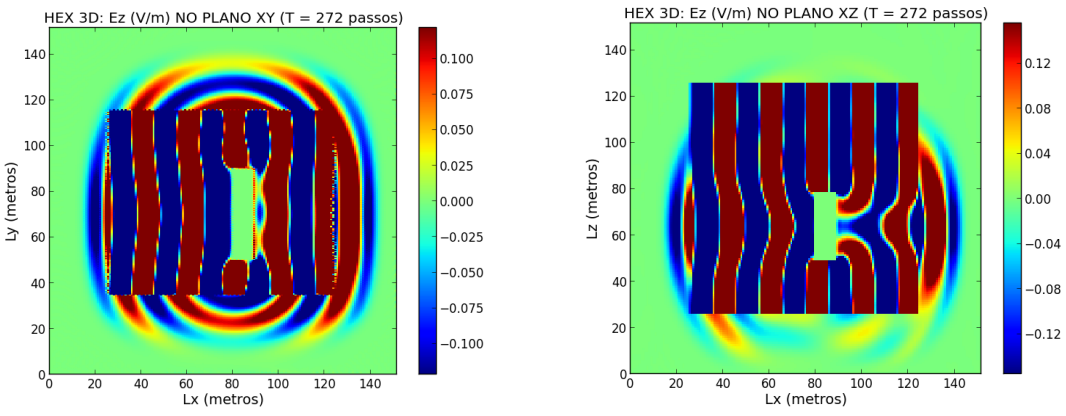
Outros formatos mais simples, como, por exemplo, dente de serra na direção y e reta na direção x (ver Figura 5.4 no capítulo 5) produziram vazamento não desprezível. As Figuras 6.4a e 6.4b ilustram uma onda plana incidente (senóide) na direção +x que é aplicada na grade de prismas hexagonais com N_X = 175, N_Y = 303, N_Z = 175, Δd = 1 m, R = 1,1547, número de Courant S_{CFL} = 1 / √3,3334, número de pontos por comprimento de onda N_λ = 20 e número de passos no tempo n = 211. O nível da onda senoidal foi reduzido por 10^{-5} em ambas as figuras sem vazamento perceptível.



(a) (b)

Figura 6.4 - Onda plana incidente na região de campo total: (a) plano x-y; (b) plano x-z.

Como ilustração é introduzida na região 1 uma placa retangular com condutor elétrico perfeito (altíssima condutividade elétrica, σ = 6 10^{14} S/m), para um número total de passos de tempo igual a 272 e demais parâmetros iguais aos da simulação da Figura 6.4. Assim, obtém-se o campo espalhado como mostrado nas Figuras 6.5a e 6.5b.



(a) (b)

Figura 6.5 - Ondas planas incidente e espalhada por placa retangular: (a) plano x-y; (b) plano x-z.

Na Figura 6.5 o nível da onda senoidal foi reduzido por 10^{-1} para permitir uma melhor visualização do campo espalhado. A seguir far-se-á comparações do espalhamento de campo produzido por uma placa retangular metálica e uma esfera metálica entre os dois métodos FDTD e os respectivos resultados analíticos da placa retangular e esfera.

6.4 COMPARAÇÕES DE ESPALHAMENTOS DE CAMPO PRODUZIDOS POR PLACA RETANGULAR METÁLICA

Um importante parâmetro é o espalhamento eletromagnético por um objeto (alvo) que é usualmente representado por uma área de eco ou seção transversal de radar, mais conhecida em inglês como RCS (*Radar Cross Section*). A área de eco ou RCS é definida como a área interceptando a quantidade de potência que, quando espalhada isotropicamente, produz no receptor uma densidade que é igual à densidade espalhada pelo objeto (alvo) real (BALANIS, 1989). A RCS (σ_{3D}) para um objeto tridimensional é definida em função dos campos elétricos espalhado ($\boldsymbol{E_{esp}}$) e incidente ($\boldsymbol{E_{inc}}$) como:

$$\sigma_{3D} = \lim_{r \to \infty} \left[4\pi . r^2 \frac{|\boldsymbol{E_{esp}}|^2}{|\boldsymbol{E_{inc}}|^2} \right] \tag{6.14}$$

onde *r* é a distância do ponto de observação até o objeto (alvo). A seguir, far-se-á uma breve análise das equações para calcular a RCS (σ_{3D}) teórica de uma placa retangular com condutor elétrico perfeito (com alta condutividade elétrica, $\sigma = 6\ 10^{14}$ S/m). A placa retangular metálica é considerada centrada no eixo de coordenadas (x, y, z), o qual corresponderá à posição central das grades 3D de ambos os métodos FDTD, como mostrado na Figura 6.6. A placa terá uma largura (*a*) na direção *y*, uma largura (*b*) na direção *z* e uma largura (*c*) desprezível (teoricamente nula) na direção *x*. A dedução teórica do campo espalhado nesta placa retangular usa uma aproximação simplificada a partir da óptica física e, portanto, ignorando os efeitos de borda (BALANIS, 1989). A equação que será apresentada é para o modo TM_Z em que a onda incidente propaga-se na direção +*x* com componente de campo elétrico $E_{Z,inc}$ e componente de campo magnético $H_{Y,inc}$, como mostrado na Figura 6.6.

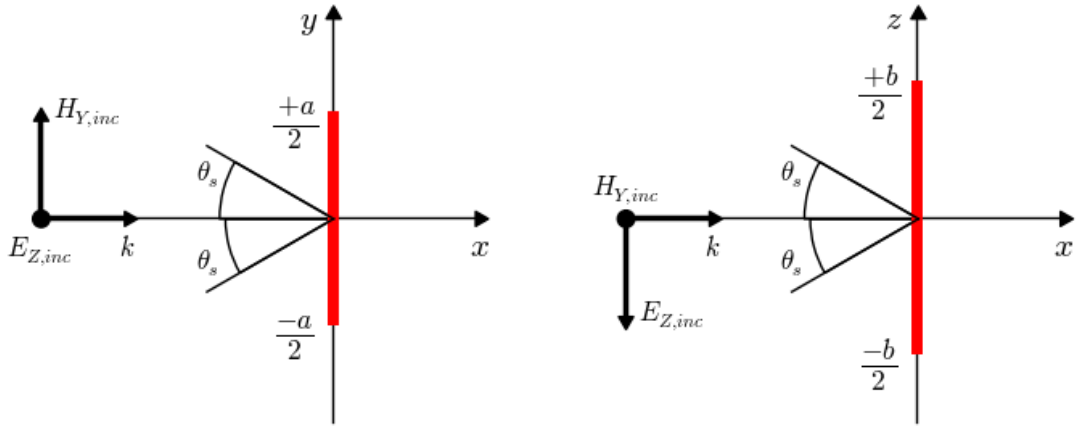


Figura 6.6 - Incidência de onda plana em uma placa retangular metálica.

Assim, para o plano E (plano x-z), que contém o campo $E_{Z,inc}$, a RCS (σ_{3D}) teórica é calculada como:

$$\sigma_{3D} = 4.\pi \left(\frac{a.b}{\lambda}\right)^2 [\cos^2\theta_i(\cos^2\theta_s\cos^2\phi_s + \operatorname{sen}^2\phi_s)].\left[\frac{\operatorname{sen} X}{X}\right]^2.\left[\frac{\operatorname{sen} Y}{Y}\right]^2 \quad (6.15)$$

$$X = \frac{k.a}{2}\operatorname{sen}\theta_s\cos\phi_s \quad (6.16)$$

$$Y = \frac{k.b}{2}(\operatorname{sen}\theta_s\operatorname{sen}\phi_s - \operatorname{sen}\theta_i) \quad (6.17)$$

onde k é o número de onda definido, em função do comprimento de onda físico (λ), como $k = 2\pi/\lambda$, a θ_i é o ângulo de incidência da onda plana em relação ao eixo –x; como a onda plana usada nas simulações é sempre normal ao eixo –x, ter-se-á sempre $\theta_i = 0°$. O ângulo Φ_s é tomado a partir do eixo +z na direção do eixo +y; como o plano de incidência da onda plana está sobre o plano x-y, usar-se-á $\Phi_s = 90°$. O ângulo de espalhamento do campo θ_s, para o modo TM_Z, é tomado a partir do eixo –x ($\theta_s = 0°$ para o valor máximo de $E_{Z,esp}$) em direção aos eixos +z ou –z ($\theta_s = 90°$ para os valores mínimos de $E_{Z,esp}$), como mostrado na Figura 6.6; assim, o ângulo θ_s produz na direção do eixo –z valores de campo $E_{Z,esp}$ equivalentes àqueles da direção +z (como um espelho). Desta forma, é sempre usado: $0° \leq \theta_s \leq 90°$.

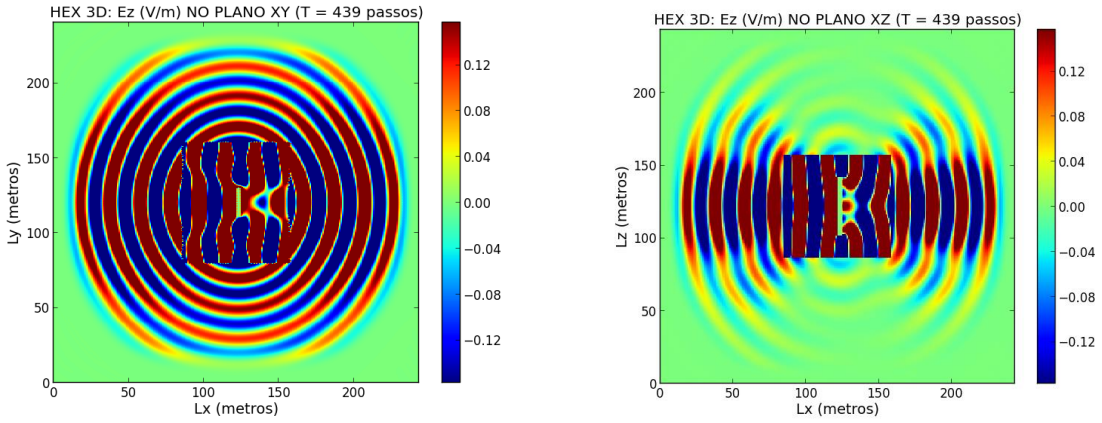
Na comparação entre valores teóricos e numéricos é conveniente transformar os valores de ângulo θ_s para um ângulo θ que tenha a seguinte variação $0° \leq \theta \leq 180°$. Assim, para medir o espalhamento de campo no plano E (plano x-z) tem-se $\theta = 0°$ no

eixo +z, θ = 90° no eixo –x e θ = 180° no eixo –z. Os níveis de espalhamento teórico (RCS (σ_{3D})) no plano x-z (plano E) são calculados usando a equação (6.15) e transformados em valores dBsm (dB / square meter) usando a seguinte equação:

$$\sigma_{3D}(\text{dBsm}) = 10.\log(\sigma_{3D}/\lambda^2) \tag{6.18}$$

Para calcular os níveis de espalhamento teórico (RCS (σ_{3D})) no plano x-y (plano H) a rigor deveria se usar uma equação específica para o modo TE_Z (BALANIS, 1989), mas como o ângulo de incidência (θ_i) é sempre zero é possível usar as mesmas equações (6.15) a (6.18), apenas trocando os valores das larguras *a* e *b* nas equações (6.16) e (6.17). Na comparação entre valores teóricos e numéricos de espalhamento no plano x-y (plano H) é conveniente também definir um ângulo θ, similar àquele usado no plano E (plano x-z) que tenha a mesma variação 0° ≤ θ ≤ 180°. Assim, para medir o espalhamento de campo no plano H (plano x-y) tem-se θ = 0° no eixo +y, θ = 90° no eixo –x e θ = 180° no eixo –y.

Na simulação com o método FDTD com grade de prismas hexagonais usa-se N_X = 281, N_Y = 481, N_Z = 281, Δ*d* = 1 m, *R* = 1,1547, número de Courant S_{CFL} = 1 / √3,3334, número de pontos por comprimento de onda N_λ = 20 e número de passos no tempo *n* = 439. A placa retangular é colocada na posição central da grade com *a* = *ly* = 20 m, *b* = *lz* = 40 m e *c* = *lx* = 3m. Nas Figuras 6.7a e 6.7b é mostrado o campo espalhado produzido pela placa retangular; o nível da onda senoidal é reduzido por 10^{-1} em ambas as figuras para facilitar a visualização.



(a) (b)

Figura 6.7 - Ondas planas incidente e espalhada por placa retangular na grade de prismas hexagonais: (a) plano x-y; (b) plano x-z.

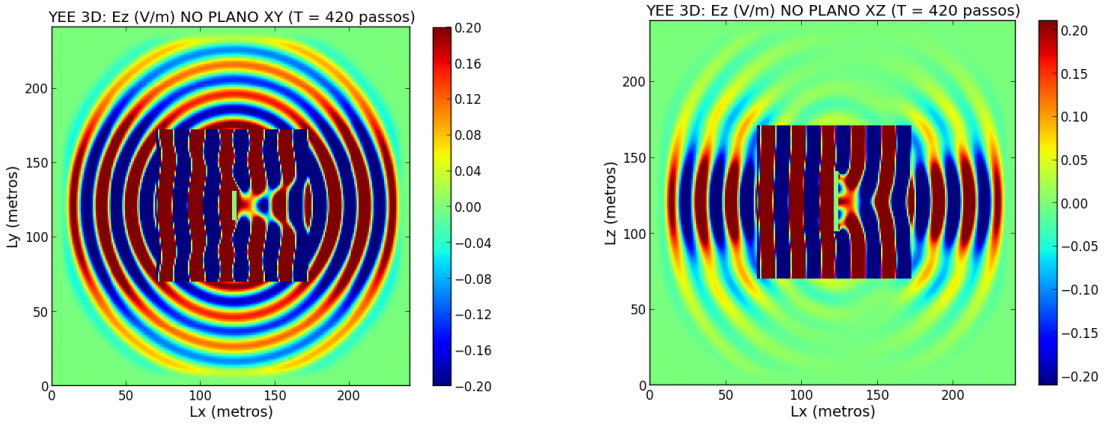
Para medir a RCS numérica são amostrados pontos nos planos x-y e x-z na região de campo espalhado em torno da placa retangular com um raio aproximado de 75 m, usando as seguintes equações:

$$\sigma_{3D,num} = \frac{4\pi.(r[n])^2.\left|E_{Z,esp}[n]\right|^2}{\left|E_{Z,inc}\right|^2} \tag{6.19}$$

$$\sigma_{3D,num}(\text{dBsm}) = 10.\log\left(\sigma_{3D,num}/\lambda^2\right) \tag{6.20}$$

onde o comprimento de onda $\lambda = N_\lambda.\Delta d$ = 20 m, $E_{Z,inc}$ = 0,86 V/m e $E_{Z,esp}[n]$ é medido na posição ou raio $r[n]$ a partir da posição central da grade 3D; para cada $E_{Z,esp}[n]$ é associado um ângulo $\theta[n]$ tal que $0° \leq \theta[n] \leq 180°$, e n são os números de pontos amostrados no meio plano H (+y, -x, -y) ou no meio plano E (+z, -x, -z), como mostrado na Figura 6.6. Assim, são feitas as comparações das RCSs dos planos E e H da placa retangular na grade de prismas hexagonais com aquelas da grade Yee e as soluções analíticas, como será mostrado nas Figuras 6.9 e 6.10.

O mesmo procedimento é aplicado para o método FDTD Yee onde usa-se $N_X = N_Y = N_Z = 241$, $\Delta x = \Delta y = \Delta z$ = 1 m, número de Courant $S_{CFL} = 1 / \sqrt{3}$, número de pontos por comprimento de onda N_λ = 20 e número de passos no tempo n = 420. A placa retangular é colocada na posição central da grade Yee com $a = ly$ = 20 m, $b = lz$ = 40 m e $c = lx$ = 3m. Nas Figuras 6.8a e 6.8b é mostrado o campo espalhado produzido pela placa retangular; o nível da onda senoidal é reduzido por 10^{-1} em ambas as figuras para facilitar a visualização.



(a) (b)

Figura 6.8 - Ondas planas incidente e espalhada por placa retangular na grade Yee: (a) plano x-y; (b) plano x-z.

Para medir a RCS são amostrados pontos no planos x-y e x-z na região de campo espalhado em torno da placa retangular com raio aproximado de 89 m usando as equações (6.19) e (6.20). O valor do comprimento de onda usado é igual àquele da grade de prismas hexagonais, ou seja $\lambda = N_\lambda.\Delta x$ = 20 m e $E_{Z,inc}$ = 0,86 V/m. Os resultados são mostrados nas Figuras (6.9) e (6.10).

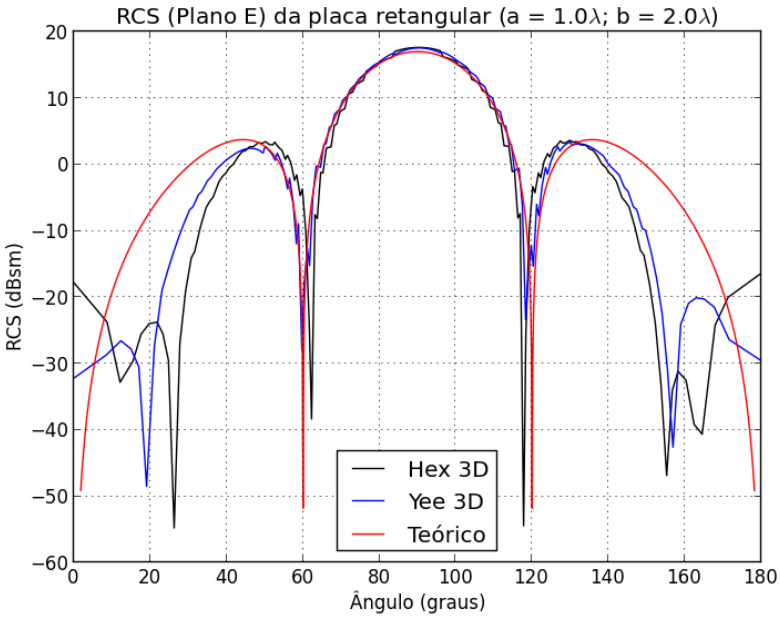


Figura 6.9 - Comparação de RCS teórica e RCSs numéricas de ambos os métodos FDTD no plano E (plano x-z), com *a* = 1,0 λ e *b* = 2,0 λ.

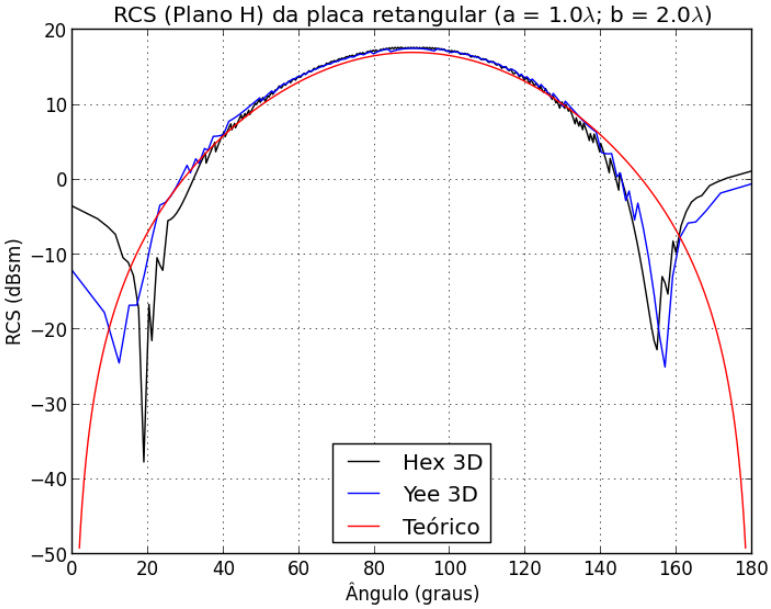


Figura 6.10 - Comparação de RCS teórica e RCSs numéricas de ambos os métodos FDTD no plano H (plano x-y), com *a* = 1,0 λ e *b* = 2,0 λ.

As curvas de RCS nos planos E ou H de ambos os métodos FDTD são muito semelhantes. Observa-se que as curvas de RCS (nos planos E ou H) da grade de prismas hexagonais e da grade Yee ajustam-se de forma muito próxima, na região central de cada gráfico, com a curva de RCS teórica. Nas bordas a RCS téorica, por ser um modelo aproximado, não pode ser comparada com precisão com as RCSs numéricas. Outras simulações são feitas para a grade de prismas hexagonais e a grade Yee, apenas alterando as dimensões da placa retangular para a = 2,0 λ e b = 1,5 λ. Os resultados obtidos nos planos E e H, são mostrados nas Figuras 6.11 e 6.12.

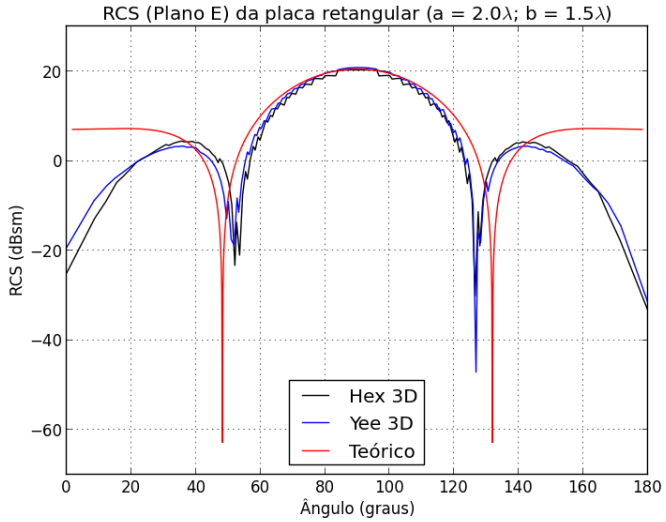


Figura 6.11 - Comparação de RCS teórica e RCSs numéricas de ambos os métodos FDTD no plano E (plano x-z), com a = 2,0 λ e b = 1,5 λ.

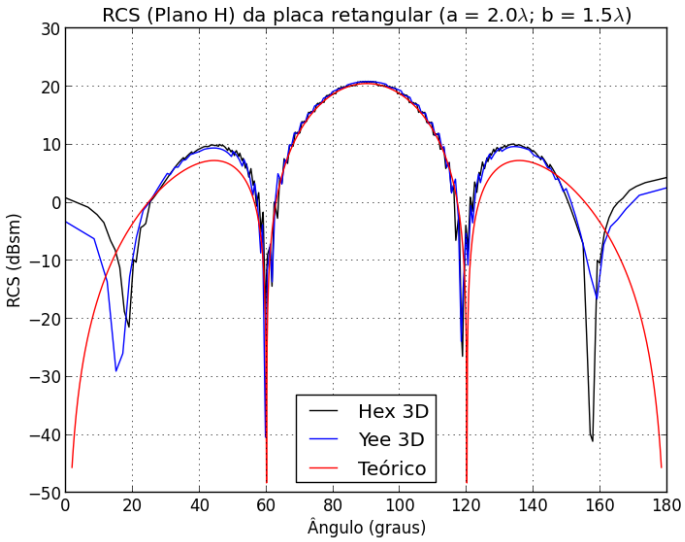


Figura 6.12 - Comparação de RCS teórica e RCSs numéricas de ambos os métodos FDTD no plano H (plano x-y), com a = 2,0 λ e b = 1,5 λ.

Novamente observa-se um comportamento similar nas curvas de RCS de ambos os métodos FDTD, mas o ajuste entre as curvas numéricas com a curva teórica, na região central de cada gráfico, é melhor no plano H que no plano E.

6.5 COMPARAÇÕES DE ESPALHAMENTOS DE CAMPO PRODUZIDOS POR ESFERA METÁLICA

Espalhamento de uma esfera metálica por onda plana é um problema clássico e, nesta seção, os resultados analíticos são usados para comparação com os resultados numéricos gerados pelo método FDTD com grade de prismas hexagonais e o método FDTD Yee. Uma breve análise é feita das equações necessárias para calcular a RCS (σ_{3D}) teórica de uma esfera com condutor elétrico perfeito (com alta condutividade elétrica, $\sigma = 6\ 10^{14}$ S/m); para mais detalhes consultar (BALANIS, 1989) e (JIN, 2010). A esfera metálica com raio *a* é centrada no eixo de coordenadas (x, y, z) que corresponde à posição central das grades 3D de ambos os métodos FDTD, como mostrado na Figura 6.13.

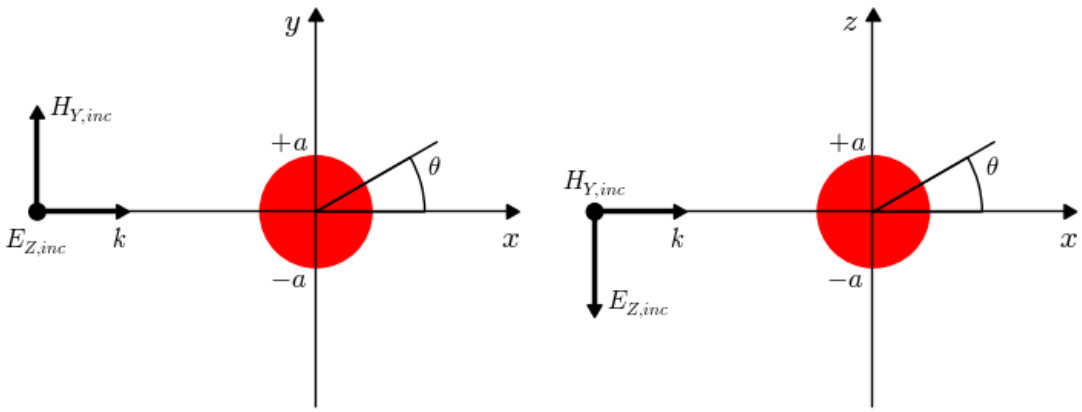


Figura 6.13 - Incidência de onda plana em uma esfera metálica.

É considerado na Figura 6.13 que a esfera metálica é atingida por uma onda plana propagando-se na direção +*x* com componente de campo elétrico $E_{Z,inc}$ e componente de campo magnético $H_{Y,inc}$. A RCS (σ_{3D}) teórica da esfera metálica é definida da mesma maneira que a da placa retangular, usando a equação (6.14) em função dos campos elétricos espalhado ($\boldsymbol{E_{esp}}$) e incidente ($\boldsymbol{E_{inc}}$). Assim, a partir de (BALANIS, 1989), tem-se:

$$\sigma_{3D} = \frac{\lambda^2}{\pi}\left[\cos^2\phi.|A_\theta|^2 + \operatorname{sen}^2\phi.\left|A_\phi\right|^2\right] \quad (6.21)$$

com:

$$|A_\theta|^2 = \left|\sum_{n=1}^{\infty} j^n\left[b_n.\operatorname{sen}\theta.P_n'^1(\cos\theta) - c_n.\frac{P_n^1(\cos\theta)}{\operatorname{sen}\theta}\right]\right|^2 \quad (6.22)$$

$$\left|A_\phi\right|^2 = \left|\sum_{n=1}^{\infty} j^n\left[b_n.\frac{P_n^1(\cos\theta)}{\operatorname{sen}\theta} - c_n.\operatorname{sen}\theta.P_n'^1(\cos\theta)\right]\right|^2 \quad (6.23)$$

onde $j = \sqrt{-1}$ é a unidade imaginária e as variáveis b_n e c_n são definidas como:

$$b_n = -a_n\frac{\hat{J}_n'(k.a)}{\hat{H}_n^{(2)\prime}(k.a)} \quad (6.24)$$

$$c_n = -a_n\frac{\hat{J}_n(k.a)}{\hat{H}_n^{(2)}(k.a)} \quad (6.25)$$

onde a é o raio da esfera. A variável a_n e o número de onda k são definidos como:

$$a_n = j^{-n}\frac{2n+1}{n(n+1)}; \qquad k = \frac{2\pi}{\lambda} \quad (6.26)$$

onde λ é o comprimento de onda físico (real). Nas equações (6.22) e (6.23) P_n^1 é o polinômio de Legendre associado de primeiro tipo de ordem inteira n e grau real um e $P_n'^1$ é a primeira derivada de P_n^1. As funções $\hat{J}_n$ e $\hat{H}_n^{(2)}$ na equação (6.25) são as funções esféricas de Bessel de 1º tipo de ordem inteira n e de Hankel de 2º tipo de ordem inteira n. Estas funções ($\hat{J}_n$ e $\hat{H}_n^{(2)}$) são definidas a partir da função cilíndrica regular de Bessel de 1º tipo (J_p) de ordem real p e da função cilíndrica regular de Hankel de 2º tipo ($H_p^{(2)}$) de ordem real p, respectivamente, como:

$$\hat{J}_n(k.a) = \sqrt{\frac{\pi k.a}{2}}J_{n+\frac{1}{2}}(k.a) \quad (6.27)$$

$$\hat{H}_n^{(2)}(k.a) = \sqrt{\frac{\pi k.a}{2}}H_{n+\frac{1}{2}}^{(2)}(k.a) \quad (6.28)$$

Na equação (6.24) as derivadas $\hat{J}_n'$ e $\hat{H}_n^{(2)\prime}$ podem ser obtidas das equações (6.27) e (6.28), respectivamente, como:

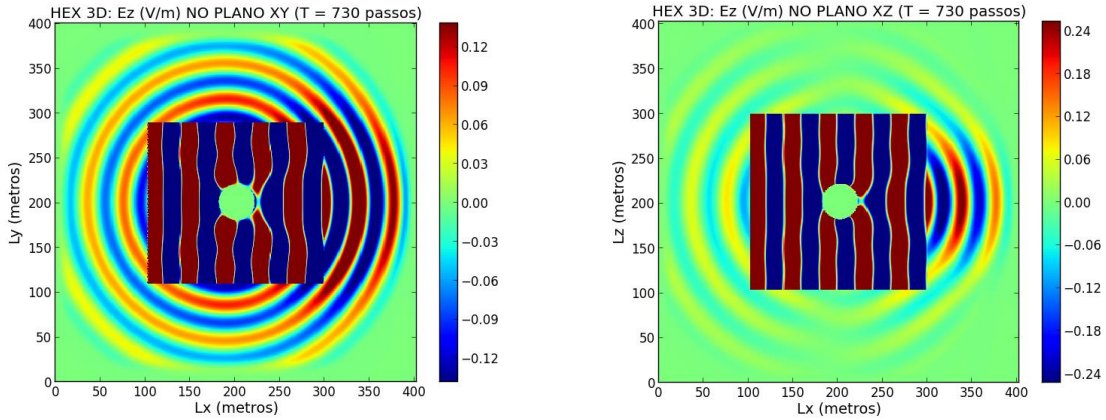
$$\hat{J}_n'(k.a) = \sqrt{\frac{\pi k}{8a}} J_{n+\frac{1}{2}}(k.a) + \sqrt{\frac{\pi k.a}{2}}.k.J'_{n+\frac{1}{2}}(k.a) \tag{6.29}$$

$$\hat{H}_n^{(2)\prime}(k.a) = \sqrt{\frac{\pi k}{8a}} H^{(2)}_{n+\frac{1}{2}}(k.a) + \sqrt{\frac{\pi k.a}{2}}.k.H^{(2)\prime}_{n+\frac{1}{2}}(k.a) \tag{6.30}$$

O ângulo Φ na equação (6.21) é definido a partir do eixo +z na direção do eixo +y no plano y-z. Para simulações no plano E (plano x-z) é usado ângulo Φ = 0° (eixo +z) e com ângulo de espalhamento (θ) variando a partir do eixo +x (θ = 0°), atravessando o eixo +z (θ = 90°) e terminando no eixo -x (θ = 180°), como mostrado na Figura 6.13. É suficiente calcular o espalhamento em um meio plano (+x, +z, -x) devido à simetria do espalhamento da esfera no plano x-z, ou seja, para: 0° ≤ θ ≤ 180°. Para simulações no plano H (plano x-y) é usado ângulo Φ = 90° (eixo +y) e com ângulo de espalhamento (θ) variando a partir do eixo +x (θ = 0°), atravessando o eixo +y (θ = 90°) e terminando no eixo -x (θ = 180°), como mostrado na Figura 6.13. Assim, o espalhamento é também calculado apenas em um meio plano (+x, +y, -x) devido à simetria do espalhamento da esfera no plano x-y. Os níveis de espalhamento teórico (RCS (σ_{3D})) da esfera metálica para os planos E e H devem ser transformados em valores dBsm usando a equação (6.18), repetida aqui por conveniência:

$$\sigma_{3D}(\text{dBsm}) = 10.\log(\sigma_{3D}/\lambda^2) \tag{6.31}$$

Na simulação com o método FDTD com grade de prismas hexagonais usa-se N_X = 465, N_Y = 801, N_Z = 465, Δd = 1 m, R = 1,1547, número de Courant S_{CFL} = 1 / √3,3334, número de pontos por comprimento de onda N_λ = 40 e número de passos no tempo n = 730. A esfera metálica é colocada na posição central da grade com raio a = 0,5 λ = 0,5(N_λ Δd) = 20 m. Nas Figuras 6.14a e 6.14b é mostrado o campo espalhado produzido pela esfera metálica; o nível da onda senoidal é reduzido por 10^{-1} em ambas as figuras para facilitar a visualização.



(a) (b)

Figura 6.14 - Ondas planas incidente e espalhada por esfera metálica na grade de prismas hexagonais: (a) plano x-y; (b) plano x-z.

Para medir as RCSs numéricas são amostrados pontos nos planos x-y e x-z na região de campo espalhado em torno da esfera, com raios $R_d \approx 140$ m (plano x-y) e $R_d \approx 161$ m (plano x-z) na região de campo distante ($R_d \geq 2L^2 / \lambda$ onde L é a maior dimensão da esfera, ou seja, $L = 2a = 1{,}0\ \lambda = 40$ m), usando as equações (6.19) e (6.20), repetidas aqui por conveniência:

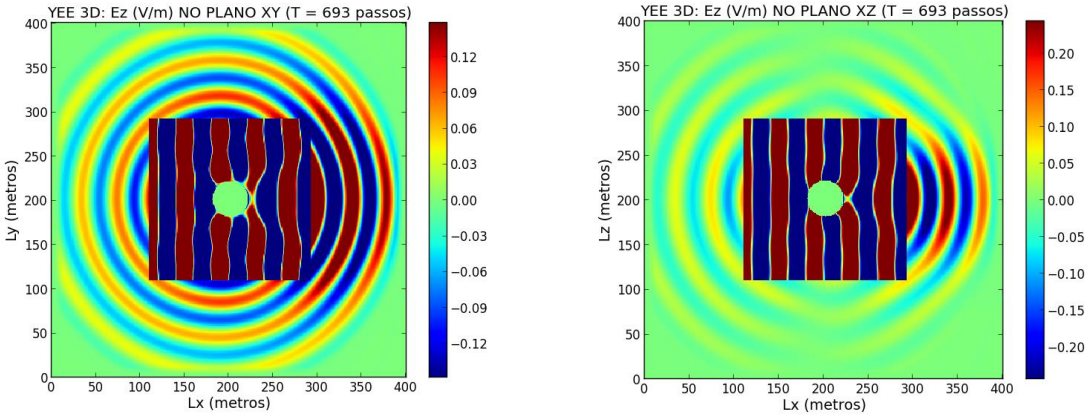
$$\sigma_{3D,num} = \frac{4\pi.(r[n])^2.\left|E_{Z,esp}[n]\right|^2}{\left|E_{Z,inc}\right|^2} \tag{6.32}$$

$$\sigma_{3D,num}(\text{dBsm}) = 10.\log\left(\sigma_{3D,num}/\lambda^2\right) \tag{6.33}$$

onde o comprimento de onda $\lambda = N_\lambda.\Delta d = 40$ m, $E_{Z,inc} = 1{,}0$ V/m e $E_{Z,esp}[n]$ é medido na posição ou raio $r[n]$ a partir da posição central da grade 3D; para cada $E_{Z,esp}[n]$ é associado um ângulo $\theta[n]$ tal que $0° \leq \theta[n] \leq 180°$, e n são os números de pontos amostrados no meio plano H (+x, +y, -x) ou no meio plano E (+x, +z, -x), como mostrado na Figura 6.13. Assim, são feitas as comparações das RCSs dos planos E e H da esfera metálica na grade de prismas hexagonais com aquelas da grade Yee e as soluções analíticas, como será mostrado nas Figuras 6.16 e 6.17.

O mesmo procedimento é aplicado para o método FDTD Yee onde usa-se $N_X = N_Y = N_Z = 401$, $\Delta x = \Delta y = \Delta z = 1$ m, número de Courant $S_{CFL} = 1 / \sqrt{3}$, número de pontos por comprimento de onda $N_\lambda = 40$ e número de passos no tempo $n = 693$. A esfera metálica é colocada na posição central da grade Yee com raio $a = 0{,}5\ \lambda = 0{,}5(N_\lambda$

Δx) = 20 m. Nas Figuras 6.15a e 6.15b é mostrado o campo espalhado produzido pela esfera metálica; o nível da onda senoidal é reduzido por 10^{-1} em ambas as figuras para facilitar a visualização.



(a) (b)

Figura 6.15 - Ondas planas incidente e espalhada por esfera metálica na grade Yee: (a) plano x-y; (b) plano x-z.

Para medir a RCS na grade Yee são amostrados pontos no planos x-y e x-z na região de campo espalhado em torno da esfera metálica, com raio $R_d \approx 162$ m (nos planos x-y ou x-z) na região de campo distante de forma similar à grade de primas hexagonais, usando as equações (6.32) e (6.33). O valor do comprimento de onda usado é igual àquele da grade de prismas hexagonais, ou seja $\lambda = N_\lambda.\Delta x = 40$ m e $E_{Z,inc} = 1{,}0$ V/m. Os resultados são mostrados nas Figuras (6.16) e (6.17).

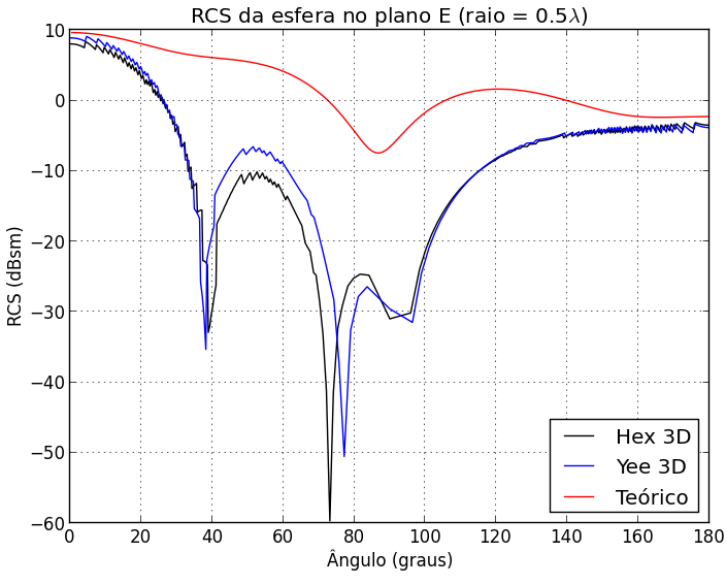


Figura 6.16 - Comparação de RCS teórica e RCSs numéricas de ambos os métodos FDTD no plano E (plano x-z), com o raio da esfera a = 0,5 λ.

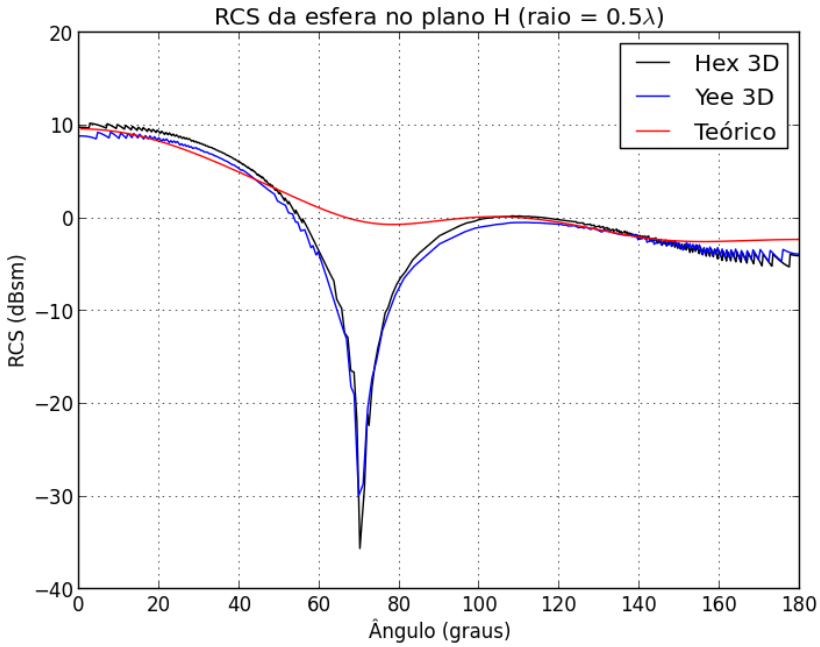


Figura 6.17 - Comparação de RCS teórica e RCSs numéricas de ambos os métodos FDTD no plano H (plano x-y), com o raio da esfera a = 0,5 λ.

Observa-se nas Figuras 6.16 e 6.17 que as curvas numéricas de RCS da esfera nos planos E e H são muito similares em ambos os métodos FDTD. No plano E as curvas numéricas de RCS apresentam dois picos negativos não existentes na curva teórica de RCS. No plano H as curvas numéricas de RCS tem comportamento mais próximo ao da curva teórica, com exceção do pico negativo. No entanto, em ambos os planos (E e H) os níveis de RCS, para os ângulos de 0° e 180°, das curvas numéricas e teóricas são muito próximos.

Outras simulações são feitas, para a grade de prismas hexagonais e a grade Yee, apenas alterando o número de pontos por comprimento de onda para N_λ = 20, de forma que o raio da esfera se torna a = 1,0λ. Os resultados obtidos nos planos E e H, são mostrados nas Figuras 6.18 e 6.19, respectivamente. Observa-se que as curvas numéricas de RCS no plano H são muito similares, embora com três picos negativos não existentes na curva analítica. No plano E as curvas numéricas de RCS são apenas semelhantes nos ângulos iniciais e finais; e ambas as curvas numéricas apresentam um pico negativo muito próximo do pico negativo da curva analítica. No entanto, em ambos os planos (E e H) os níveis de RCS, para os ângulos de 0° e 180°, das curvas numéricas e teóricas são muito próximos. Estas diferenças não-desprezíveis nos formatos das curvas numéricas em relação às curvas teóricas, nas

Figuras 6.16 a 6.19, são devidas ao processo de discretização, ou seja, devido à aproximação de contornos curvos por passos espaciais em forma de escada.

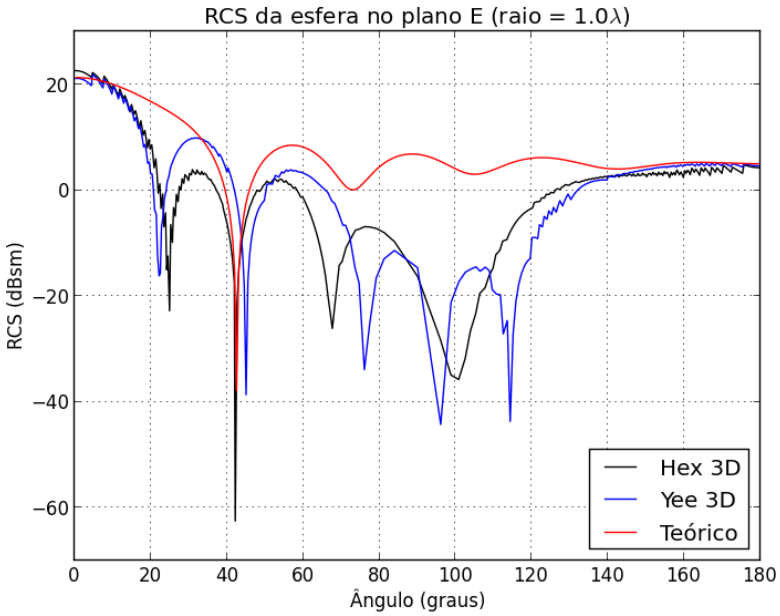


Figura 6.18 - Comparação de RCS teórica e RCSs numéricas de ambos os métodos FDTD no plano E (plano x-z), com o raio da esfera *a* = 1,0 λ.

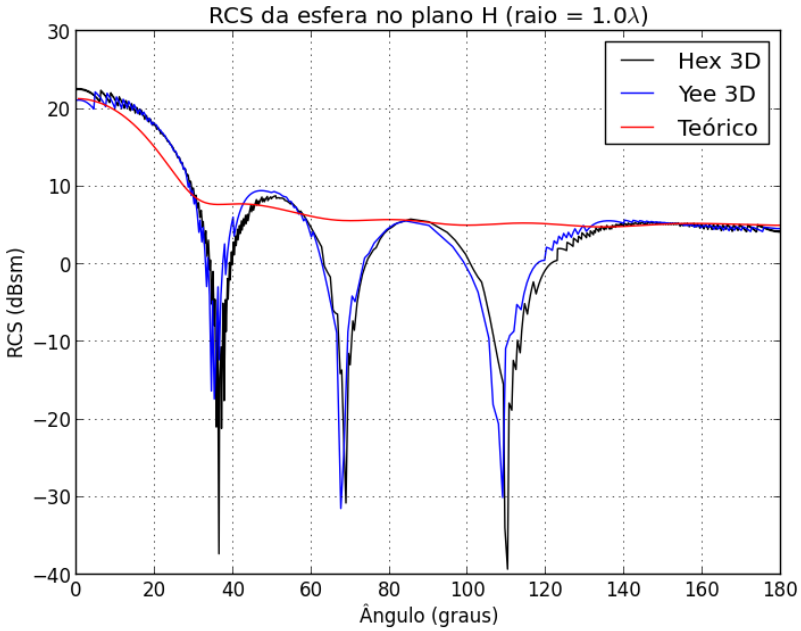


Figura 6.19 - Comparação de RCS teórica e RCSs numéricas de ambos os métodos FDTD no plano H (plano x-y), com o raio da esfera *a* = 1,0 λ.

7 COMPENSAÇÃO DE DISPERSÃO NUMÉRICA PARA O MÉTODO FDTD COM GRADE DE PRISMAS HEXAGONAIS

O método FDTD com grade de prismas hexagonais tem uma importante vantagem em relação ao método FDTD Yee, isto é, uma compensação mais efetiva da dispersão numérica (JOAQUIM; SCHEER, 2016), como será analisado e provado neste capítulo. Usualmente, em edificações, cada andar tem dimensões L_X e L_Y (no plano horizontal ou plano x-y) que são consideravelmente maiores que a altura do andar (dimensão L_Z). É suposto, como mostrado na Figura 7.1, que uma fonte senoidal de sinal (antena) é colocada na posição central deste andar com dimensões, por simplicidade, iguais no plano horizontal ($L_X = L_Y$) e que $L_X = L_Y \geq 10.L_Z$. Assim, a anisotropia numérica para uma distância (r) da antena, tal que $r \geq 5.L_Z$, será limitada por um ângulo $\theta = \tan^{-1}[(L_Z/2) / (5.L_Z)]$ tal que $0° \leq \theta \leq 5{,}72°$ nos planos x-z ou y-z, como mostrado na Figura 7.1.

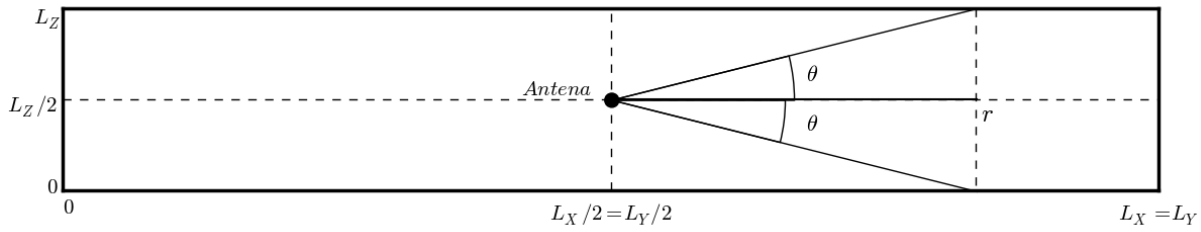


Figura 7.1 - Desenho da seção transversal de um andar de edifício mostrando uma antena em seu centro geométrico.

No método FDTD com grade de prismas hexagonais, o uso dos parâmetros $N_\lambda = 10$, $R = 1{,}1547$, $S_{CFL} = 1/\sqrt{3{,}33333209}$ e qualquer $\Delta d = \Delta$ produzirá anisotropia numérica igual a 0,0247% para este ângulo $\theta = 5{,}72°$. Este valor de anisotropia numérica é em torno de 9 vezes maior que a anisotropia numérica máxima no plano x-y, que é igual a 0,002761%. Dessa forma, a anisotropia numérica máxima no plano x-y pode ser desconsiderada na compensação da dispersão numérica para o método FDTD com grade de prismas hexagonais. Contudo, para o método FDTD Yee uma simplificação semelhante na compensação da dispersão numérica não pode ser feita. No método FDTD Yee, o uso dos parâmetros $N_\lambda = 10$, $S_{CFL} = 1/\sqrt{3}$ e qualquer $\Delta x = \Delta y = \Delta z = \Delta$ produzirá anisotropia numérica igual a 0,0331% para este ângulo $\theta = 5{,}72°$ nos planos x-z ou y-z, mas produz uma anisotropia numérica máxima igual a 0,8391%

no plano x-y que não pode ser ignorada. Consequentemente, neste exemplo, para que o método FDTD Yee produza o mesmo valor de anisotrópica numérica que o método FDTD com grade de prismas hexagonais, isto é, anisotropia numérica igual a 0,0247%; ele deve usar parâmetro $N_\lambda \approx 58$ calculado pela equação (4.75), ou seja, a grade Yee deve ser em torno de 73 vezes mais densa que a grade de prismas hexagonais (ver cálculo de densidade de malha mais adiante neste texto). Logicamente esta estimativa é muito aproximada, mas indica uma promissora possibilidade de uso de malhas menos densas e maior precisão nas simulações para o método FDTD com grade de prismas hexagonais em relação ao método FDTD Yee.

Um algoritmo simples e usual, que reduz a dispersão numérica no método FDTD Yee (TAFLOVE; HAGNESS, 2005), consiste em simplesmente usar uma velocidade numérica média (v_{med}) definida, em função das velocidades numéricas máxima e mínima, como:

$$v_{med} = \frac{\min[c^*(\theta)] + \max[c^*(\theta)]}{2} = \frac{c^*(0°) + c^*(45°)}{2} \tag{7.1}$$

Consequentemente, a permissividade elétrica (ε) e a permeabilidade magnética (μ) têm uma redução dada por:

$$\varepsilon' = \left(\frac{v_{med}}{c}\right)\varepsilon; \qquad \mu' = \left(\frac{v_{med}}{c}\right)\mu; \quad \text{com: } v_{med} \leq c \tag{7.2}$$

onde *c* é a velocidade física (real) da onda eletromagnética. Assim, reduzindo ambos os parâmetros ε e μ é obtida uma nova velocidade de fase (*c*'), com valor maior que a velocidade de fase (*c*), definida como:

$$c' = \frac{1}{\sqrt{\mu'.\varepsilon'}} = \left(\frac{c}{v_{med}}\right)\frac{1}{\sqrt{\mu.\varepsilon}} \tag{7.3}$$

No entanto, a impedância (*Z*) no meio não se altera, tal que:

$$Z = \sqrt{\frac{\mu}{\varepsilon}} = \sqrt{\frac{\mu'}{\varepsilon'}} \tag{7.4}$$

Para a grade de prismas hexagonais, um algoritmo similar pode ser aplicado. O ângulo θ, como mostrado na Figura 7.1, é calculado como:

$$\theta = \tan^{-1}\left(\frac{L_Z}{2.r}\right) \tag{7.5}$$

onde L_Z é a altura do andar e a distância (r) a partir da antena é definida como:

$$r = \sqrt{x^2 + y^2} \tag{7.6}$$

Na equação (7.6) as coordenadas cartesianas x e y são definidas, a partir da posição central da grade no plano x-y, como $x = i.\Delta x$ e $y = j.\Delta y$, respectivamente. Assim, a velocidade média normalizada em função do ângulo θ é:

$$\frac{v_{med}(\theta)}{c} = \frac{c^*(0°)/c + c^*(\theta)/c}{2} \tag{7.7}$$

A velocidade média normalizada ($v_{med}(\theta) / c$) é aplicada na localização r ou (x, y) e qualquer z produzindo os parâmetros ε' e μ' a partir da equação (7.2). É observado que quanto maior a distância (r), menor é o ângulo θ calculado da equação (7.5) e menor a anisotropia numérica calculada, usando a análise de Fourier, para este ângulo θ; consequentemente, a redução produzida pela equação (7.2) é cada vez mais precisa. A equação (7.7) pode ser usada enquanto a anisotropia numérica para o ângulo θ é maior ou igual à anisotropia numérica máxima no plano x-y. Dessa forma, para parâmetro N_λ = 10, as dimensões aproximadas máximas no plano x-y, que podem ser usadas com este simples algoritmo, são dadas por $L_X = L_Y \leq 30.L_Z$, isto é, um ângulo $\theta \geq 1{,}88°$.

7.1 MEDIDAS DE DISPERSÃO NUMÉRICA PARA AMBOS OS MÉTODOS FDTD

Para que se possa medir a dispersão numérica no método FDTD com prismas hexagonais ou método FDTD Yee é necessário comparar as ondas senoidais numéricas com uma onda senoidal analítica. A partir da função senoidal discreta descrita pela equação (3.56), que é usada como fonte de excitação em ambos os métodos FDTD, é possível encontrar uma função analítica equivalente definida como:

$$f_s(t,r) = A_{s0}\left[1 - \exp\left(\frac{-t + r/c}{\varsigma_0}\right)\right].\mathrm{sen}(2\pi f.t - k.r + \beta_0) \tag{7.8}$$

onde A_{s0} corresponde a uma amplitude inicial e β_0 a uma fase ou ângulo inicial. Na equação (7.8) o tempo (t), a frequência (f), o número de onda (k) e a constante de tempo (ς_0) são relacionados com os termos discretos, usados nas equações (3.55) e (3.56), da seguinte forma:

$$t = n.\Delta t; \quad f = \frac{c}{\lambda}; \quad k = \frac{2\pi}{\lambda}; \quad \varsigma_0 = \frac{\alpha_T}{f} \tag{7.9}$$

onde o tamanho do passo de tempo é $\Delta t = S_{CFL}.\Delta/c$ e o comprimento de onda é $\lambda = N_\lambda.\Delta$, com $\Delta = \Delta d$ para grade de prismas hexagonais e $\Delta = \Delta x$ para grade Yee; n é o número de passos de tempo, α_T é o coeficiente de atenuação e os termos N_λ, S_{CFL} e c são definidos como na seção 3.3. A função analítica para campo distante (JOAQUIM; SCHEER, 2016), a ser usada na comparação com as ondas senoidais numéricas, é obtida como a primeira derivada, em relação à distância (r), da função dada pela equação (7.8) e reduzida por (1/r), tal que:

$$f_s'(r) = \frac{A_{s0}}{r}\left\{\frac{-1}{c.\varsigma_0}\exp\left(\frac{-t + r/c}{\varsigma_0}\right).\mathrm{sen}(2\pi f.t - k.r + \beta_0) - k\left[1 - \exp\left(\frac{-t + r/c}{\varsigma_0}\right)\right].cos(2\pi f.t - k.r + \beta_0)\right\} \tag{7.10}$$

Assim, é suficiente usar na equação (7.10) os valores obtidos da equação (7.9). A distância r é a variável em função da qual a função analítica é plotada; o valor máximo da variável r é escolhido para cada simulação em função da máxima distância percorrida pelas ondas numéricas. A amplitude inicial A_{s0} e a fase inicial β_0 são escolhidas para cada simulação de forma a ajustar a onda analítica com as ondas numéricas e permitir a comparação de propagação e, consequentemente, o cálculo das dispersões numéricas.

A dispersão numérica, que foi definida na equação (4.77), é repetida aqui por conveniência, numa forma mais simples, como:

$$\Delta v_{dispersão}(\%) = \left(\frac{c^*}{c} - 1\right).100 \tag{7.11}$$

onde as velocidades numérica (c^*) e analítica (c) são definidas para um certo tempo $t = n.\Delta t$, respectivamente, como:

$$c^* = \frac{r^*}{t} \tag{7.12a}$$

$$c = \frac{r}{t} \tag{7.12b}$$

com r^* e r sendo as distâncias percorridas pelas ondas numérica e analítica, respectivamente. Aplicando as equações (7.12a) e (7.12b) na equação (7.11) é obtido:

$$\Delta v_{dispersão}(\%) = \left(\frac{r^*}{r} - 1\right).100 \tag{7.13}$$

A equação (7.13) pode ser usada diretamente para calcular as dispersões numéricas nas direções desejadas das grades 3D de ambos os métodos FDTD. Para a grade de prismas hexagonais são escolhidas três direções no plano x-y; a direção *x* (0°) corresponde à velocidade numérica mínima e as direções *y* (90°) e *d* (30°) correspondem à velocidade numérica máxima (ver Figura 4.2b). Para a grade Yee são escolhidas três direções no plano x-y; as direções *x* (0°) e *y* (90°) correspondem à velocidade numérica mínima e a direção *d* (45°) corresponde à velocidade numérica máxima (ver Figura 4.2a). Na implementação prática do algoritmo, que calcula a dispersão numérica, um ponto de nulo inicial (x_0) é escolhido da onda numérica na direção *x* (para a grade de prismas hexagonais ou grade Yee). Escolhe-se uma fase inicial (β_0) arbitrária para a função analítica da equação (7.10) e o algoritmo automaticamente calcula a fase inicial (β_0) exata que ajusta a fase da onda senoidal analítica com a onda numérica na direção *x* neste especifico ponto de nulo inicial (x_0). Assim, é produzida dispersão numérica zero para esse ponto de nulo inicial (x_0). A partir desta referência inicial de dispersão numérica zero é possível calcular as dispersões numéricas para os demais pontos de nulo das ondas numéricas nas três direções escolhidas no plano x-y em cada método FDTD. A amplitude inicial (A_{s0}) da onda analítica é também escolhida para ajustar-se com as amplitudes das ondas numéricas, embora isso não seja obrigatório. É possível usar também uma equação alternativa para a equação (7.13) que subtrai as distâncias destes pontos de nulo

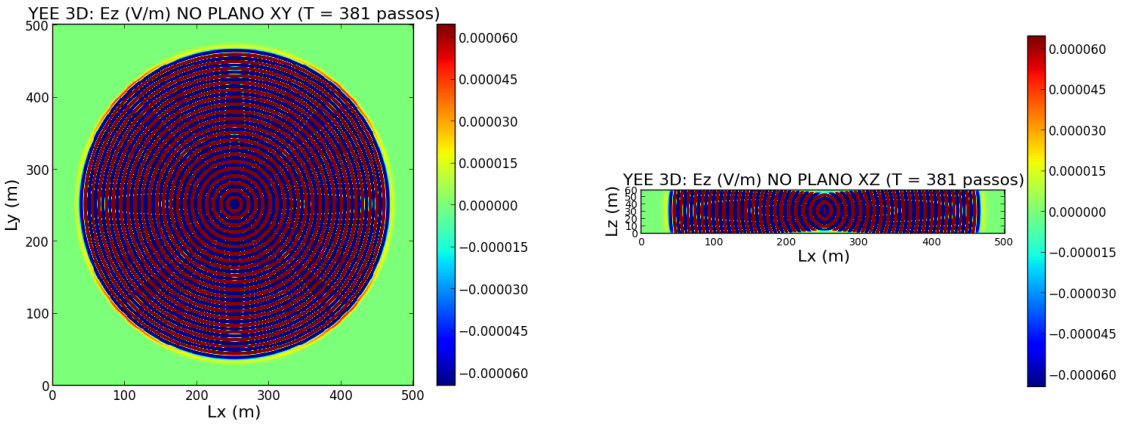
iniciais (r_0^* e r_0) em todas as direções em que a dispersão numérica é calculada, tal que:

$$\Delta v_{dispersão}(\%) = \left(\frac{r^* - r_0^*}{r - r_0} - 1\right).100 \tag{7.14}$$

As curvas de dispersão obtidas com a equação (7.13) serão denominadas de modo 1, e aquelas obtidas com a equação (7.14) de modo 2.

7.1.1 Medidas de dispersão numérica para o método FDTD Yee

Inicialmente é feita uma medida de dispersão numérica no método FDTD Yee sem compensação de dispersão, isto é, (c / v_{med}) = 1. É usado $N_X = N_Y = 501$, $N_Z = 61$, $\Delta x = \Delta y = \Delta z = 1$ m, número de pontos por comprimento de onda $N_\lambda = 10$, número de Courant $S_{CFL} = 1 / \sqrt{3}$, espessuras PMLx = PMLz = PMLy = 20 m e número de passos no tempo $n = 381$. As Figuras 7.2a e 7.2b mostram a propagação da onda senoidal nos planos x-y e x-z, respectivamente. O nível é reduzido em 10^{-4} para facilitar a visualização das ondas.



(a) (b)

Figura 7.2 - Propagação de onda senoidal na grade Yee: (a) plano x-y; (b) plano x-z.

Na Figura 7.3 são mostradas as curvas, a partir do centro do plano x-y (Figura 7.2a), nas direções positivas *x* (0°), *y* (90°) e *d* (45°), incluída a curva teórica com valores iniciais A_{s0} = 0,14 V e β_0 = 156,9859° que ajusta a fase da onda analítica com o ponto de nulo inicial x_0 = 21,8311 m.

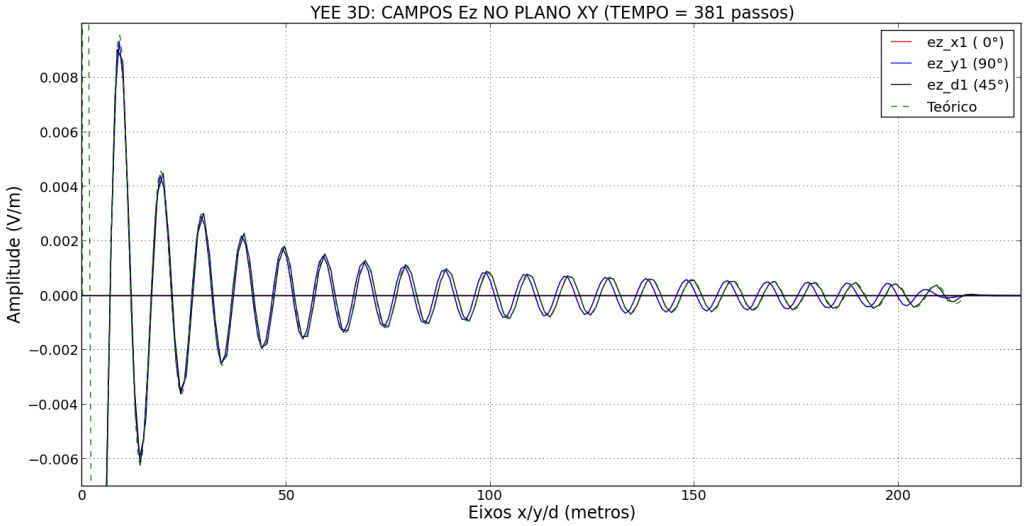


Figura 7.3 - Ondas numéricas nas direções *X* (0°), *y* (90°) e *d* (45°) e onda analítica.

Uma ampliação do final destas curvas é feita na Figura 7.4, sendo fácil observar a dispersão numérica presente nas ondas numéricas em relação á onda analítica.

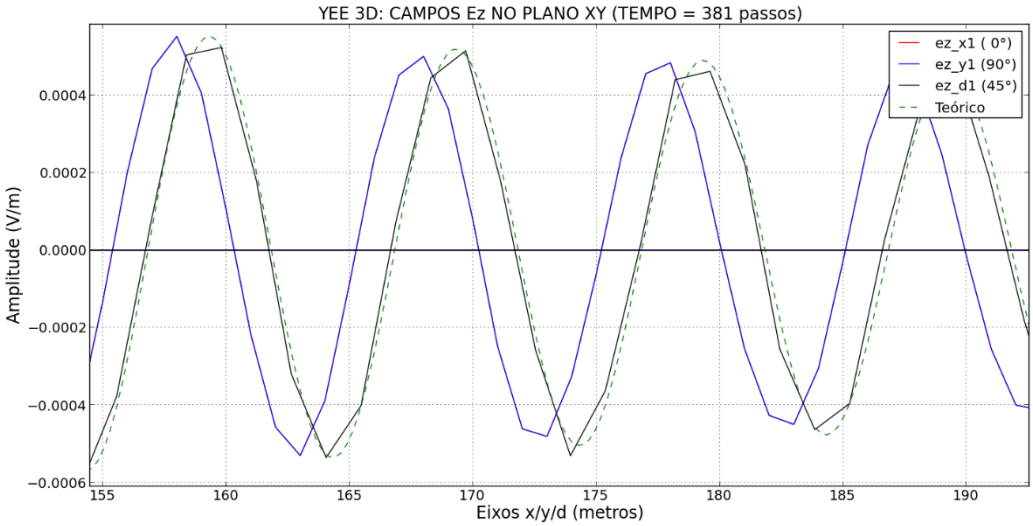
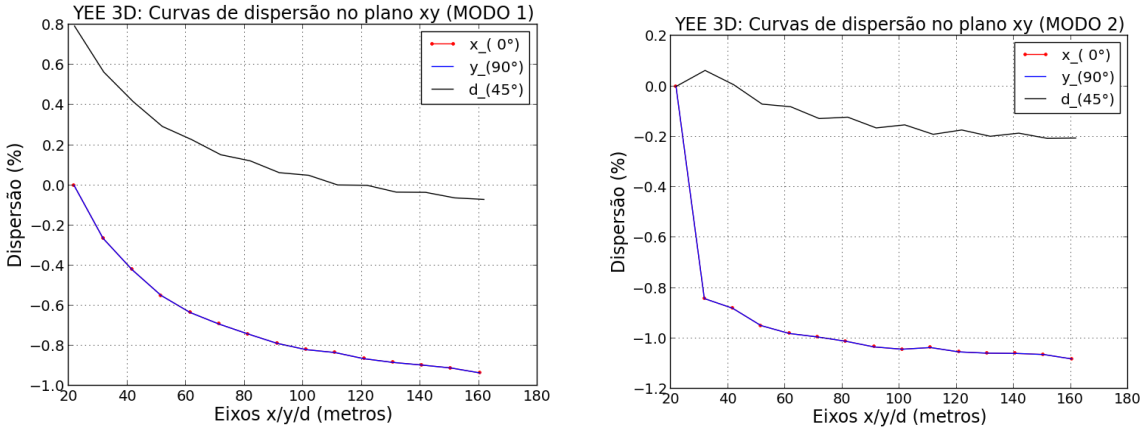


Figura 7.4 - Ampliação da parte final da Figura 7.3 que mostra a dispersão numérica das ondas numéricas em relação à onda analítica no plano x-y da grade Yee.

As curvas de dispersão para as direções *x* (0°), *y* (90°) e *d* (45°) são mostradas na Figura 7.5. As curvas de dispersão nas direções *x* (0°) e *y* (90°) são idênticas, mas a curva de dispersão na direção *d* (45°) está separada das outras duas curvas por um certo valor que é proporcional à anisotropia numérica máxima no plano x-y para a grade Yee. Nestas curvas as medidas de dispersão são mais precisas à medida que

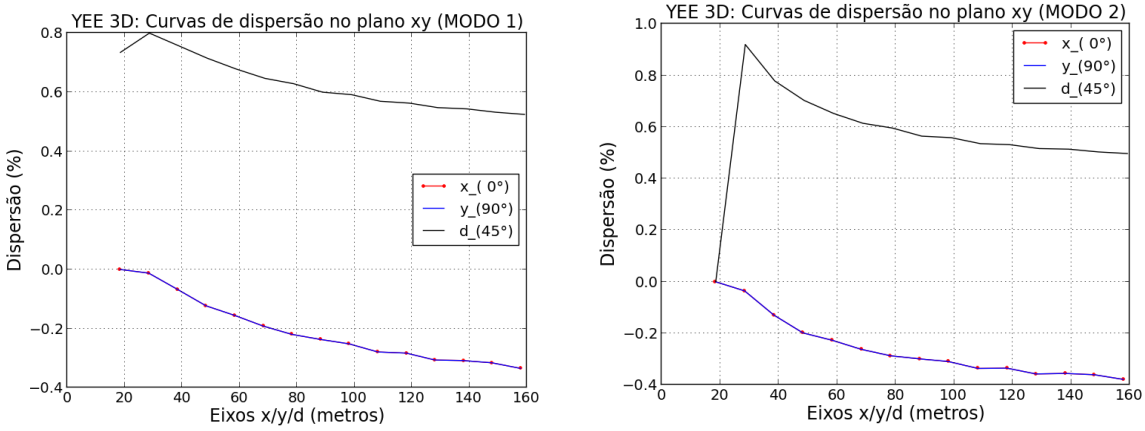
as distâncias das ondas numéricas (r^*) e da onda analítica (r) aumentam. A dispersão numérica medida (maior valor em módulo) no final das curvas (modo 2) nas direções x (0°) ou y (90°) é -1,081279% que é, em módulo, apenas 2,36% menor que o valor teórico (-1,107391%) da Tabela 4.3.



(a) (b)

Figura 7.5 - Curvas de dispersão (sem compensação) no plano x-y para a grade Yee: (a) modo 1; (b) modo 2.

Uma segunda simulação é feita usando uma compensação de dispersão simples, que é definida pela equação (7.1), com fator de ajuste (c / v_{med}) = 1,00697309916 obtido da análise de Fourier. Os demais parâmetros são iguais ao da simulação anterior, com exceção do número de Courant que é reduzido para S_{CFL} = 1 / √3,1 a fim de garantir a estabilidade da simulação, e a fase inicial da onda analítica que é alterada para β_0 = 33,8131° de forma a ajustar a fase da onda analítica com o ponto de nulo inicial x_0 = 18,4097. As curvas de dispersão são mostradas na Figura 7.6



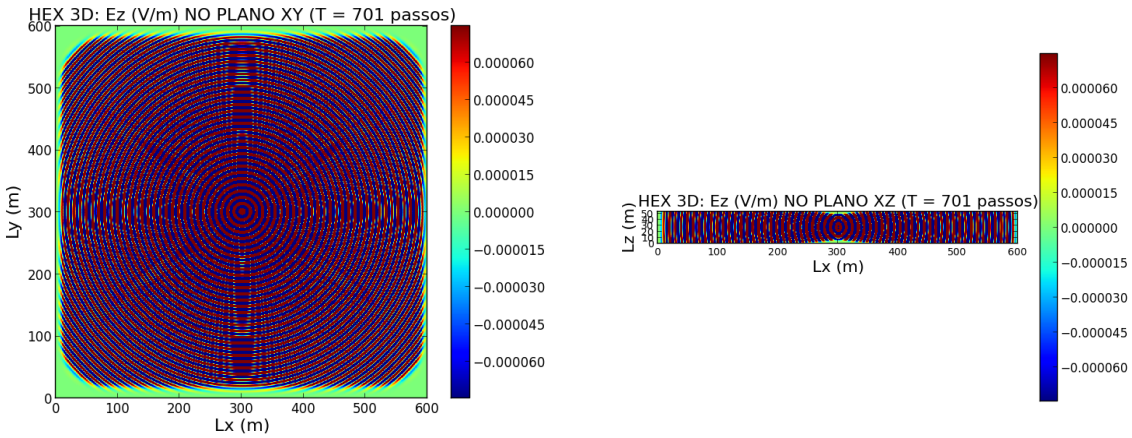
(a) (b)

Figura 7.6 - Curvas de dispersão (com compensação simples) no plano x-y para a grade Yee: (a) modo 1; (b) modo 2.

Observa-se na Figura 7.6b que essa simples compensação de dispersão reduz as dispersões no final das curvas (modo 2) nas direções *x* (0°) ou *y* (90°) para -0,378882%, ou seja, por um fator de aproximadamente 2,85. No entanto, não compensa a anisotropia numérica presente entre a direção *d* (45°) e as direções *x* (0°) ou *y* (90°) (TAFLOVE; HAGNESS, 2005).

7.1.2 Medidas de dispersão numérica para o método FDTD com grade de prismas hexagonais

Inicialmente é feita uma medida de dispersão numérica no método FDTD com grade de prismas hexagonais sem compensação de dispersão, isto é, $(c / v_{med}) = 1$. É usado $N_X = 693$, $N_Y = 1201$, $N_Z = 61$, $\Delta d = 1$ m, $R = 1{,}1547$, número de pontos por comprimento de onda $N_\lambda = 10$, número de Courant $S_{CFL} = 1 / \sqrt{3{,}3334}$, espessuras PMLx = PMLz ≈ 22,5 m e PMLy = 40 m, e número de passos no tempo $n = 701$. As Figuras 7.7a e 7.7b mostram a propagação da onda senoidal nos planos x-y e x-z, respectivamente. O nível é reduzido em 10^{-4} para facilitar a visualização das ondas.



(a) (b)

Figura 7.7 - Propagação de onda na grade de prismas hexagonais: (a) plano x-y; (b) plano x-z.

Na Figura 7.8 são mostradas as curvas, a partir do centro do plano x-y (Figura 7.7a), nas direções positivas *x* (0°), *y* (90°) e *d* (30°), incluída a curva teórica com valores iniciais $A_{s0} = 0{,}11$ e $\beta_0 = 159{,}5161°$ que ajusta a fase da onda analítica com o ponto de nulo inicial $x_0 = 25{,}8806$ m.

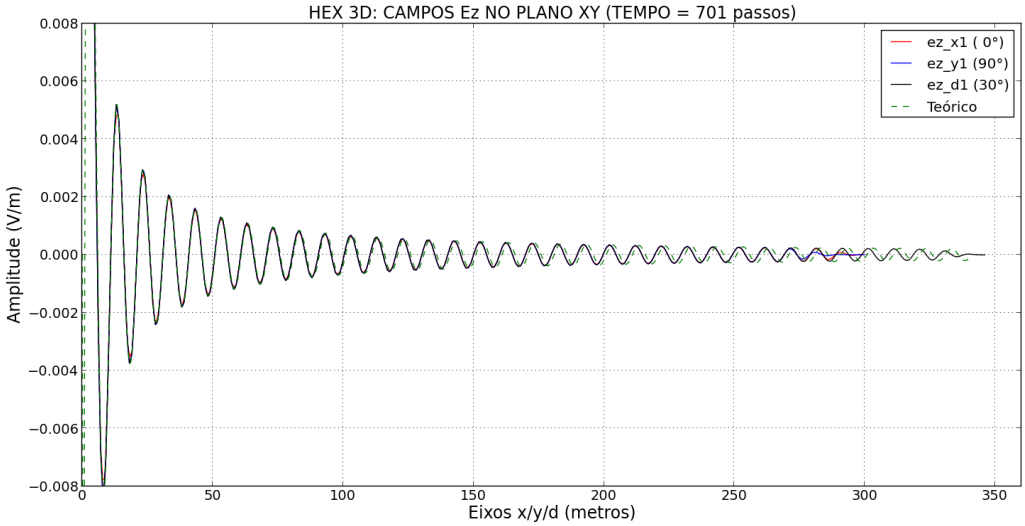


Figura 7.8 - Ondas numéricas nas direções *x* (0°), *y* (90°) e *d* (30°) e onda analítica.

Uma ampliação do final destas curvas é feita na Figura 7.9, sendo fácil observar a dispersão presente nas ondas numéricas em relação à onda analítica.

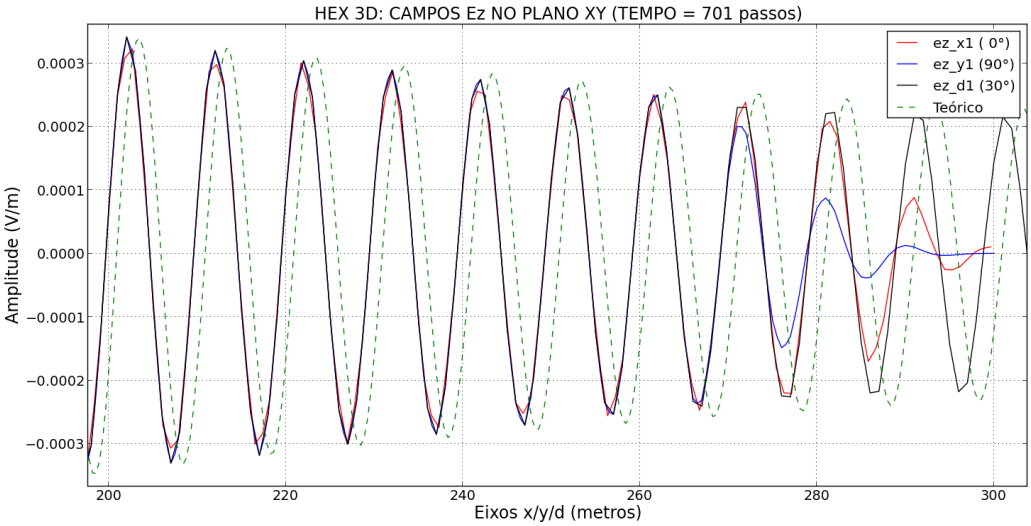
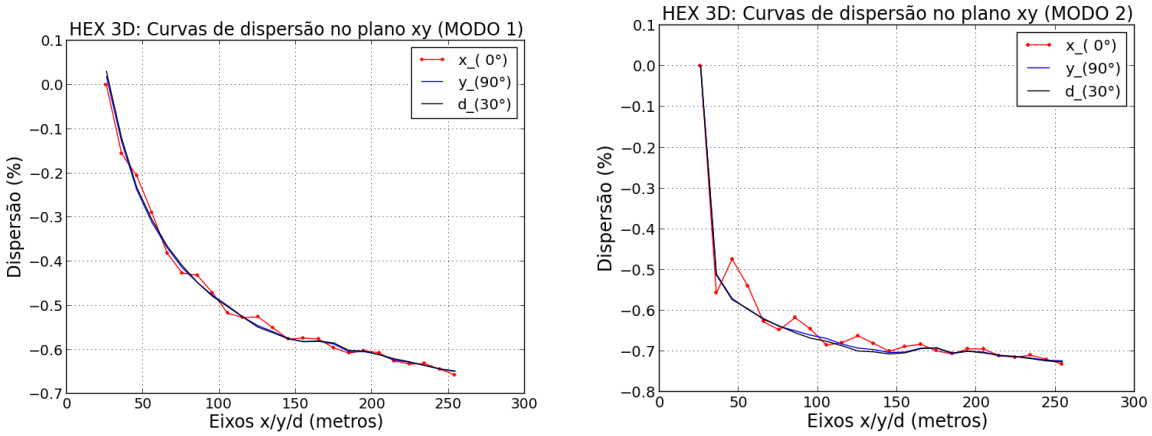


Figura 7.9 - Ampliação da parte final da Figura 7.8 que mostra a dispersão numérica das ondas numéricas em relação à onda analítica no plano x-y da grade de prismas hexagonais.

As curvas de dispersão para as direções *x* (0°), *y* (90°) e *d* (30°) são mostradas na Figura 7.10. Essas curvas de dispersão, diferentemente do método FDTD Yee, são muito próximas nas três direções, embora haja uma certa oscilação da curva de dispersão (modo 1 ou 2) na direção *x* (0°) em torno das curvas nas direções *y* (90°) e *d* (30°). Nestas curvas as medidas de dispersão são mais precisas à medida que as

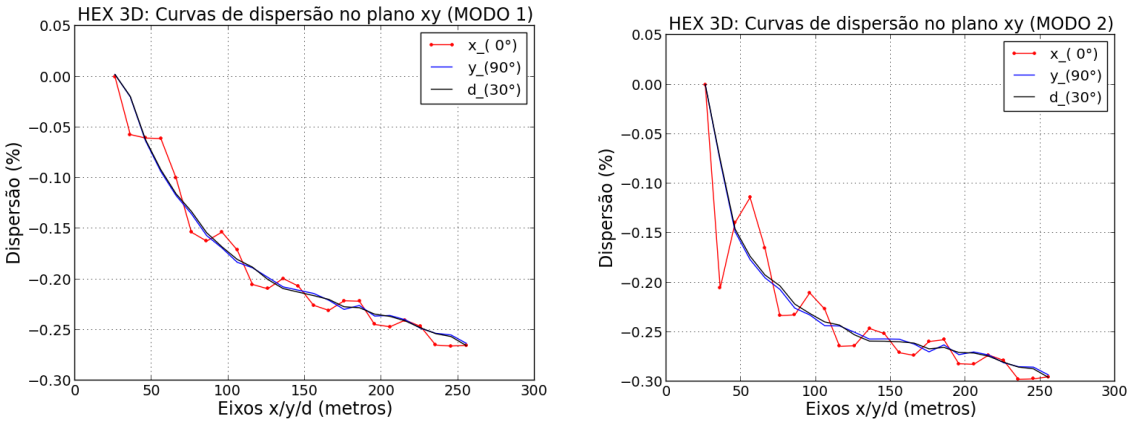
distâncias das ondas numéricas (r^*) e da onda analítica (r) aumentam. A dispersão numérica medida no final da curva (modo 2) na direção x (0°) é -0,730881% que é, em módulo, apenas 0,37% menor que o valor teórico (-0,733579%) da Tabela 4.3.



(a) (b)

Figura 7.10 - Curvas de dispersão (sem compensação) no plano x-y para a grade de prismas hexagonais: (a) modo 1; (b) modo 2.

Uma segunda simulação é feita usando compensação de dispersão simples como definida pela equação (7.1) com (c / v_{med}) = 1,00423964009 obtido da análise de Fourier. Os demais parâmetros são iguais ao da simulação anterior, com exceção da fase inicial da onda analítica que é alterada para β_0 = 163,8155° de forma a ajustar a fase da onda analítica com o ponto de nulo inicial x_0 = 26,0001 m. As curvas de dispersão são mostradas na Figura 7.11.



(a) (b)

Figura 7.11 - Curvas de dispersão (com compensação simples) no plano x-y para a grade de prismas hexagonais: (a) modo 1; (b) modo 2.

Observa-se na Figura 7.11b que essa simples compensação de dispersão reduz as dispersões nos finais das três curvas (modo 2); na direção *x* (0°) para -0,295096%, na direção *y* (90°) para -0,293116% e na direção *d* (30°) para -0,295601%, ou seja, a redução de dispersão nas três direções é dada por um fator de aproximadamente 2,5 vezes em relação à simulação anterior (sem compensação de dispersão). Os valores de dispersão numérica nas três direções são muito próximos devido à baixa anisotropia numérica no plano x-y para a grade de prismas hexagonais.

Uma terceira simulação é feita usando a compensação de dispersão desenvolvida nas equações (7.5) a (7.7), que será chamada nesta tese de compensação de dispersão otimizada. É utilizada a análise de Fourier, desenvolvida na seção 4.1 para a grade de primas hexagonais, com o objetivo de obter a velocidade média normalizada ($v_{med}(\theta)$ / c) em cada posição no plano x-y ($x = i.\Delta x$, $y = j.\Delta y$), usando o número de Courant máximo S_{CFL} = 1 / √ 3,24193735 dado pela equação exata (4.71) com razão R =1,1547, N_λ = 10 e Δd = 1 m. No entanto, na simulação, com o método FDTD com grade de prismas hexagonais, é usado um número de Courant menor dado por S_{CFL} = 1 / √ 3,3334, que garante a estabilidade da simulação. Na Figura 7.12 são mostradas, de forma similar à Figura 7.8, as curvas de onda senoidal nas direções positivas *x* (0°), *y* (90°) e *d* (30°), incluída a curva teórica com valores iniciais A_{s0} = 0,11 V e β_0 = 163,8154° que ajusta a fase da onda analítica com o ponto de nulo inicial x_0 = 26,0001 m.

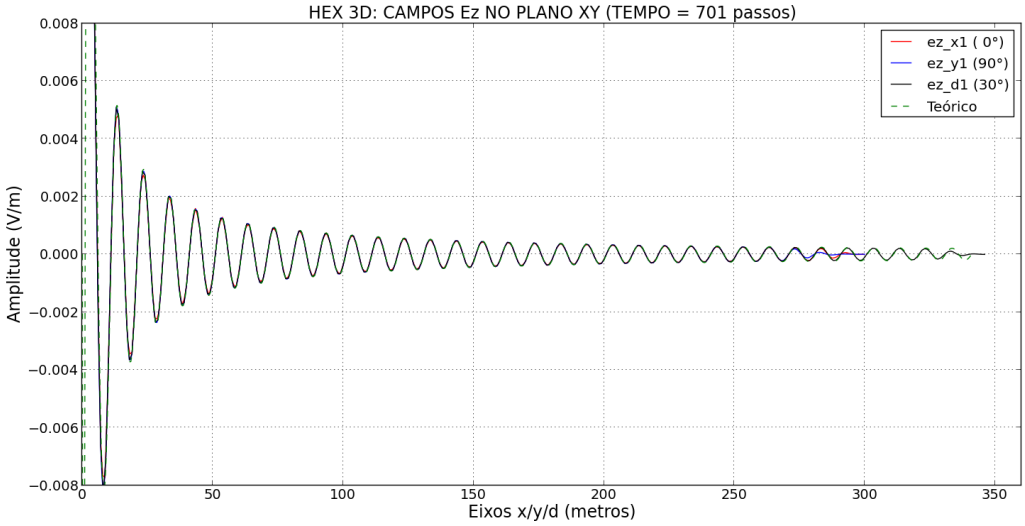


Figura 7.12 - Ondas numéricas nas direções *X* (0°), *y* (90°) e *d* (30°) e onda analítica.

Uma ampliação do final destas curvas é feita na Figura 7.13, onde observa-se uma baixa dispersão numérica presente nas ondas numéricas em relação à onda analítica.

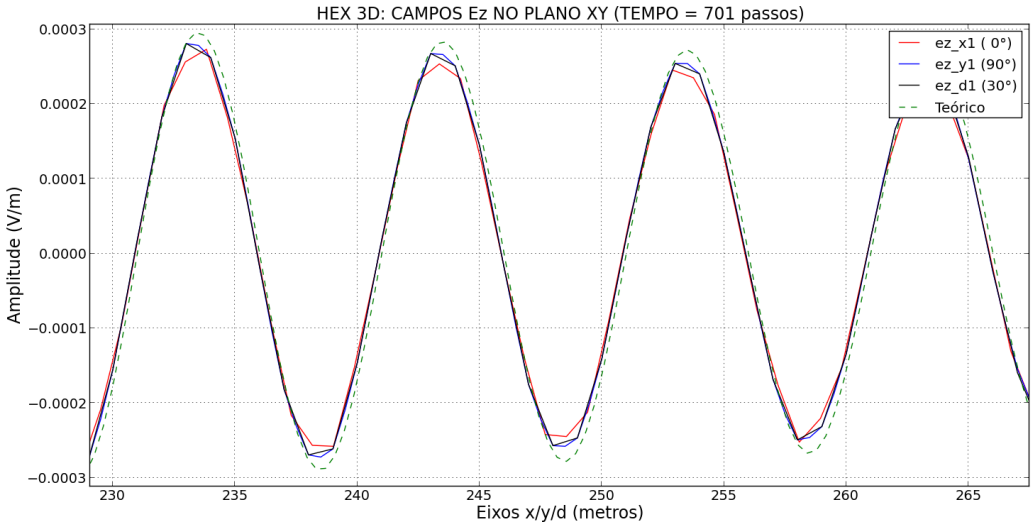
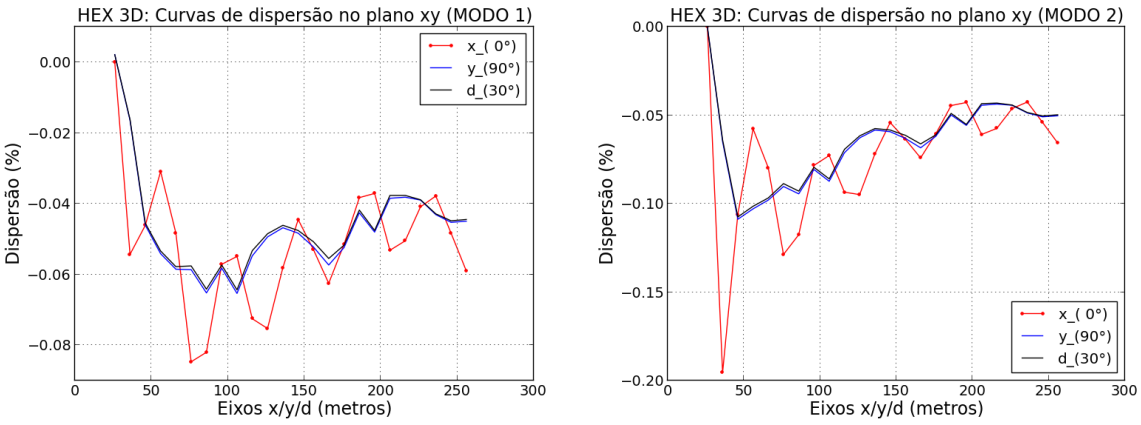


Figura 7.13 - Ampliação da parte final da Figura 7.12 que mostra a dispersão numérica das ondas numéricas em relação à onda analítica no plano x-y da grade de prismas hexagonais.

As curvas de dispersão para as direções *x* (0°), *y* (90°) e *d* (30°) são mostradas na Figura 7.14.

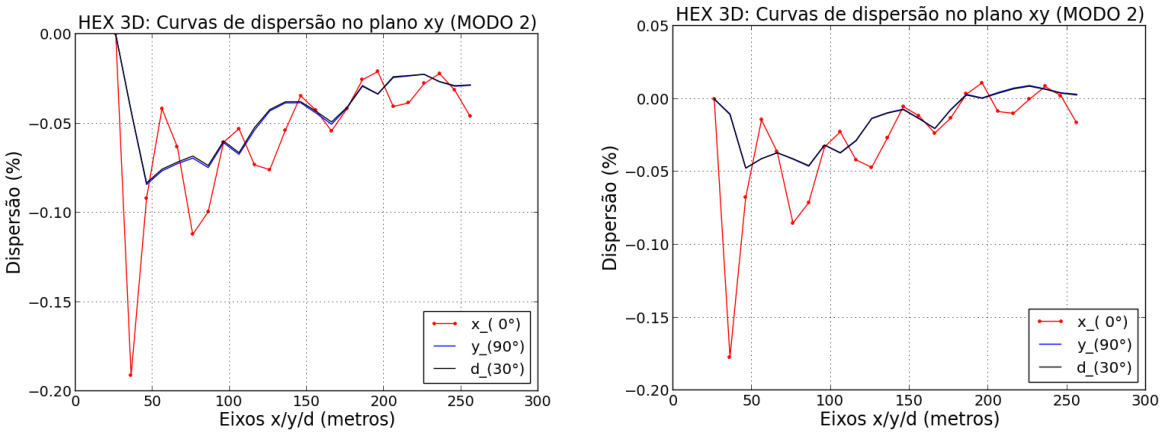


(a) (b)

Figura 7.14 - Curvas de dispersão (com compensação otimizada) no plano x-y para a grade de prismas hexagonais: (a) modo 1; (b) modo 2.

Curvas com dispersão numérica ainda menores podem ser obtidas se utilizado um fator de redução (F_R), levemente menor que um, multiplicando os valores de velocidade média normalizada ($v_{med}(\theta)$ / c) usados nesta subseção. Assim, duas curvas de dispersão (modo 2) são obtidas para fatores de redução F_R = 0,9998 e F_R

= 0,9995 mostradas nas Figuras 7.15a e 7.15b, respectivamente. As curvas de dispersão (modo 2) apresentam os piores valores de dispersão numérica (maior em módulo), sendo portanto escolhidas nesta comparação.



(a) (b)

Figura 7.15 - Curvas de dispersão com compensação otimizada (modo 2) no plano x-y para a grade de prismas hexagonais: (a) fator de redução F_R = 0,9998; (b) fator de redução F_R = 0,9995.

Nesse ponto, é interessante analisar mais profundamente o comportamento das curvas de dispersão (modo 2) das Figuras 7.14b (F_R = 1,0), 7.15a (F_R = 0,9998) e 7.15b (F_R = 0,9995). Assim, três tabelas são feitas com os valores de dispersões médias e finais das curvas nas direções *x* (0°), *y* (90°) e *d* (30°). As curvas de dispersão (modo 2) em cada direção são formadas por 24 pontos de nulo. Assim, para aumentar a precisão dos valores de dispersão média, apenas os 17 pontos de nulo finais, em cada uma das três curvas de dispersão, são usados nos cálculo das dispersões médias.

TABELA 7.1 - COMPARAÇÃO DE DISPERSÕES NUMÉRICAS (F_R = 1,0).

Direção	Dispersão média (%)	Dispersão final (%)
x (0°)	-0,0633324	-0,0656092
y (90°)	-0,0588197	-0,0502354
d (30°)	-0,0578454	-0,0496863

TABELA 7.2 - COMPARAÇÃO DE DISPERSÕES NUMÉRICAS (F_R = 0,9998).

Direção	Dispersão média (%)	Dispersão final (%)
x (0°)	-0,0436360	-0,04612338
y (90°)	-0,0384926	-0,02859126
d (30°)	-0,0379183	-0,02832711

TABELA 7.3 - COMPARAÇÃO DE DISPERSÕES NUMÉRICAS (F_R = 0,9995).

Direção	Dispersão média (%)	Dispersão final (%)
x (0°)	-0,0137427	-0,0164023
y (90°)	-0,0075500	0,00328833
d (30°)	-0,0077602	0,00275917

Observa-se que as dispersões média ou final nas direções *y* (90°) e *d* (30°) são muito similares em cada tabela. A dispersão média na direção *x* (0°) em cada tabela é sempre um pouco maior (em módulo) que nas direções *y* (90°) ou *d* (30°). Observa-se que, quanto menor o fator de redução (F_R), menores são os valores absolutos de dispersão numérica nas três direções. O maior valor absoluto de dispersão numérica em cada tabela é aquele no final da direção *x* (0°); esse valor na Tabela 7.1 é em torno de 11,13 vezes menor que o valor de dispersão final (-0,730881%) na direção *x* (0°) da simulação sem compensação de dispersão (Figura 7.10b). Na Tabela 7.2 o valor final na direção *x* (0°) é 15,84 vezes menor que o valor sem compensação; e na Tabela 7.3 esse valor final na direção *x* (0°) é 44,56 vezes menor que o valor sem compensação. Verifica-se também que as curvas de dispersão (modo 1 ou 2) nas Figuras 7.14 e 7.15, tendem levemente a subir na direção da dispersão zero, confirmando a tendência, já analisada teoricamente nas equações (7.5) a (7.7), do algoritmo com compensação de dispersão otimizada produzir menos dispersão à medida que as ondas numéricas se afastam da antena.

É interessante estimar as densidades de malha para o método FDTD Yee alcançar a mesmas dispersões numéricas finais na direção *x* (0°), como dado nas Tabelas 7.1 a 7.3. Para fazer essa estimativa considera-se que o método FDTD Yee usa compensação de dispersão simples, ou seja, reduz a dispersão numérica (sem compensação) por três vezes (TAFLOVE; HAGNESS, 2005). Na Tabela 7.4 esta estimativa é feita relacionando o número de pontos por comprimento de onda (N_λ) com a dispersão numérica sem compensação na grade Yee. Observa-se na Tabela 7.4 um aumento cada vez maior do parâmetro N_λ na grade Yee para produzir aproximadamente as mesmas dispersões máximas das Tabelas 7.1 a 7.3 referentes a grade de prismas hexagonais com N_λ = 10. Observa-se também que as anisotropias numéricas da grade Yee são muito maiores que a anisotropia numérica da grade de prismas hexagonais (0,0027058% para N_λ = 10), indicando uma compensação não uniforme, como já demonstrado na subseção 7.1.1.

TABELA 7.4 - COMPARAÇÃO DO PARÂMETRO N_λ (MÉTODO YEE) COM A DISPERSÃO NUMÉRICA.

N_λ	Anisotropia (%)	Dispersão (%) sem compensação	Dispersão (%) com compensação
24	0,1427894	-0,1907118	-0,0635706
28	0,1049065	-0,1400513	-0,0466837
47	0,0372325	-0,0496655	-0,0165551

É possível estimar, a partir da Tabela 7.4, a relação entre o número de pontos na grade Yee, a serem armazenados para um certo parâmetro N_λ, e o número de pontos necessários na grade de prismas hexagonais. Para fazer esta estimativa, considera-se que ambas as grades têm as mesmas dimensões L_X, L_Y e L_Z. Na grade Yee tem-se $\Delta x = \Delta y = \Delta z = \Delta$ com $L_X = N_X \Delta$, $L_Y = N_Y \Delta$, $L_Z = N_Z \Delta$ e $\Delta = \lambda / N_\lambda$, onde λ é o comprimento de onda físico (real). Para a grade de prismas hexagonais tem-se $\Delta x = \Delta z = \Delta d / R$, $\Delta y = \Delta d / 2$, razão $R = 2/\sqrt{3} \approx 1{,}1547$ com $L_X = N_{X0}\, \Delta d / R$, $L_Y = N_{Y0}\, \Delta d / 2$, $L_Z = N_{Z0}\, \Delta d / R$ e $\Delta d = \lambda / N_{\lambda 0}$. Os parâmetros N_λ e $N_{\lambda 0}$ são específicos à grade Yee e à grade de prismas hexagonais, respectivamente. Logo, para a grade Yee tem-se

$$V = L_X . L_Y . L_Z = N_X . N_Y . N_Z . \Delta^3 = N_X . N_Y . N_Z . \left(\frac{\lambda}{N_\lambda}\right)^3 \tag{7.15}$$

E para a grade de prismas hexagonais tem-se:

$$V = L_X . L_Y . L_Z = N_{X0} . N_{Y0} . N_{Z0} . \frac{(\Delta d)^3}{2R^2} = N_{X0} . N_{Y0} . N_{Z0} . \frac{1}{2R^2} \left(\frac{\lambda}{N_{\lambda 0}}\right)^3 \tag{7.16}$$

Igualando as equações (7.15) e (7.16) e rearranjando termos obtém-se a densidade de malha (F_D) da grade Yee em relação à grade de primas hexagonais como:

$$F_D = \frac{N_X . N_Y . N_Z}{N_{X0} . N_{Y0} . N_{Z0}} = \frac{1}{2R^2} \left(\frac{N_\lambda}{N_{\lambda 0}}\right)^3 = \frac{3}{8} \left(\frac{N_\lambda}{N_{\lambda 0}}\right)^3 \tag{7.17}$$

A partir da equação (7.17), usando os valores de parâmetro N_λ da Tabela 7.4 e com $N_{\lambda 0} = 10$, obtém-se a Tabela 7.5.

TABELA 7.5 - COMPARAÇÃO DE DENSIDADES DE MALHA DA GRADE YEE EM RELAÇÃO À GRADE DE PRIMAS HEXAGONAIS ($N_{\lambda 0}$ = 10).

Parâmetro N_λ da grade Yee	Densidade de malha (F_D) para parâmetro $N_{\lambda 0}$ = 10
10	0,375
13,87	1,0
24	5,184
28	8,232
47	38,93

É fácil perceber na Tabela 7.5 o aumento de densidade de malha (F_D) da grade Yee em relação à grade de prismas hexagonais à medida que o parâmetro N_λ aumenta acima do limiar (13,87); observa-se que se o parâmetro N_λ dobrar, a densidade de malha (F_D) aumenta em oito vezes. Este aumento em densidade de malha (F_D) significa aumento de uso de memória e também aumento de tempo de processamento para o método FDTD Yee.

Finalmente, a Tabela 7.6 faz uma estimativa do aumento relativo de distância (r / L_Z) na Figura 7.1, que pode ser alcançado usando a compensação de dispersão otimizada para a grade de prismas hexagonais, em função do parâmetro N_λ. A partir da equação (7.5) é obtida a distância relativa (r / L_Z) = 0,5 / tan(θ_{max}). O ângulo θ_{max} corresponde aqui ao ângulo nos planos x-z ou y-z (Figura 7.1), que produz a mesma anisotropia máxima do plano x-y para o parâmetro N_λ.

TABELA 7.6 - DISTÂNCIA RELATIVA (r / L_Z) EM FUNÇÃO DO PARÂMETRO N_λ.

N_λ	Anisotropia (%) no plano x-y	Ângulo θ_{max} (°) nos planos x-z ou y-z	Distância relativa (r / L_Z)
10	0,00275058	1,88	15,23
12	0,00130488	1,571	18,23
14	0,00070434	1,349	21,23
16	0,00041287	1,182	24,23
18	0,00025775	1,0515	27,24
20	0,00016911	0,947	30,24
40	0,00001056	0,47407	60,43

Verifica-se um aumento linear da distância relativa (r / L_Z) em função do parâmetro N_λ. Por exemplo, se o parâmetro N_λ dobrar, a distância relativa (r / L_Z) também dobra. É importante notar que, se a antena for colocada no teto (o que é usual na prática), as

distâncias relativas (r / L_Z) serão o dobro dos valores da Tabela 7.6. Como um exemplo prático, se o andar tiver uma altura (L_Z) de 3 metros e o parâmetro N_λ = 10 for usado, as larguras máximas (L_X = L_Y) do andar serão da ordem de 90 metros (antena a meia altura) como na Figura 7.1 ou 180 metros (antena no teto) independentemente da frequência real usada, desde que seja mantido o valor do parâmetro N_λ.

7.2 CONSIDERAÇÕES FINAIS PARA A COMPENSAÇÃO DE DISPERSÃO NUMÉRICA NA GRADE DE PRISMAS HEXAGONAIS

Para o método FDTD Yee existem outras formas mais eficientes de compensar a dispersão numérica (SHLAGER; SCHNEIDER, 2003; PANARETOS; ABERLE; DÍAZ, 2006; SMITH et al., 2012; CHEN et al., 2013), que aquela usada na subseção 7.1.1. Uma forma eficiente é incorporar ao algoritmo Yee convencional (explícito) um esquema de diferença finita, desenvolvido por J. Fang (SHLAGER; SCHNEIDER, 2003; TAFLOVE; HAGNESS, 2005) a partir da expansão em série de Taylor; este algoritmo será aqui denominado de esquema Fang(2,4), onde o número dois corresponde a precisão de 2ª ordem no tempo e o numero quatro a precisão de 4ª ordem no espaço (SMITH et al., 2012). Assim, cada campo elétrico ou magnético num ponto da grade é calculado em função de campos em quatro pontos adjacentes em vez dos dois pontos adjacentes usados no método Yee convencional. A anisotropia numérica máxima para o esquema Fang(2,4), nos planos x-y, x-z ou y-z, é dada pela seguinte fórmula aproximada (TAFLOVE; HAGNESS, 2005):

$$\Delta v_{YEE\_FANG\_2D}(\%) \approx \left(\frac{\pi}{N_\lambda}\right)^4 \frac{100}{18} \tag{7.18}$$

Observa-se, que para número de pontos por comprimento de onda N_λ = 10, a anisotropia numérica, calculada pela equação (7.18), é igual a 0,05411% e, portanto, aproximadamente 15 vezes menor que a anisotropia numérica para o método FDTD Yee convencional, como dado na Tabela 4.1. Verifica-se, também, comparando a equação (7.18) com a equação (4.76), que a anisotropia numérica no método FDTD com esquema Fang(2,4) é sempre 20 vezes maior, para qualquer parâmetro N_λ, que aquela da grade 2D de hexágonos ou da grade de prismas hexagonais no plano x-y (Tabelas 4.1 e 4.2).

Métodos FDTD Yee modificados têm sido objeto de pesquisa recente (SMITH et al., 2012) para alcançar precisão de 2ª ordem no tempo e 4ª ordem no espaço, denominado esquema general(2,4) ou precisão de 4ª ordem em tempo e espaço, denominado esquema general(4,4). Os esquemas general(2,4) e general(4,4) têm produzidos menores valores de anisotropia numérica que o esquema Fang(2,4). Por exemplo, para parâmetro $N_\lambda = 10$ a anisotropia numérica nos esquemas general(2,4) ou general(4,4) tem valor em torno de 0,001% (nos planos x-y, x-z ou y-z) que é quase 3 vezes menor que o valor de anisotropia 0,00276% obtido para grade de prismas hexagonais no plano x-y (Tabela 4.1). É importante observar que estes métodos FDTD de alta ordem são de banda larga (SHLAGER; SCHNEIDER, 2003; SMITH et al., 2012), ou seja, a anisotropia e dispersão numéricas diminuem a medida que o parâmetro N_λ aumenta; logo, se um parâmetro N_λ mínimo é escolhido para o menor comprimento de onda (maior frequência) presente na onda eletromagnética que se propaga na grade 3D, todas as outras frequências presentes na onda seram amostradas com resolução maior que o parâmetro N_λ mínimo, o que garante valores baixos de anisotropia e dispersão numéricas em toda a banda de frequência ocupada pela onda.

No entanto, esta redução grande na anisotropia numérica, produzida por estes esquemas FDTD explícitos de ordem mais alta, tem um custo (TAFLOVE; HAGNESS, 2005; INAN; MARSHALL, 2011). O cálculo das diferenças finitas espaciais com 4ª ordem de precisão nas interfaces com materiais diferentes, nas quais surgem alterações bruscas de permissividade elétrica, permeabilidade magnética ou condutividade elétrica, exigem cuidados especiais. Isto ocorre devido a natureza não local do processo de diferenciação numérica no espaço (uso de quatro pontos), o qual pode transportar dados de campos elétrico ou magnético através das descontinuidades (alterações de tipo de material) em uma maneira não física. Assim, é exigido um tratamento matemático específico em cada interface onde existam alterações bruscas de tipo de material; logo, condições especiais de contorno em tais interfaces aumentam significativamente a complexidade do programa computacional usado para calcular os campos em estruturas com muitos objetos e distintos materiais na grade 3D, como é o caso dos ambientes internos em edificações.

Desta forma, o novo método FDTD, desenvolvido para a grade de prismas hexagonais com o seu respectivo algoritmo de compensação otimizada de dispersão numérica, tem uma importante vantagem na comparação com métodos FDTD de

ordem mais alta (em espaço); pois, tendo precisão de 2ª ordem no espaço, não necessita nenhum tratamento especial em interfaces com diferentes materiais, o que possibilita ao programa computacional calcular os campos diretamente a partir das equações de diferença finita, independentemente da quantidade ou localização das interfaces (contornos) existentes na estrutura sob modelamento.

O algoritmo de compensação otimizada de dispersão numérica, desenvolvido nas equações (7.5) a (7.7) para a grade de prismas hexagonais, mostrou-se eficaz conforme comprovado pelas simulações realizadas na subseção 7.1.2. Este algoritmo é muito simples, não altera a forma matemática das equações de diferença finita e mantém a característica de banda larga deste método FDTD; ele produz uma correção de dispersão numérica com um valor baixo desejado e que é isotrópica, ou seja, a dispersão numérica é praticamente igual em todas as direções no plano x-y, e tem, também, uma variação desprezível na direção *z*, desde que as dimensões L_X e L_Y (no andar de um edifício) sejam muito maiores que a dimensão L_Z (altura) deste andar.

Apesar da análise de redução de dispersão numérica para a grade de prismas hexagonais ter sido feita apenas para um meio homogêneo e sem perdas, ou seja, o espaço livre (ou vácuo), os resultados obtidos neste capítulo mostram-se promissores para aplicação em meios não homogêneos e com perdas, que é o caso de edificações que usam redes *Wireless*.

8 ALGUMAS CONSIDERAÇÕES NA IMPLEMENTAÇÃO COMPUTACIONAL

Neste capitulo analisar-se-á brevemente o conjunto de programas desenvolvido para realizar a pesquisa com o método FDTD com grade de prismas hexagonais. A estrutura básica deste conjunto de programas é mostrada na Figura 8.1.

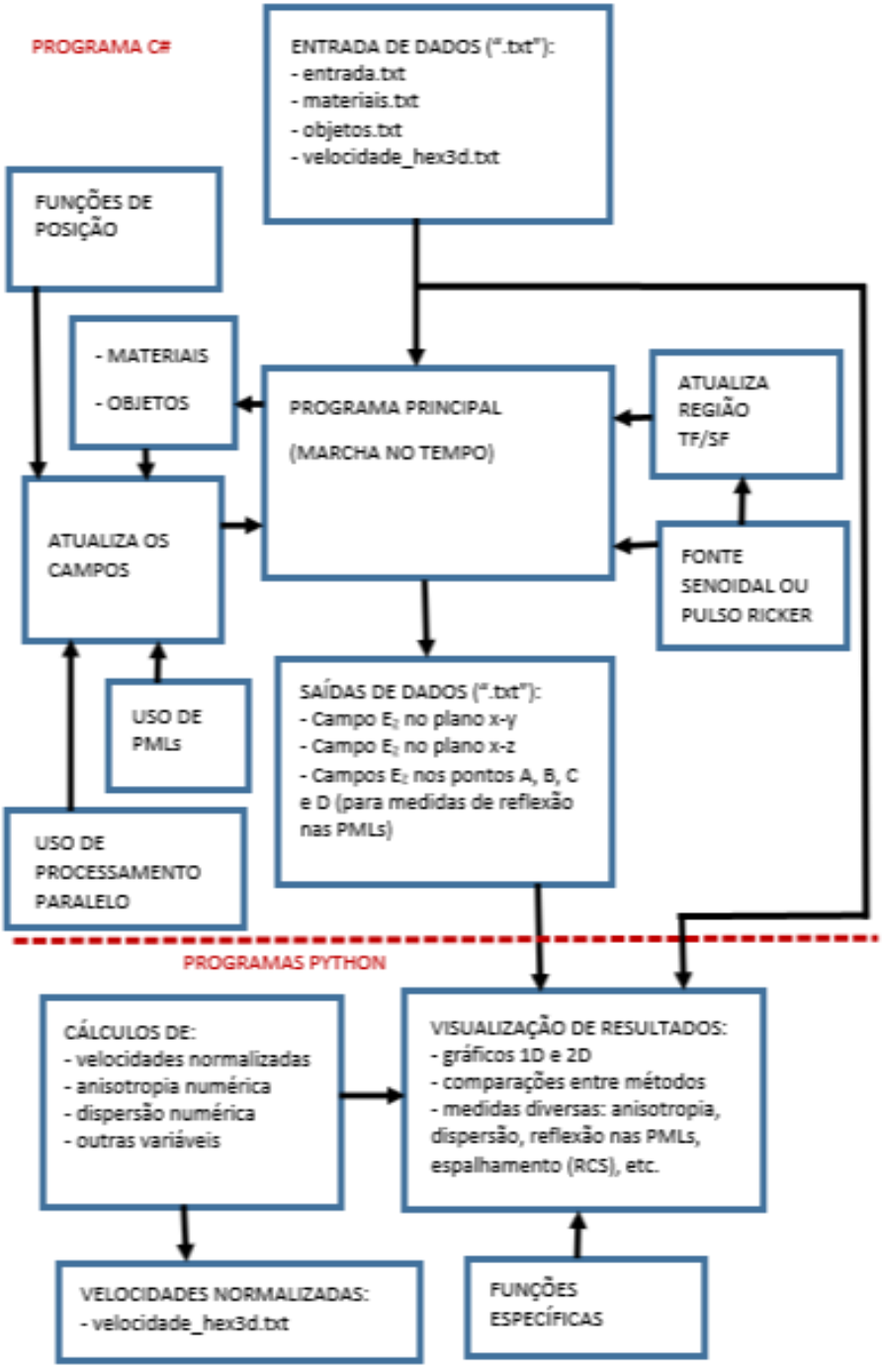


Figura 8.1 - Desenho esquemático simplificado dos programas desenvolvidos para o método FDTD com grade de primas hexagonais e o método FDTD Yee.

Esta estrutura é, também, válida para o método FDTD Yee, que é frequentemente usado para fazer comparações com o método FDTD com grade de prismas hexagonais.

A entrada de dados constitui-se de arquivos no formato texto (".txt"). Por exemplo, o arquivo "entrada.txt" fornece vários parâmetros fundamentais ao programa executável na linguagem de programação C# tais como: N_X, N_Y, N_Z, N_λ, S_{CFL}, Δd ou Δx, número máximo de passos no tempo (n), tipo de fonte (senoidal ou pulso, e se ela é *hard* ou *soft*), uso de PMLs e suas espessuras, uso de região TF/SF e suas dimensões, número de materiais, número de objetos, etc. (ver Apêndices 2 e 3 para mais detalhes). O arquivo "materiais.txt" permite colocar vários tipos de materiais nas grades 3D de ambos os métodos FDTD. Cada material é definido por permissividade relativa (ε_r), permeabilidade relativa (μ_r), condutividade elétrica (σ) e condutividade magnética (σ_m). O arquivo "objetos.txt" permite definir três tipos de objetos: placa retangular, cilindro e esfera. Cada um destes objetos pode ser definido o número de vezes desejado em quaisquer posições da grade 3D, com as dimensões desejadas e com os tipos de materiais definidos no arquivo "materiais.txt". A partir dos arquivos "materiais.txt" e "objetos.txt" o programa principal cria uma matriz 3D que define o tipo de material em cada posição da grade 3D como um número inteiro; por exemplo, o vácuo é 0, cobre é 1, madeira é 2, concreto é 3, etc. Assim, se esses números inteiros forem de tamanhos iguais a um byte, poderão representar até 256 tipos diferentes de materiais e reduzir bastante o uso de memória. A partir destes números que definem o tipo de material em cada posição da grade 3D, as funções que atualizam os campos elétricos e magnéticos são capazes de acessar os valores das matrizes de coeficientes (C_{H10}, C_{HA}, C_{HB}, C_{E10}, C_{EA}, C_{EB}, etc.) calculados a partir dos parâmetros ε_r, μ_r, σ e σ_m do especifico material naquela posição.

No plano x-y, da grade de prismas hexagonais, em cada linha (que corresponde ao índice j na direção y) os campos elétricos ou magnéticos são calculados apenas nas posições (índices i na direção x) pares ou ímpares desta linha (ver Figura 3.2). Assim, para economizar memória é conveniente usar matrizes de duas dimensões com índices p e k; o índice k corresponde às posições na direção z e o índice p é definido em função dos índices i e j onde o campo é calculado. As funções de posição determinam os índices p para cada posição (i, j) dos oito campos da grade de prismas hexagonais. Assim, para cada campo o número total de pontos é reduzido em torno de ($N_X.N_Y.N_Z$ / 2). Funções de posição mais sofisticadas são necessárias para as

PMLs, de forma que unicamente as posições *p* e *k* dentro das PMLs sejam armazenadas nas variáveis usadas na formulação CPML (ver Apêndice 2). Uma formulação semelhante de funções de posição, mas mais simples, é também usada no método FDTD Yee (ver Apêndice 3). No processo de visualização de gráficos 1D e 2D dos campos elétricos (E_Z) nos planos x-y e x-z, para a grade de prismas hexagonais, é necessário interpolar as posições (*i*, *j*) onde os campos não são calculados (ver Apêndice 4).

No cálculo das condutividades, presentes nas PMLs da grade de prismas hexagonais, foi desenvolvido um algoritmo, no programa principal (ver Apêndice 2), capaz de gerar uma variação de condutividades igual à moldura (com canto a 60°) no plano x-y, mostrada na Figura 5.5. Essa variação de condutividades é feita numa matriz 2D (índices *i* e *j*) usando números inteiros em que o número menor corresponde ao início da PML (condutividade mínima ou zero) e o número maior ao final da PML (condutividade máxima). Assim, as funções usadas na formulação CPML, a partir do número inteiro em uma certa posição (i, j) das PMLs, podem acessar os coeficientes desejados, que foram calculados para o valor de condutividade naquela posição. O objetivo aqui é também reduzir o uso de memória.

Uma versão com processamento paralelo (usando quatro CPUs), que reduziu em aproximadamente quatro vezes o tempo de processamento para o método FDTD com grade de prismas hexagonais (ver Apêndice 1), foi usada na seção 4.7, devido às grandes dimensões necessárias (L_X, L_Y, L_Z) na grade, para que fosse possível medir com mais precisão as anisotropias numéricas nas direções desejadas nos planos x-y e x-z. No entanto, nesta versão não é usada a formulação com PMLs.

Inicialmente, os programas de ambos os métodos FDTD foram implementados em linguagem de programação C++ sob Windows 7 ou 8. No entanto, uma limitação grande de uso de memória foi verificada; as matrizes dos oito campos (em conjunto) não podiam usar mais que 2 GB. No entanto, usando a linguagem de programação C# (sob Windows 7 ou 8 em computador de 64 bits) a matriz de cada campo pode usar até 2 GB. Na prática, o tamanho total das oito matrizes é limitado pela memória RAM disponível no computador usado nas simulações, que foi em torno de 6 GB. Não é recomendável usar um tamanho de memória, para as matrizes, maior que a memória RAM disponível num certo computador, pois isto reduz em muito a velocidade de processamento. Todavia, se for necessário, há uma forma de se usar dimensões muitos maiores para as matrizes de campo na linguagem de programação C#, que

consiste em criar uma classe chamada de Matrix (ver Apêndice 2) usando a classe de coleção *List*<*T*> (SHARP, 2014); no entanto, não houve a necessidade de usar essa classe nesta pesquisa.

Na implementação do algoritmo de marcha de tempo, no programa principal, é importante usar a ordem correta de atualizações de campos elétricos e magnéticos com a atualização das regiões de campo total e campo espalhado (TF/SF), de forma que não haja vazamento do campo incidente para a região de campo espalhado (ver Apêndices 2 e 3)

Programas em linguagem Python (ver Apêndices 4 a 10) foram usados para visualizar as saídas dos campos elétricos (E_Z) nos planos x-y e x-z, em formatos 2D (graduação de cores) e 1D (curvas). Funções especificas foram desenvolvidas para medir as anisotropia e dispersão numéricas, e, também, os níveis de reflexão nas PMLs a partir das matrizes de campo E_Z fornecidas pelo programa C#. Para medir a anisotropia e dispersão numéricas é necessário um algoritmo que calcule com precisão os pontos de nulo nas ondas senoidais numéricas e analítica; é necessário fazer uma interpolação entre o ponto anterior (positivo) e o ponto posterior (negativo) para se achar a posição exata do ponto de nulo. Para calcular as posições desses pontos de nulo é possível usar uma interpolação linear ou uma interpolação denominada aqui de senoidal; essa interpolação senoidal é mais precisa que a interpolação linear, e usa uma curva senoidal valendo-se do fato que as ondas senoidais atenuam em função inversa da distância na grade 3D. Funções analíticas para o cálculo de RCS de placa retangular metálica e esfera metálica foram usadas junto com funções específicas para determinar as RCSs numéricas das simulações em ambos os métodos FDTD. Programas específicos em Python, usando a análise de Fourier, foram desenvolvidos para calcular as velocidades normalizadas, anisotropia e dispersão numéricas teóricas nos planos x-y, x-z e y-z de ambos os métodos FDTD. Um programa Python foi usado para ajudar na dedução simbólica do número de Courant para a grade de prismas hexagonais.

Por fim, observa-se que a implementação computacional da formulação FDTD para a grade de prismas hexagonais é muito mais complexa que aquela do método FDTD Yee, devido principalmente às funções de posição usadas para a grade 3D e PMLs. Essa maior complexidade de implementação torna o método FDTD com grade de prismas hexagonais mais lento computacionalmente que o método FDTD Yee. Por exemplo, nas simulações com esfera metálica da seção 6.5, que utilizam uma grade

3D relativamente grande, o método FDTD Yee foi em torno de sete vezes mais rápido que o método FDTD com grade de prismas hexagonais. No entanto, tais desvantagens podem ser superadas pelo método FDTD com grade de prismas hexagonais, quando esse utiliza a compensação de dispersão otimizada analisada no capítulo 7.

9 CONCLUSÕES E PERSPECTIVAS

Foi provado teoricamente (análise de Fourier) e praticamente (simulações numéricas com este novo método FDTD) que, para um certo número de pontos por comprimento de onda maior ou igual a dez, o método FDTD com grade de prismas hexagonais tem algumas centenas de vezes menos anisotropia numérica máxima no plano x-y que o método FDTD Yee; e ele tem, em ambos os planos x-z e y-z, em torno de 25% menos anisotropia numérica que o método FDTD Yee. O método FDTD com grade de prismas hexagonais tem, também, em torno de 33% menos dispersão numérica que o método FDTD Yee.

Uma formulação de camada de absorção (CPML) para a grade de prismas hexagonais foi desenvolvida com níveis de reflexão um pouco maiores na comparação com a grade Yee; se menores níveis de reflexão forem necessários nas camadas de absorção (PMLs) da grade de prismas hexagonais, é suficiente aumentar as espessuras das PMLs.

Ondas planas foram introduzidas na grade de prismas hexagonais, usando a formulação de regiões de campos total e espalhado (TF/SF). As medidas de níveis de espalhamento de campo elétrico em dois objetos metálicos (placa retangular ou esfera), dentro da grade de prismas hexagonais, foram, em geral, muito similares às medidas realizadas na grade Yee. Isto significa que as diferenças geométricas, da grade de prismas hexagonais em relação à grade Yee, não produzem reduções perceptíveis (ou significativas) de precisão nas simulações envolvendo campos espalhados por objetos presentes na grade de prismas hexagonais.

Um método de compensação de dispersão numérica eficiente e simples foi desenvolvido para a grade de prismas hexagonais, desde que as dimensões no plano x-y, da estrutura a ser simulada, sejam muito maiores que a dimensão *z* (altura). Assim, demonstrou-se a possibilidade de uso do método FDTD com grade de prismas hexagonais para simulação de propagação de ondas eletromagnéticas irradiadas por redes *Wireless* usadas em andares de edifícios ou outras edificações, que, normalmente, possuem as dimensões no plano horizontal (plano x-y) muito maiores que a altura (dimensão *z*).

O método FDTD com grade de prismas hexagonais tem, como principal desvantagem em relação ao método FDTD Yee, uma implementação computacional mais complexa. No entanto, usando a compensação de dispersão otimizada analisada

no capítulo 7, o método FDTD com grade de prismas hexagonais poderá ter uma densidade de malha algumas dezenas de vezes menor que o método FDTD Yee. As consequências práticas serão maior precisão nas simulações, reduções no tamanho da memória e tempo de processamento computacional.

Nesta tese foram desenvolvidas algumas ferramentas básicas para permitir o uso prático do método FDTD com grade de prismas hexagonais tais como: fontes pontuais (senoidal ou pulso), camadas de absorção (PMLs), regiões de campos total e espalhado (TF/SF), alguns tipos de objetos (placa retangular, cilindro e esfera) e possibilidade de uso de diversos materiais na grade 3D. No entanto, há várias outras formulações presentes no método FDTD Yee, que podem ser desenvolvidas ou adaptadas para o método FDTD com grade de prismas hexagonais tais como: uso de onda plana com direção e polarização arbitrárias para as regiões de campos total e espalhado (TF/SF); transformação de campo próximo para campo distante; uso de materiais dispersivos, anisotrópicos e não-lineares; antenas de fio e outros tipos mais complexos de antenas; uso de processamento paralelo com CPUs e/ou GPUs; incorporação de objetos geometricamente mais complexos para emular ambientes físicos presentes em redes *Wireless*; formas de visualização de resultados mais sofisticadas. É importante lembrar que a formulação FDTD para grade de prismas hexagonais, desenvolvida nesta tese para propagação de ondas eletromagnéticas, pode também ser adaptada para simulação de propagação de ondas acústicas.

Espera-se que esta pesquisa possa fornecer avanços na modelagem de radiopropagação em ambientes internos no contexto de redes *Wireless*, além de contribuir com novas ideias que sejam úteis na pesquisa de métodos numéricos em eletromagnetismo e acústica.

REFERÊNCIAS

ARFKEN, G. B.; WEBER, H. J. **Física matemática**: métodos matemáticos para engenharia e física. Rio de Janeiro: Elsevier, 2007.

BALANIS, C. A. **Advanced engineering electromagnetics**. New York: John Wiley & Sons, 1989.

BÉRENGER, J. P. A perfectly matched layer for the absorption of electromagnetic waves. In: JOURNAL OF COMPUTACIONAL PHYSICS. v. 114, p. 185-200, 1994.

CHEN, W. J.; SHAO, W.; LI, J. L.; WANG, B. Z. Numerical dispersion analysis and key parameter selection in Laguerre-FDTD method. In: IEEE MICROWAVE AND WIRELESS COMPONENTS LETTERS. v. 23, n. 12, december 2013.

CHEW, W. C.; WEEDON, W. H. A 3D perfectly matched medium from modified Maxwell´s equations with stretched coordinates. In: IEEE MICROWAVE GUIDED WAVE LETTERS. v. 7, p. 599-604, 1994.

DESCOMBES, S.; DUROCHAT, C.; LANTERI, S.; MOYA, L.; SCHEID, C.; VIQUERAT, J. Recents advances on a DGTD method for time-domain electromagnetics. In: PHOTONICS AND NANOSTRUCTURES: FUNDAMENTALS AND APPLICATIONS. v. 11, p. 291-302, 2013.

FEI, X.; XIAOHONG, T.; XIANJING, Z. The construction of low-dispersive FDTD on hexagon. In: IEEE TRANS. ON ANTENNAS AND PROPAG. v. 53, n. 11, p. 3697-3703, 2005.

GARCIA, S.; PANTOJA, M.; COEVORDEN, C. de J. van; BRETONES, A.; MARTIN, R. A new hybrid DGTD/FDTD method in 2-D. In: IEEE TRANSACTIONS ON MICROWAVE AND WIRELESS COMPONENTS LETTERS. v. 18, n.12, p. 764-766, 2008.

GEDNEY, S. D. An anisotropic perfectly matched layer absorbing media for the truncation of FDTD lattices. In: IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION. v.44, p. 1630-1639, 1996.

GRANDE, A.; PEREDA, J. A.; SERROUKH, A.; BARBA, I.;CABECEIRA, A. C.; REPRESA, J. Reinterpreting four-stage split step FDTD methods as two-stage methods. In: IEEE TRANS. ON ANTENNAS AND PROPAG. v. 61, n. 11, november 2013.

GRANDE, A.; PEREDA, J. A. Accuracy limitations of the locally one-dimensional FDTD technique. In: IEEE ANTENNAS AND WIRELESS PROPAGATION LETTERS. v. 13, 2014.

HAMILTON, B.; BILBAO, S. Hexagonal vs. rectilinear grids for explicit finite differences schemes for the two-dimensional wave equation. In: PROCEEDINGS OF MEETINGS ON ACOUSTICS, 2013, Montreal, Canada. v. 19, p.1-9. http://dx.doi.org/ 10.1121/1.4800308.

HEH, D. Y.; TAN, E. L. Further reinterpretation of multi-stage implicit FDTD schemes. In: IEEE TRANS. ON ANTENNAS AND PROPAG. v. 62, n. 8, august 2014.

HESTHAVEN, J. S.; WARBURTON, T. Nodal high-order methods on unstructured grids. I. time-domain solution of Maxwell's equations. In: J. COMP. PHYS. v. 181, n. 1, p. 186-221, 2002.

INAN, U. S.; MARSHALL, R. A. **Numerical electromagnetics:** the FDTD method. New York: Cambridge University Press, 2011.

JIN, J.M. **Theory and computation of electromagnetic fields**. New Jersey: John Wiley & Sons, 2010.

JOAQUIM, M.; SCHEER, S. Análise de anisotropia numérica de grade de prismas hexagonais utilizando o método das diferenças finitas no domínio do tempo. In: MOMAG 2014: 16º SBMO - SIMPÓSIO BRASILEIRO DE MICRO-ONDAS E OPTOELETRÔNICA, 11º CBMag - CONGRESSO BRASILEIRO DE ELETROMAGNETISMO, 2014, Curitiba. **Anais**... Curitiba: Ed. UTFPR, 2014. p. 290-295. 1 CD-ROM; $4^{3/4}$ pol.

JOAQUIM, M.; SCHEER, S. Finite-difference time-domain method for three-dimensional grid of hexagonal prisms. In: WAVE MOTION. v. 63C, p. 32-54, 2016. http://dx.doi.org/10.1016/j.wavemoti.2016.01.005.

KABAKIAN, V.; SHANKAR, V.; HALL, W. Unstructered grid-based discontinuous Garlekin method for broadband electromagnetic simulations. In: JOURNAL OF SCIENTIFIC COMPUTING. v. 20, n. 3, p. 405-431, 2004.

KONG, Y. D.; CHU, Q. X. High-order split-step unconditionally-stable FDTD methods and numerical analysis. In: IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION. v. 59, n. 9, september 2011.

KONG, Y. D.; CHU, Q. X.; LI, R.. L. A compact fourth-order unconditionally stable six-stages split-step FDTD method and numerical analysis. In: MICROWAVE AND OPTICAL TECHNOLOGY LETTERS. v. 56, n. 5, may 2014a.

KONG, Y. D.; CHU, Q. X.; LI, R.. L. Efficient unconditionally stable one-step leapfrog ADI-FDTD method with low numerical dispersion. In: IET MICROWAVES, ANTENNAS & PROPAGATION. v. 8, iss. 5, p. 337-345, 2014b, doi: 10.1049/iet-map. 2013.0269.

LAI, Z. et al. On the use of an intelligent ray launching for indoor scenarios In: ANTENNAS AND PROPAGATION (EUCAP), 2010 PROCEEDINGS OF THE FOURTH EUROPEAN CONFERENCE ON 2010.

LIANG, D.; YUAN, Q. The spatial fourth-order energy-conserved S-FDTD scheme for Maxwell's equations. In: JOURNAL OF COMPUTACIONAL PHYSICS. v. 243, p. 344-364, 2013.

LIU, Y. Fourier analysis of numerical algorithms for the Maxwell equations. In: JOURNAL OF COMPUTACIONAL PHYSICS. v. 124, p. 396-416, 1996.

NEVE, M. J. et al. Physical layer engineering for indoor wireless systems in the twenty-first century. In: 2010 LOUGHBOROUGH ANTENNAS & PROPAGATION CONFERENCE, 8-9 november 2010, Loughborough, UK.

PANARETOS, A. H.; ABERLE, J. T.; DÍAZ, R. E. A three-dimensional finite-difference time-domain scheme based on a transversely extended-curl operator. In: IEEE TRANSACTIONS ON MICROWAVE THEORY AND TECHNIQUES. v. 54, n. 12, december 2006.

POTTER, M. E.; LAMOUREUX, M.; NAUTA, M. D. An FDTD scheme on a face-centered-cubic (FCC) grid for the solution of the wave equation. In: JOURNAL OF COMPUTATIONAL PHYSICS. v. 230, p. 6169-6183, 2011.

POTTER, M. E.; NAUTA, M. D. FDTD on face-centered cubic (FCC) grids for Maxwell's equations. In: IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION. v. 61, n. 4, april 2013.

RANA, M.; MOHAN, A. S. Nonorthogonal LOD-FDTD method for EM scattering from two-dimensional structures. In: IEEE TRANSACTIONS ON ELECTROMAGNETIC COMPATIBILITY. v. 55, n. 4, august 2013.

RAPPAPORT, T. S. **Comunicações sem fio**: princípios e práticas. 2. ed. São Paulo: Pearson Prentice Hall, 2009.

REITZ, J. R.; MILFORD, F. J.; CHRISTY, R. W. **Fundamentos da teoria eletromagnética**. Rio de Janeiro: Campus, 1982.

ROCHE, G. D. L. et al. Combination of geometric and finite difference models for radio wave propagation in outdoor to indoor scenarios. In: ANTENNAS AND PROPAGATION (EUCAP), 2010 PROCEEDINGS OF THE FOURTH EUROPEAN CONFERENCE ON 2010.

RODEN, J.; GEDNEY, S. D. Convolution PML (CPML): an efficient FDTD implementation of the CFS-PML for arbitrary media. In: MICROWAVE AND OPT. TECH. LETT. v. 27, p. 334-339, 2000.

SAXENA, A. K.; SRIVASTAVA, K. V. Stability and dispersion analysis of higher order unconditionally stable three-step locally one-dimensional finite-difference time-domain method. In: IET MICROWAVES, ANTENNAS & PROPAGATION. v. 7, iss. 12, p. 954-960, 2013, doi:10.1049/iet-map.2013.0195.

SAXENA, A. K.; SRIVASTAVA, K. V. A three-dimensional unconditionally stable five-step LOD-FDTD method. In: IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION. v. 62, n. 3, march 2014.

SCHNEIDER, J. B.; KRUBLAK, R. J. Dispersion of homogeneous and inhomogeneous waves in the Yee finite-difference time-domain grid. In: IEEE TRANS. MICROW. THEORY AND TECHNIQUES. v. 49, n. 2, p. 280-287, 2001.

SCHNEIDER, J. B. **Understanding the finite-difference time-domain method**. 2013. Disponível em: <www.eecs.wsu.edu/~schneidj/ufdtd>. Acesso em: 28/03/2013.

SESCU, A.; HIXON, R. Numerical anisotropy study of a class of compact schemes.In: J. SCI COMPUT. v. 61, p. 327-342, 2014, doi: 10.1007/s10915-014-9826-0.

SHAMS, R.; SADEGHI, P. On optimization of finite-difference time-domain (FDTD) computation on heterogeneous and GPU clusters. In: JORNAL PARALLEL DISTRIB. COMPUT. 14 October 2010.

SHARP, J. **Microsoft Visual C# 2013**: passo a passo. Porto Alegre: Bookman, 2014.

SHLAGER, K. L.; SCHNEIDER, J. B. Comparison of the dispersion properties of several low-dispersion finite-difference time-domain algorithms. In: IEEE TRANS. ON ANTENNAS AND PROPAG. v. 51, n. 3, p. 642-653, 2003.

SMITH, W. S.; RAZMADZE, A.; SHAO, X. M.; DREWNIAK, J. L. A hierarchy of explicit low-dispersion FDTD methods for electrically large problems. In: IEEE TRANS. ON ANTENNAS AND PROPAG. v. 60, n. 12, p. 5787-5800, 2012.

SONG, H.; YANG, H. W. Analysis of numerical dispersion properties of SFDTD and MRTD schemes. In: OPTIK. v. 123, p. 272-275, 2012.

SU, Z.; YANG, Y.; TAN, J.; LONG, Y. A parameter-optimized symplectic FDTD method based on the (4, 4) stencil. In: IEEE ANTENNAS AND WIRELESS PROPAGATION LETTERS. v. 12, 2013.

TAFLOVE, A.; HAGNESS, S. C. **Computational electrodynamics**: the finite-difference time-domain method. 3rd ed. Boston: Artech House, 2005

THIEL, M.; SARABANDI, K. 3D-Wave propagation analysis of Indoor wireless channels utilizing hybrid methods. In: IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION. v. 57, n. 5, May 2009.

WANG, J.; ZHOU, B.; SHI, L.; GAO, C.; CHEN, B. A novel 3-D HIE-FDTD method with one-step leapfrog scheme. In: IEEE TRANSACTIONS ON MICROWAVE THEORY AND THECHNIQUES. v. 62, n. 6, june 2014.

YEE, K. S. Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. In: IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION. v.14, p. 302-30, 1966.

ZHANG, Q.; ZHOU, B. A novel HIE-FDTD method with large time-step size. In: IEEE ANTENNAS AND PROPAGATION MAGAZINE. v. 57, n. 2, april 2015.

ZHU, M.; CAO, Q.; ZHAO, L. Study and analysis of a novel Runge-Kutta high order finite-difference time-domain method. In: IET MICROWAVES, ANTENNAS & PROPAGATION. v. 8, iss. 12, p. 951-958, 2014, doi:10.1049/iet-map.2013.0650.

ZYGIRIDIS, T.; PYRIALAKOS, G.; KANTARTZIS, N.; TSIBOUKIS, T. Accelerated unconditionally stable FDTD scheme with modified operators. In: COMPEL: THE INTERNATIONAL JOURNAL FOR COMPUTATION AND MATHEMATICS IN ELECTRICAL AND ELECTRONIC ENGINEERING. v. 34, iss. 5, p. 1564-1577, 2015.

APÊNDICES

APÊNDICE 1. PROGRAMA C# PARA O MÉTODO FDTD COM GRADE DE PRISMAS HEXAGONAIS (SEM E COM PROCESSAMENTO PARALELO)

ESTE APÊNDICE CONTÉM 14 ARQUIVOS
ARQUIVO 1: Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Globalization;


namespace progCsharpe_Hex3d_matriz2d_float_1b
{
   class Program
   {
      static void Main(string[] args)
      {
         // INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS

         DateTime temp_inicio = DateTime.Now;

         int Nx = 13; // numero de pontos na direção x
         int Ny = 21;  // numero de pontos na direção y
         int Nz = 11; // numero de pontos na direção z
         int ppw = 15; // numero de pontos por comprimento de onda
         int tempomax = 3;

         int tipo_fonte = 1;  // 0=pulso ricker ou 1=senoide
         int HARD_SOFT = 1;  // fonte: 0=hard ou 1=soft
         int ATIVANDO_DIPOLO_1 = 0; // 1=ativa 0=desativa
         int ATIVANDO_DIPOLO_2 = 0;// 1=ativa 0=desativa

         float rdsdz = 1.0f;  // razão ds/dz
         float fator_comp = 1.4f; // em comprimentos de onda para dipolo_1

         string arqsaida1 = "ez_plano_xy1.txt";
         string arqsaida2 = "ez_plano_xz1.txt";

         int delta_passo = 10;
         float fator_cdtds = 2.0f;
         float fator_at = 0.5f;
         int fator_geometria = 1;
         float fa = 1.0f;
         float fb = 1.0f;
         float fc = 1.0f;
         int tempomax2 = 4;
```

```
        int tempomax3 = 5;
        // FINAL DAS VARIÁVEIS MAIS IMPORTANTES

        // ENTRADA DE VALORES PARA O ARQUIVO DE ENTRADA
        using (StreamReader entrada = new StreamReader(@"entrada.txt"))
        {
           // INICIO DA LEITURA DO ARQUIVO
           //CultureInfo.CreateSpecificCulture("en-US");
           Nx = int.Parse(entrada.ReadLine());
           Ny = int.Parse(entrada.ReadLine());
           Nz = int.Parse(entrada.ReadLine());
           ppw = int.Parse(entrada.ReadLine());
           tempomax = int.Parse(entrada.ReadLine());
           tipo_fonte = int.Parse(entrada.ReadLine());
           HARD_SOFT = int.Parse(entrada.ReadLine());
           ATIVANDO_DIPOLO_1 = int.Parse(entrada.ReadLine());
           ATIVANDO_DIPOLO_2 = int.Parse(entrada.ReadLine());
           rdsdz = float.Parse(entrada.ReadLine());
           fator_comp = float.Parse(entrada.ReadLine());
           arqsaida1 = entrada.ReadLine();
           arqsaida2 = entrada.ReadLine();
           delta_passo = int.Parse(entrada.ReadLine());
           fator_cdtds = float.Parse(entrada.ReadLine());
           fator_at = float.Parse(entrada.ReadLine());
           fator_geometria = int.Parse(entrada.ReadLine());
           fa = float.Parse(entrada.ReadLine());
           fb = float.Parse(entrada.ReadLine());
           fc = float.Parse(entrada.ReadLine());
           tempomax2 = int.Parse(entrada.ReadLine());
           tempomax3 = int.Parse(entrada.ReadLine());
        }// FINAL DA LEITURA DO ARQUIVO

        // VALIDANDO VALORES
        if (Nx % 2 == 0) Nx = Nx + 1; // Nx sempre impar para funcionar corretamente
        if (Ny % 2 == 0) Ny = Ny + 1;
        if (Nz % 2 == 0) Nz = Nz + 1;
        float cdtds;
        cdtds = (float)(1.0 / Math.Sqrt(fator_cdtds)); // número de Courant


        // MOSTRANDO ALGUNS VALORES NA TELA DO COMPUTADOR
        //CultureInfo.CreateSpecificCulture("en-US");

        Console.WriteLine("INICIO DO PROGRAMA FDTD ( VERSAO HEX 3D MATRIZ 2D:
float )\n");
        Console.WriteLine();
        Console.WriteLine("TEMPO INICIAL = {0}", temp_inicio);
        Console.WriteLine();
        Console.WriteLine("Nx = {0}", Nx);
        Console.WriteLine("Ny = {0}", Ny);
        Console.WriteLine("Nz = {0}", Nz);
        Console.WriteLine();
        Console.WriteLine("Pontos por comprimento de onda (ppw)   = {0}", ppw);
        Console.WriteLine("Tempo (numero de passos)               = {0}", tempomax);
        Console.WriteLine();
```

```
Console.WriteLine("Tipo de Sinal ( Pulso = 0 ou Senoide = 1 )  = {0}", tipo_fonte);
Console.WriteLine("Tipo de Fonte ( Hard  = 0 ou Soft    = 1 )  = {0}", HARD_SOFT);
Console.WriteLine();
Console.WriteLine("Numero de Courant: 1/raiz( {0} )  = {1}", fator_cdtds, cdtds);
Console.WriteLine("Razao ds / dz             = {0}", rdsdz);
Console.WriteLine();
Console.WriteLine("Fator ajuste ( fa )       = {0}", fa);
Console.WriteLine("Fator ajuste ( fb )       = {0}", fb);
Console.WriteLine("Fator ajuste ( fc )       = {0}", fc);
Console.WriteLine();
Console.WriteLine("Fator atenuacao (senoide)   = {0}", fator_at);
Console.WriteLine("Fator geometria             = {0}", fator_geometria);
Console.WriteLine();

// OUTRAS VARIÁVEIS INTERNAS DO PROGRAMA

const int ativo = 1; // tipo dipolo_1
const int passivo = 0;
int comprimento_dipolo;
comprimento_dipolo = (int)(fator_comp * rdsdz * ppw);

float ds, dx, dy, dz, Lx, Ly, Lz;
float imp0 = (float)(120.0 * Math.PI);// impedância característica no vácuo
float cha, chb, cea, ceb, chze2, cezh2;
ds = 1.0f; // dimensão dos lados do hexagono maior 2D
dx = (float)(ds * Math.Sqrt(3.0) / 2.0);// tamanho da célula na direção x
dy = ds / 2.0f;  // tamanho da célula na direção y
dz = ds / rdsdz;
Lx = (Nx) * dx; // comprimento na direção x
Ly = (Ny) * dy; // comprimento na direção y
Lz = (Nz) * dz; // comprimento na direção z

cha = cdtds / imp0;
chb = (float)(rdsdz * cdtds / (Math.Sqrt(3.0) * imp0));
cea = cdtds * imp0;
ceb = (float)(rdsdz * cdtds * imp0 / Math.Sqrt(3.0));
chze2 = 2.0f * cdtds / (3.0f * imp0);
cezh2 = 2.0f * cdtds * imp0 / 3.0f;
float HARD_SOFT2 = (float)HARD_SOFT;

// calculando dimensões das matrizes para os campos E e H
int ix, rx, jy, ry, jy2, ry2;
ix = Nx / 2;
rx = Nx % 2;
jy = Ny / 2;
ry = Ny % 2;
jy2 = (Ny - 1) / 2;
ry2 = (Ny - 1) % 2;

int PEZ, PH1, PH2, PH3;
PEZ = (ix + rx) * (jy + ry) + ix * jy;
PH2 = ix * (jy + ry) + (ix + rx) * jy;
PH1 = ix * (jy2 + ry2) + ix * jy2;
PH3 = PH1;
```

```
        Console.WriteLine("INICIALIZACAO - ALOCACAO DINAMICA DE MEMORIA:
MATRIZES 2D float)\n");
        // ALOCAÇÃO DINAMICA DE MEMÓRIA
        // ARRAY 2D
        //
        float[,] ez = new float[PEZ + 1, Nz + 1];
        float[,] e3 = new float[PH1 + 1, Nz + 1];
        float[,] h1 = new float[PH2 + 1, Nz + 1];
        float[,] e2 = new float[PH1 + 1, Nz + 1];
        float[,] h2 = new float[PH1 + 1, Nz + 1];
        float[,] e1 = new float[PH1 + 1, Nz + 1];
        float[,] h3 = new float[PH1 + 1, Nz + 1];
        float[,] hz = new float[PH1 + 1, Nz + 1];
        float[,] ez_pxy = new float[Nx + 1, Ny + 1];
        float[,] ez_pxz = new float[Nx + 1, Nz + 1];
        //
        // LISTA DE LISTAS
        /*
         Matrix ez = new Matrix(PEZ + 1, Nz + 1);
         Matrix e3 = new Matrix(PH1 + 1, Nz + 1);
         Matrix h1 = new Matrix(PH2 + 1, Nz + 1);
         Matrix e2 = new Matrix(PH1 + 1, Nz + 1);
         Matrix h2 = new Matrix(PH1 + 1, Nz + 1);
         Matrix e1 = new Matrix(PH1 + 1, Nz + 1);
         Matrix h3 = new Matrix(PH1 + 1, Nz + 1);
         Matrix hz = new Matrix(PH1 + 1, Nz + 1);
         Matrix ez_pxy = new Matrix(Nx + 1, Ny + 1);
         Matrix ez_pxz = new Matrix(Nx + 1, Nz + 1);
         */
        // Calculando memória total necessária
        long memoria = (PEZ + 1) * (Nz + 1) + 6 * (PH1 + 1) * (Nz + 1) + (PH2 + 1) * (Nz + 1)
+
           (Nx + 1) * (Ny + 1) + (Nx + 1) * (Nz + 1);
        memoria = memoria * sizeof(float);

        // posição da fonte (pulso ricker ou senóide)
        int px = Nx / 2 + 1;
        int py = Ny / 2 + 1;
        int pz = Nz / 2 + 1;
        int pcentro = Ax.p2b(px, py, Nx, Ny, PEZ);

        // INICIO DOS CÁLCULOS PARA GRADE HEXAGONAL 3D


        // inicializando grade 3d e dipolos (ainda não implementada

        Console.WriteLine("MEMÓRIA ALOCADA PARA CADA float: {0} BYTES",
sizeof(float));
        Console.WriteLine("MEMÓRIA ALOCADA PARA ARRAYS 2D:     {0} BYTES",
memoria);
        Console.WriteLine("FINAL INICIALIZACAO: hz[p,k] = {0}\n", hz[PH1, Nz]);
        Console.WriteLine("MATRIZES DE SAIDA: FLOAT SCIENTIFIC, MANTISSA 18
DIGITOS\n");

        // INICIO LOOP DO TEMPO
```

```
        for (int tempo = 0; tempo < tempomax; tempo++)
        {
            // TM3 e TE3 USA COMPUTAÇÃO PARALELA COM DUAS CPUs
            DateTime temp_inicio2 = DateTime.Now;
            /*
            TM.atualiza_tmz_h1(cha, chb, ez, e2, e3, h1, Nx, Ny, Nz, PH1, PH2, PH3, PEZ);
            TM.atualiza_tmz_h2(cha, chb, ez, e1, e3, h2, Nx, Ny, Nz, PH1, PEZ);
            TM.atualiza_tmz_h3(cha, chb, ez, e1, e2, h3, Nx, Ny, Nz, PH1, PH3, PEZ);
            //TM.atualiza_tmz_ez(  cezh2,  h1, h2, h3, ez, Nx, Ny, Nz, PH1, PH2, PH3, PEZ);
            TE.atualiza_tez_hz(chze2, e1, e2, e3, hz, Nx, Ny, Nz, PH1, PH3);

            TE.atualiza_tez_e1(cea, ceb, hz, h2, h3, e1, Nx, Ny, Nz, PH1, PH3);
            TE.atualiza_tez_e2(cea, ceb, hz, h1, h3, e2, Nx, Ny, Nz, PH2, PH3);
            TE.atualiza_tez_e3(cea, ceb, hz, h1, h2, e3, Nx, Ny, Nz, PH1, PH2, PH3);
            //TE.atualiza_tez_hz(  chze2,  e1, e2, e3, hz, Nx, Ny, Nz, PH1, PH3 );
            TM.atualiza_tmz_ez(cezh2, h1, h2, h3, ez, Nx, Ny, Nz, PH1, PH2, PH3, PEZ);
            */
            Task tmh1a = Task.Run(() => TM3.atualiza_tmz_h1(cha, chb, ez, e2, e3, h1, Nx, 1,
Ny/4, Ny, Nz, PH1, PH2, PH3, PEZ));
            Task tmh1b = Task.Run(() => TM3.atualiza_tmz_h1(cha, chb, ez, e2, e3, h1, Nx, Ny
/ 4, Ny / 2, Ny, Nz, PH1, PH2, PH3, PEZ));
            Task tmh1c = Task.Run(() => TM3.atualiza_tmz_h1(cha, chb, ez, e2, e3, h1, Nx, Ny
/ 2, Ny*3 / 4, Ny, Nz, PH1, PH2, PH3, PEZ));
            Task tmh1d = Task.Run(() => TM3.atualiza_tmz_h1(cha, chb, ez, e2, e3, h1, Nx, Ny
* 3 / 4, Ny + 1, Ny, Nz, PH1, PH2, PH3, PEZ));
            Task.WaitAll(tmh1a, tmh1b, tmh1c, tmh1d);
            Task tmh2a = Task.Run(() => TM3.atualiza_tmz_h2(cha, chb, ez, e1, e3, h2, Nx, 1,
Ny/4, Ny, Nz, PH1, PEZ));
            Task tmh2b = Task.Run(() => TM3.atualiza_tmz_h2(cha, chb, ez, e1, e3, h2, Nx, Ny
/ 4, Ny / 2, Ny, Nz, PH1, PEZ));
            Task tmh2c = Task.Run(() => TM3.atualiza_tmz_h2(cha, chb, ez, e1, e3, h2, Nx, Ny
/ 2, Ny * 3 / 4, Ny, Nz, PH1, PEZ));
            Task tmh2d = Task.Run(() => TM3.atualiza_tmz_h2(cha, chb, ez, e1, e3, h2, Nx, Ny
* 3 / 4, Ny, Ny, Nz, PH1, PEZ));
            Task.WaitAll(tmh2a, tmh2b, tmh2c, tmh2d);
            Task tmh3a = Task.Run(() => TM3.atualiza_tmz_h3(cha, chb, ez, e1, e2, h3, Nx, 1,
Ny/4, Ny, Nz, PH1, PH3, PEZ));
            Task tmh3b = Task.Run(() => TM3.atualiza_tmz_h3(cha, chb, ez, e1, e2, h3, Nx, Ny
/ 4, Ny / 2, Ny, Nz, PH1, PH3, PEZ));
            Task tmh3c = Task.Run(() => TM3.atualiza_tmz_h3(cha, chb, ez, e1, e2, h3, Nx, Ny
/ 2, Ny*3 / 4, Ny, Nz, PH1, PH3, PEZ));
            Task tmh3d = Task.Run(() => TM3.atualiza_tmz_h3(cha, chb, ez, e1, e2, h3, Nx, Ny
* 3 / 4, Ny, Ny, Nz, PH1, PH3, PEZ));
            Task.WaitAll(tmh3a, tmh3b, tmh3c, tmh3d);
            //Task tehza = Task.Run(() => TE3.atualiza_tez_hz(chze2, e1, e2, e3, hz, Nx,1,
Ny/4, Ny, Nz, PH1, PH3));
            //Task tehzb = Task.Run(() => TE3.atualiza_tez_hz(chze2, e1, e2, e3, hz, Nx, Ny /
4, Ny / 2, Ny, Nz, PH1, PH3));
            //Task tehzc = Task.Run(() => TE3.atualiza_tez_hz(chze2, e1, e2, e3, hz, Nx, Ny /
2, Ny*3 / 4, Ny, Nz, PH1, PH3));
            //Task tehzd = Task.Run(() => TE3.atualiza_tez_hz(chze2, e1, e2, e3, hz, Nx, Ny *
3 / 4, Ny + 1, Ny, Nz, PH1, PH3));
            //Task.WaitAll(tehza, tehzb, tehzc, tehzd);
```

```
            Task tee1a = Task.Run(() => TE3.atualiza_tez_e1(cea, ceb, hz, h2, h3, e1, Nx, 1,
Ny/4, Ny, Nz, PH1, PH3));
            Task tee1b = Task.Run(() => TE3.atualiza_tez_e1(cea, ceb, hz, h2, h3, e1, Nx, Ny
/ 4, Ny / 2, Ny, Nz, PH1, PH3));
            Task tee1c = Task.Run(() => TE3.atualiza_tez_e1(cea, ceb, hz, h2, h3, e1, Nx, Ny
/ 2, Ny*3 / 4, Ny, Nz, PH1, PH3));
            Task tee1d = Task.Run(() => TE3.atualiza_tez_e1(cea, ceb, hz, h2, h3, e1, Nx, Ny
* 3 / 4, Ny + 1, Ny, Nz, PH1, PH3));
            Task.WaitAll(tee1a, tee1b, tee1c, tee1d);
            Task tee2a = Task.Run(() => TE3.atualiza_tez_e2(cea, ceb, hz, h1, h3, e2, Nx, 1,
Ny/4, Ny, Nz, PH2, PH3));
            Task tee2b = Task.Run(() => TE3.atualiza_tez_e2(cea, ceb, hz, h1, h3, e2, Nx, Ny
/ 4, Ny / 2, Ny, Nz, PH2, PH3));
            Task tee2c = Task.Run(() => TE3.atualiza_tez_e2(cea, ceb, hz, h1, h3, e2, Nx, Ny
/ 2, Ny*3 / 4, Ny, Nz, PH2, PH3));
            Task tee2d = Task.Run(() => TE3.atualiza_tez_e2(cea, ceb, hz, h1, h3, e2, Nx, Ny
* 3 / 4, Ny, Ny, Nz, PH2, PH3));
            Task.WaitAll(tee2a, tee2b, tee2c, tee2d);
            Task tee3a = Task.Run(() => TE3.atualiza_tez_e3(cea, ceb, hz, h1, h2, e3, Nx, 1,
Ny/4, Ny, Nz, PH1, PH2, PH3));
            Task tee3b = Task.Run(() => TE3.atualiza_tez_e3(cea, ceb, hz, h1, h2, e3, Nx, Ny
/ 4, Ny / 2, Ny, Nz, PH1, PH2, PH3));
            Task tee3c = Task.Run(() => TE3.atualiza_tez_e3(cea, ceb, hz, h1, h2, e3, Nx, Ny
/ 2, Ny*3 / 4, Ny, Nz, PH1, PH2, PH3));
            Task tee3d = Task.Run(() => TE3.atualiza_tez_e3(cea, ceb, hz, h1, h2, e3, Nx, Ny
* 3 / 4, Ny, Ny, Nz, PH1, PH2, PH3));
            Task.WaitAll(tee3a, tee3b, tee3c, tee3d);
            Task tehza = Task.Run(() => TE3.atualiza_tez_hz(chze2, e1, e2, e3, hz, Nx, 1, Ny /
4, Ny, Nz, PH1, PH3));
            Task tehzb = Task.Run(() => TE3.atualiza_tez_hz(chze2, e1, e2, e3, hz, Nx, Ny / 4,
Ny / 2, Ny, Nz, PH1, PH3));
            Task tehzc = Task.Run(() => TE3.atualiza_tez_hz(chze2, e1, e2, e3, hz, Nx, Ny / 2,
Ny * 3 / 4, Ny, Nz, PH1, PH3));
            Task tehzd = Task.Run(() => TE3.atualiza_tez_hz(chze2, e1, e2, e3, hz, Nx, Ny * 3
/ 4, Ny + 1, Ny, Nz, PH1, PH3));
            Task.WaitAll(tehza, tehzb, tehzc, tehzd);
            Task tmeza = Task.Run(() => TM3.atualiza_tmz_ez(cezh2, h1, h2, h3, ez, Nx, 1,
Ny/4, Ny, Nz, PH1, PH2, PH3, PEZ));
            Task tmezb = Task.Run(() => TM3.atualiza_tmz_ez(cezh2, h1, h2, h3, ez, Nx, Ny /
4, Ny / 2, Ny, Nz, PH1, PH2, PH3, PEZ));
            Task tmezc = Task.Run(() => TM3.atualiza_tmz_ez(cezh2, h1, h2, h3, ez, Nx, Ny /
2, Ny*3 / 4, Ny, Nz, PH1, PH2, PH3, PEZ));
            Task tmezd = Task.Run(() => TM3.atualiza_tmz_ez(cezh2, h1, h2, h3, ez, Nx, Ny *
3 / 4, Ny + 1, Ny, Nz, PH1, PH2, PH3, PEZ));
            Task.WaitAll(tmeza, tmezb, tmezc, tmezd);

            ez[pcentro, pz] = HARD_SOFT2 * ez[pcentro, pz] + Ax.fonte_1(tempo, 0.0f, cdtds,
ppw, tipo_fonte, fator_at);

            //tempo_teste[tempo] = tempo;
            //ez_teste[tempo] = fonte_1(tempo, 0.0, cdtds, ppw, tipo_fonte, fator_at);

            if (tempo == tempomax2 - 1)
            {
               int nteste = tempomax2;
```

```
                string txx = nteste.ToString();
                string txsaida1 = @"ez_pxy_1_" + txx + ".txt";
                Saida.saida1(txsaida1, Nx, Ny, PEZ, pz, ez, ez_pxy);
                string txsaida2 = @"ez_pxz_2_" + txx + ".txt";
                Saida.saida2(txsaida2, Nx, Ny, Nz, PEZ, py, ez, ez_pxz);
            }
            if (tempo == tempomax3 - 1)
            {
                int nteste = tempomax3;
                string txx = nteste.ToString();
                string txsaida1 = @"ez_pxy_1_" + txx + ".txt";
                Saida.saida1(txsaida1, Nx, Ny, PEZ, pz, ez, ez_pxy);
                string txsaida2 = @"ez_pxz_2_" + txx + ".txt";
                Saida.saida2(txsaida2, Nx, Ny, Nz, PEZ, py, ez, ez_pxz);
            }

            DateTime temp_final2 = DateTime.Now;
            TimeSpan tempo_total2 = temp_final2 - temp_inicio2;
            if ((tempo + 1) % delta_passo == 0)
            {
                Console.WriteLine(" {0,8} / {1} em {2} ", tempo + 1, tempomax, tempo_total2);
            }
        } // FINAL LOOP DO TEMPO

        /* PÓS-PROCESSAMENTO DO CAMPO Ez 3D
        // para plano xy: campo ez
        for (int p = 1; p < PEZ + 1; p++)
        {
            int i, j;
            i = Ax.ez_i(p, Nx);
            j = Ax.ez_j(p, Nx);
            ez_pxy[i,j] = ez[p,pz];
        }

        // para plano xz: campo ez
        //int j = py;
        //int p;
        for (int i = 1; i < Nx + 1; i++)
            for (int k = 1; k < Nz + 1; k++)
                if ((py % 2 == 1 && i % 2 == 1 && i <= Nx) ||
                    (py % 2 == 0 && i % 2 == 0 && i <= Nx - 1)){
            int p = Ax.p2b(i, py, Nx, Ny, PEZ);
            ez_pxz[i,k] = ez[p,k];
            }

            // ESCREVENDO NO PRIMEIRO ARQUIVO TEXTO E LENDO DESTE ARQUIVO
- funcionou
                using (StreamWriter saidatx = new StreamWriter(@"ez_pxy_1.txt"))
                {
                for (int i = 0; i < Nx + 1; i++)
                {

                    for (int j = 0; j < Ny + 1; j++)
                    {
```

```
saidatx.WriteLine(ez_pxy[i,j].ToString("E18",CultureInfo.CreateSpecificCulture("en-US")));
                    }

                }
            } // FINAL DA ESCRITA NO ARQUIVO TEXTO

            // ESCREVENDO NO SEGUNDO ARQUIVO TEXTO E LENDO DESTE ARQUIVO
- funcionou
                using (StreamWriter saidatx = new StreamWriter(@"ez_pxz_2.txt"))
                {
                for (int i = 0; i < Nx + 1; i++)
                {

                    for (int j = 0; j < Nz + 1; j++)
                    {


saidatx.WriteLine(ez_pxz[i,j].ToString("E18",CultureInfo.CreateSpecificCulture("en-US")));
                    }

                }
            } // FINAL DA ESCRITA NO ARQUIVO TEXTO
            */
          //METODOS PARA ESCRITA NOS DOIS ARQUIVOS DE SAÍDA
          int nteste2 = tempomax;
          string txx2 = nteste2.ToString();
          string txsaida1a = @"ez_pxy_1_" + txx2 + ".txt";
          Saida.saida1(txsaida1a, Nx, Ny, PEZ, pz, ez, ez_pxy);
          string txsaida2a = @"ez_pxz_2_" + txx2 + ".txt";
          Saida.saida2(txsaida2a, Nx, Ny, Nz, PEZ, py, ez, ez_pxz);

          // MEDINDO TEMPO DE EXECUÇÃO DO PROGRAMA
          DateTime temp_final = DateTime.Now;
          TimeSpan tempo_total = temp_final - temp_inicio;
          Console.WriteLine("\nTEMPO INICIAL: {0}", temp_inicio);
          Console.WriteLine("\nTEMPO FINAL:   {0}", temp_final);
          Console.WriteLine("\nTEMPO TOTAL (HH : MM : SS): {0}", tempo_total);
          Console.Write("\nENTRE COM UM NÚMERO OU LETRA PARA SAIR DO
PROGRAMA: ");
          string FINAL = Console.ReadLine();
          // FINAL DO PROGRAMA
      }
   }
}
////////////////////////////////////////////////////////////////////////////////
```

ARQUIVO 2: Ax.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace progCsharpe_Hex3d_matriz2d_float_1b
{
    class Ax
    {            // FUNÇÕES DE POSIÇÃO
        // Função para hz e h3
        public static int p2a(int i, int j, int Nx, int Ny, int PH3)
        {
            int p1, p2;

            if ((i <= 0) || (i >= Nx + 1) || (j <= 0) || (j >= Ny + 1))
            {
                p2 = 0;
            }
            else
            {
                p1 = i + (j - 1) * (Nx - 1);
                p2 = (p1 + 1) / 2;
                if (p2 > PH3)
                    p2 = 0;
            }
            return p2;
        }

        // Função para ez e e3
        public static int p2b(int i, int j, int Nx, int Ny, int PEZ)
        {
            int p1, p2;
            if ((i <= 0) || (i >= Nx + 1) || (j <= 0) || (j >= Ny + 1))
            {
                p2 = 0;
            }
            else
            {
                p1 = i + (j - 1) * (Nx);
                p2 = (p1 + 1) / 2;
                if (p2 > PEZ)
                    p2 = 0;
            }
            return p2;
        }

        // Função para h2 e e1
        public static int p2c(int i, int j, int Nx, int Ny, int PH2)
        {
            int p1, p2;
            if ((i <= 0) || (i >= Nx + 1) || (j <= 0) || (j >= Ny + 1))
            {
                p2 = 0;
            }
            else
            {
                p1 = i + (j - 1) * (Nx);
                p2 = (p1) / 2;
                if (p2 > PH2)
                    p2 = 0;
```

```
        }
        return p2;
    }

    // Função para h1 e e2
    public static int p2d(int i, int j, int Nx, int Ny, int PH1)
    {
        int p1, p2;
        if ((i <= 0) || (i >= Nx + 1) || (j <= 0) || (j >= Ny + 1))
        {
            p2 = 0;
        }
        else
        {
            p1 = i + (j - 1) * (Nx - 1);
            p2 = (p1 + 1) / 2;
            if (p2 > PH1)
                p2 = 0;
        }
        return p2;
    }

    // FUNÇÃO PARA FONTE DO TIPO PULSO RICKER OU SENOIDE

    public static float fonte_1(int tempo1, float posicao, float cdtds1, int ppw, int tipo_fonte,
float fator_at)
    {
        double arg, arg2, funcao, periodo, cte_tempo, atenuacao;
        double pi = Math.PI;
        double tempo2 = (double)tempo1;
        double ppw2 = (double)ppw;

        if (tipo_fonte == 0)
        { // pulso ricker
            arg = pi * ((cdtds1 * tempo2 - posicao) / ppw2 - 1.0);
            arg2 = arg * arg;
            funcao = (1.0 - 2.0 * arg2) * Math.Exp(-arg2);
        }

        else
        {  // senóide com amortecimento inicial
            periodo = ppw2 / cdtds1; // periodo da senóide
            // ou tempo para o pico do pulso ricker
            cte_tempo = fator_at * periodo;
            atenuacao = 1.0 - Math.Exp(-1.0 * tempo2 / cte_tempo);
            funcao = atenuacao * Math.Sin(2.0 * pi * tempo2 * cdtds1 / ppw2);
        }
        return (float)funcao;
    }
    //**********************************

    // funções para obter i,j a partir de p para campo ez
    public static int ez_i(int p2, int Nx)
    {
        int p1, i, j;
```

```
            p1 = 2 * p2 - 1;
            j = p1 / Nx + 1;
            if ((p1 % Nx == 0) && (p1 >= Nx))
                j = j - 1;
            i = p1 - (j - 1) * Nx;
            return i;
        }

        public static int ez_j(int p2, int Nx)
        {
            int p1, i, j;
            p1 = 2 * p2 - 1;
            j = p1 / Nx + 1;
            if ((p1 % Nx == 0) && (p1 >= Nx))
                j = j - 1;
            i = p1 - (j - 1) * Nx;
            return j;
        }

        //FINAL DOS MÉTODOS
    }
}
//////////////////////////////////////////////////////
```

ARQUIVO 3: TE.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace progCsharpe_Hex3d_matriz2d_float_1b
{
    class TE
    {
        // ATUALIZANDO MODO TEz

        public static void atualiza_tez_e1(float cea, float ceb,
            float[,] hz, float[,] h2, float[,] h3,
            float[,] e1, int Nx, int Ny, int Nz, int PH1, int PH3)
        {
            int p, s, n, nw, sw;
            // MODO TEz
            // campo e1
            for (int j = 1; j < Ny + 1; j++)
                for (int i = 1; i < Nx + 1; i++)
                    for (int k = 1; k < Nz; k++)
                        if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
                            (j % 2 == 0 && i % 2 == 1 && i <= Nx - 2))
                        {
                            p = Ax.p2d(i, j, Nx, Ny, PH1);

                            s = Ax.p2a(i, j - 1, Nx, Ny, PH3);
                            n = Ax.p2a(i, j + 1, Nx, Ny, PH3);
```

```
                nw = Ax.p2d(i, j, Nx, Ny, PH1);
                sw = Ax.p2a(i, j - 1, Nx, Ny, PH3);
                e1[p, k] = e1[p, k] -
                    cea * (hz[s, k] - hz[n, k]) -  // sem troca de sinal
                    ceb * (h2[nw, k + 1] + h3[sw, k + 1] -
                    h2[nw, k] - h3[sw, k]);
            }
}

public static void atualiza_tez_e2(float cea, float ceb,
    float[,] hz, float[,] h1, float[,] h3,
    float[,] e2, int Nx, int Ny, int Nz, int PH2, int PH3)
{
    int p, se, nw, e, nw2;

    // campo e2
    for (int j = 1; j < Ny; j++)
        for (int i = 1; i < Nx + 1; i++)
            for (int k = 1; k < Nz; k++)
                if ((j % 2 == 1 && i % 2 == 1 && i <= Nx - 2) ||
                    (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
                {
                    p = Ax.p2a(i, j, Nx, Ny, PH3);

                    se = Ax.p2a(i, j, Nx, Ny, PH3);
                    nw = Ax.p2a(i - 1, j + 1, Nx, Ny, PH3);
                    e = Ax.p2a(i, j, Nx, Ny, PH3);
                    nw2 = Ax.p2c(i, j + 1, Nx, Ny, PH2);
                    e2[p, k] = e2[p, k] -
                        cea * (hz[se, k] - hz[nw, k]) -
                        ceb * (h3[e, k + 1] - h1[nw2, k + 1] -
                        h3[e, k] + h1[nw2, k]);
                }
}


public static void atualiza_tez_e3(float cea, float ceb,
    float[,] hz, float[,] h1, float[,] h2,
    float[,] e3, int Nx, int Ny, int Nz, int PH1, int PH2, int PH3)
{
    int p, ne, sw, e, sw2;

    // campo e3
    for (int j = 1; j < Ny; j++)
        for (int i = 1; i < Nx + 1; i++)
            for (int k = 1; k < Nz; k++)
                if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
                    (j % 2 == 0 && i % 2 == 1 && i <= Nx - 2))
                {
                    p = Ax.p2d(i, j, Nx, Ny, PH1);

                    ne = Ax.p2a(i, j + 1, Nx, Ny, PH3);
                    sw = Ax.p2a(i - 1, j, Nx, Ny, PH3);
                    e = Ax.p2d(i, j, Nx, Ny, PH1);
                    sw2 = Ax.p2c(i, j, Nx, Ny, PH2);
```

```
                    e3[p, k] = e3[p, k] -
                        cea * (hz[ne, k] - hz[sw, k]) -  // sem troca de sinal
                        ceb * (-1.0f * h2[e, k + 1] - h1[sw2, k + 1] +
                        h2[e, k] + h1[sw2, k]);
                }
        }


        public static void atualiza_tez_hz(float chze2,
            float[,] e1, float[,] e2, float[,] e3,
            float[,] hz, int Nx, int Ny, int Nz, int PH1, int PH3)
        {
            int p, s, n, se, nw, ne, sw;

            // campo hz
            for (int j = 1; j < Ny + 1; j++)
                for (int i = 1; i < Nx + 1; i++)
                    for (int k = 1; k < Nz + 1; k++) // verificar ??
                        if ((j % 2 == 1 && i % 2 == 1 && i <= Nx - 2) ||
                            (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
                        {
                            p = Ax.p2a(i, j, Nx, Ny, PH3);

                            s = Ax.p2d(i, j - 1, Nx, Ny, PH1);
                            n = Ax.p2d(i, j + 1, Nx, Ny, PH1);
                            se = Ax.p2a(i + 1, j - 1, Nx, Ny, PH3);
                            nw = Ax.p2a(i, j, Nx, Ny, PH3);
                            ne = Ax.p2d(i + 1, j, Nx, Ny, PH1);
                            sw = Ax.p2d(i, j - 1, Nx, Ny, PH1);
                            hz[p, k] = hz[p, k] -
                                chze2 * (e1[s, k] + e2[se, k] + e3[ne, k] -
                                e1[n, k] - e2[nw, k] - e3[sw, k]);
                        }
        }

        // FINAL DOS MÉTODOS
    }
}

//////////////////////////////////////////////////

ARQUIVO 4: TM.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace progCsharpe_Hex3d_matriz2d_float_1b
{
    class TM
    {
        // ATUALIZANDO MODO TMz
```

```
public static void atualiza_tmz_h1(float cha, float chb,
    float[,] ez, float[,] e2, float[,] e3,
   float[,] h1, int Nx, int Ny, int Nz, int PH1, int PH2, int PH3, int PEZ)
{
   int p, s, n, se, ne;
   // MODO TMz
   // campo h1
   for (int j = 1; j < Ny + 1; j++)
      for (int i = 1; i < Nx + 1; i++)
         for (int k = 1; k < Nz + 1; k++)
            if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
               (j % 2 == 0 && i % 2 == 1 && i <= Nx))
            {
               p = Ax.p2c(i, j, Nx, Ny, PH2);

               s = Ax.p2b(i, j - 1, Nx, Ny, PEZ);
               n = Ax.p2b(i, j + 1, Nx, Ny, PEZ);
               se = Ax.p2a(i, j - 1, Nx, Ny, PH3);
               ne = Ax.p2d(i, j, Nx, Ny, PH1);
               h1[p, k] = h1[p, k] +
                  cha * (ez[s, k] - ez[n, k]) +
                  chb * (e2[se, k] + e3[ne, k] -
                  e2[se, k - 1] - e3[ne, k - 1]);
            }
}


public static void atualiza_tmz_h2(float cha, float chb,
    float[,] ez, float[,] e1, float[,] e3,
   float[,] h2, int Nx, int Ny, int Nz, int PH1, int PEZ)
{
   int p, se, nw, se2, w;

   // campo h2
   for (int j = 1; j < Ny; j++)
      for (int i = 1; i < Nx + 1; i++)
         for (int k = 1; k < Nz + 1; k++)
            if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
               (j % 2 == 0 && i % 2 == 1 && i <= Nx - 2))
            {
               p = Ax.p2d(i, j, Nx, Ny, PH1);

               se = Ax.p2b(i + 1, j, Nx, Ny, PEZ);
               nw = Ax.p2b(i, j + 1, Nx, Ny, PEZ);
               se2 = Ax.p2d(i, j, Nx, Ny, PH1);
               w = Ax.p2d(i, j, Nx, Ny, PH1);

               h2[p, k] = h2[p, k] +
                  cha * (ez[se, k] - ez[nw, k]) + // sem troca de sinal
                  chb * (e3[w, k] - e1[se2, k] -
                  e3[w, k - 1] + e1[se2, k - 1]);
            }
}

public static void atualiza_tmz_h3(float cha, float chb,
```

```
            float[,] ez, float[,] e1, float[,] e2,
            float[,] h3, int Nx, int Ny, int Nz, int PH1, int PH3, int PEZ)
        {
            int p, ne, sw, ne2, w;

            // campo h3
            for (int j = 1; j < Ny; j++)
                for (int i = 1; i < Nx + 1; i++)
                    for (int k = 1; k < Nz + 1; k++)
                        if ((j % 2 == 1 && i % 2 == 1 && i <= Nx - 2) ||
                            (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
                        {
                            p = Ax.p2a(i, j, Nx, Ny, PH3);

                            ne = Ax.p2b(i + 1, j + 1, Nx, Ny, PEZ);
                            sw = Ax.p2b(i, j, Nx, Ny, PEZ);
                            ne2 = Ax.p2d(i, j + 1, Nx, Ny, PH1);
                            w = Ax.p2a(i, j, Nx, Ny, PH3);
                            h3[p, k] = h3[p, k] +
                                cha * (ez[ne, k] - ez[sw, k]) +   //erro  sinal  corrigido
                                chb * (-1.0f * e1[ne2, k] - e2[w, k] +
                                e1[ne2, k - 1] + e2[w, k - 1]);
                        }
        }

        public static void atualiza_tmz_ez(float cezh2,
            float[,] h1, float[,] h2, float[,] h3,
            float[,] ez, int Nx, int Ny, int Nz, int PH1, int PH2, int PH3, int PEZ)
        {
            int p, s, n, se, nw, ne, sw;

            // campo ez
            for (int j = 1; j < Ny + 1; j++)
                for (int i = 1; i < Nx + 1; i++)
                    for (int k = 1; k < Nz + 1; k++)
                        if ((j % 2 == 1 && i % 2 == 1 && i <= Nx) ||
                            (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
                        {
                            p = Ax.p2b(i, j, Nx, Ny, PEZ);

                            s = Ax.p2c(i, j - 1, Nx, Ny, PH2);
                            n = Ax.p2c(i, j + 1, Nx, Ny, PH2);
                            se = Ax.p2d(i, j - 1, Nx, Ny, PH1);
                            nw = Ax.p2d(i - 1, j, Nx, Ny, PH1);
                            ne = Ax.p2a(i, j, Nx, Ny, PH3);
                            sw = Ax.p2a(i - 1, j - 1, Nx, Ny, PH3);
                            ez[p, k] = ez[p, k] +
                                cezh2 * (h1[s, k] + h2[se, k] + h3[ne, k] -
                                h1[n, k] - h2[nw, k] - h3[sw, k]);
                        }
        }

        // FINAL DOS METODOS
    }
}
```

//

ARQUIVO 5: TE3.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace progCsharpe_Hex3d_matriz2d_float_1b
{
   class TE3
   {
      // ATUALIZANDO MODO TEz COM PROCESSAMENTO PARALELO

      public static void atualiza_tez_e1(float cea, float ceb,
         float[,] hz, float[,] h2, float[,] h3,
         float[,] e1, int Nx, int jinicial, int jfinal, int Ny, int Nz, int PH1, int PH3)
      {
         int p, s, n, nw, sw;
         // MODO TEz
         // campo e1
         for (int j = jinicial; j < jfinal; j++)
            for (int i = 1; i < Nx + 1; i++)
               for (int k = 1; k < Nz; k++)
                  if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
                     (j % 2 == 0 && i % 2 == 1 && i <= Nx - 2))
                  {
                     p = Ax.p2d(i, j, Nx, Ny, PH1);

                     s = Ax.p2a(i, j - 1, Nx, Ny, PH3);
                     n = Ax.p2a(i, j + 1, Nx, Ny, PH3);
                     nw = Ax.p2d(i, j, Nx, Ny, PH1);
                     sw = Ax.p2a(i, j - 1, Nx, Ny, PH3);
                     e1[p, k] = e1[p, k] -
                        cea * (hz[s, k] - hz[n, k]) -  // sem troca de sinal
                        ceb * (h2[nw, k + 1] + h3[sw, k + 1] -
                        h2[nw, k] - h3[sw, k]);
                  }
      }

      public static void atualiza_tez_e2(float cea, float ceb,
         float[,] hz, float[,] h1, float[,] h3,
         float[,] e2, int Nx, int jinicial, int jfinal, int Ny, int Nz, int PH2, int PH3)
      {
         int p, se, nw, e, nw2;

         // campo e2
         for (int j = jinicial; j < jfinal; j++)
            for (int i = 1; i < Nx + 1; i++)
               for (int k = 1; k < Nz; k++)
                  if ((j % 2 == 1 && i % 2 == 1 && i <= Nx - 2) ||
                     (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
                  {
```

```
                p = Ax.p2a(i, j, Nx, Ny, PH3);

                se = Ax.p2a(i, j, Nx, Ny, PH3);
                nw = Ax.p2a(i - 1, j + 1, Nx, Ny, PH3);
                e = Ax.p2a(i, j, Nx, Ny, PH3);
                nw2 = Ax.p2c(i, j + 1, Nx, Ny, PH2);
                e2[p, k] = e2[p, k] -
                    cea * (hz[se, k] - hz[nw, k]) -
                    ceb * (h3[e, k + 1] - h1[nw2, k + 1] -
                    h3[e, k] + h1[nw2, k]);
            }
}


public static void atualiza_tez_e3(float cea, float ceb,
    float[,] hz, float[,] h1, float[,] h2,
    float[,] e3, int Nx, int jinicial, int jfinal, int Ny, int Nz, int PH1, int PH2, int PH3)
{
    int p, ne, sw, e, sw2;

    // campo e3
    for (int j = jinicial; j < jfinal; j++)
        for (int i = 1; i < Nx + 1; i++)
            for (int k = 1; k < Nz; k++)
                if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
                    (j % 2 == 0 && i % 2 == 1 && i <= Nx - 2))
                {
                    p = Ax.p2d(i, j, Nx, Ny, PH1);

                    ne = Ax.p2a(i, j + 1, Nx, Ny, PH3);
                    sw = Ax.p2a(i - 1, j, Nx, Ny, PH3);
                    e = Ax.p2d(i, j, Nx, Ny, PH1);
                    sw2 = Ax.p2c(i, j, Nx, Ny, PH2);
                    e3[p, k] = e3[p, k] -
                        cea * (hz[ne, k] - hz[sw, k]) -  // sem troca de sinal
                        ceb * (-1.0f * h2[e, k + 1] - h1[sw2, k + 1] +
                        h2[e, k] + h1[sw2, k]);
                }
}


public static void atualiza_tez_hz(float chze2,
    float[,] e1, float[,] e2, float[,] e3,
    float[,] hz, int Nx, int jinicial, int jfinal, int Ny, int Nz, int PH1, int PH3)
{
    int p, s, n, se, nw, ne, sw;

    // campo hz
    for (int j = jinicial; j < jfinal; j++)
        for (int i = 1; i < Nx + 1; i++)
            for (int k = 1; k < Nz + 1; k++) // verificar ??
                if ((j % 2 == 1 && i % 2 == 1 && i <= Nx - 2) ||
                    (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
                {
                    p = Ax.p2a(i, j, Nx, Ny, PH3);
```

```
                    s = Ax.p2d(i, j - 1, Nx, Ny, PH1);
                    n = Ax.p2d(i, j + 1, Nx, Ny, PH1);
                    se = Ax.p2a(i + 1, j - 1, Nx, Ny, PH3);
                    nw = Ax.p2a(i, j, Nx, Ny, PH3);
                    ne = Ax.p2d(i + 1, j, Nx, Ny, PH1);
                    sw = Ax.p2d(i, j - 1, Nx, Ny, PH1);
                    hz[p, k] = hz[p, k] -
                        chze2 * (e1[s, k] + e2[se, k] + e3[ne, k] -
                        e1[n, k] - e2[nw, k] - e3[sw, k]);
                }
        }

        // FINAL DOS MÉTODOS
    }
}
////////////////////////////////////////////////////////////////////////////
```

ARQUIVO 6: TM3.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace progCsharpe_Hex3d_matriz2d_float_1b
{
    class TM3
    {
        // ATUALIZANDO MODO TMz COM PROCESSAMENTO PARALELO

        public static void atualiza_tmz_h1(float cha, float chb,
             float[,] ez, float[,] e2, float[,] e3,
            float[,] h1, int Nx, int jinicial, int jfinal, int Ny, int Nz, int PH1, int PH2, int PH3, int PEZ)
        {
            int p, s, n, se, ne;
            // MODO TMz
            // campo h1
            for (int j = jinicial; j < jfinal; j++)
                for (int i = 1; i < Nx + 1; i++)
                    for (int k = 1; k < Nz + 1; k++)
                        if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
                            (j % 2 == 0 && i % 2 == 1 && i <= Nx))
                        {
                            p = Ax.p2c(i, j, Nx, Ny, PH2);

                            s = Ax.p2b(i, j - 1, Nx, Ny, PEZ);
                            n = Ax.p2b(i, j + 1, Nx, Ny, PEZ);
                            se = Ax.p2a(i, j - 1, Nx, Ny, PH3);
                            ne = Ax.p2d(i, j, Nx, Ny, PH1);
                            h1[p, k] = h1[p, k] +
                                cha * (ez[s, k] - ez[n, k]) +
                                chb * (e2[se, k] + e3[ne, k] -
                                e2[se, k - 1] - e3[ne, k - 1]);
```

```
            }
}


public static void atualiza_tmz_h2(float cha, float chb,
    float[,] ez, float[,] e1, float[,] e3,
   float[,] h2, int Nx, int jinicial, int jfinal, int Ny, int Nz, int PH1, int PEZ)
{
   int p, se, nw, se2, w;

   // campo h2
   for (int j = jinicial; j < jfinal; j++)
      for (int i = 1; i < Nx + 1; i++)
         for (int k = 1; k < Nz + 1; k++)
            if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
               (j % 2 == 0 && i % 2 == 1 && i <= Nx - 2))
            {
               p = Ax.p2d(i, j, Nx, Ny, PH1);

               se = Ax.p2b(i + 1, j, Nx, Ny, PEZ);
               nw = Ax.p2b(i, j + 1, Nx, Ny, PEZ);
               se2 = Ax.p2d(i, j, Nx, Ny, PH1);
               w = Ax.p2d(i, j, Nx, Ny, PH1);

               h2[p, k] = h2[p, k] +
                  cha * (ez[se, k] - ez[nw, k]) + // sem troca de sinal
                  chb * (e3[w, k] - e1[se2, k] -
                  e3[w, k - 1] + e1[se2, k - 1]);
            }
}

public static void atualiza_tmz_h3(float cha, float chb,
    float[,] ez, float[,] e1, float[,] e2,
   float[,] h3, int Nx, int jinicial, int jfinal, int Ny, int Nz, int PH1, int PH3, int PEZ)
{
   int p, ne, sw, ne2, w;

   // campo h3
   for (int j = jinicial; j < jfinal; j++)
      for (int i = 1; i < Nx + 1; i++)
         for (int k = 1; k < Nz + 1; k++)
            if ((j % 2 == 1 && i % 2 == 1 && i <= Nx - 2) ||
               (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
            {
               p = Ax.p2a(i, j, Nx, Ny, PH3);

               ne = Ax.p2b(i + 1, j + 1, Nx, Ny, PEZ);
               sw = Ax.p2b(i, j, Nx, Ny, PEZ);
               ne2 = Ax.p2d(i, j + 1, Nx, Ny, PH1);
               w = Ax.p2a(i, j, Nx, Ny, PH3);
               h3[p, k] = h3[p, k] +
                  cha * (ez[ne, k] - ez[sw, k]) +   //erro  sinal  corrigido
                  chb * (-1.0f * e1[ne2, k] - e2[w, k] +
                  e1[ne2, k - 1] + e2[w, k - 1]);
            }
```

```
        }

        public static void atualiza_tmz_ez(float cezh2,
             float[,] h1, float[,] h2, float[,] h3,
            float[,] ez, int Nx, int jinicial, int jfinal, int Ny, int Nz, int PH1, int PH2, int PH3, int PEZ)
        {
            int p, s, n, se, nw, ne, sw;

            // campo ez
            for (int j = jinicial; j < jfinal; j++)
                for (int i = 1; i < Nx + 1; i++)
                    for (int k = 1; k < Nz + 1; k++)
                        if ((j % 2 == 1 && i % 2 == 1 && i <= Nx) ||
                            (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
                        {
                            p = Ax.p2b(i, j, Nx, Ny, PEZ);

                            s = Ax.p2c(i, j - 1, Nx, Ny, PH2);
                            n = Ax.p2c(i, j + 1, Nx, Ny, PH2);
                            se = Ax.p2d(i, j - 1, Nx, Ny, PH1);
                            nw = Ax.p2d(i - 1, j, Nx, Ny, PH1);
                            ne = Ax.p2a(i, j, Nx, Ny, PH3);
                            sw = Ax.p2a(i - 1, j - 1, Nx, Ny, PH3);
                            ez[p, k] = ez[p, k] +
                                cezh2 * (h1[s, k] + h2[se, k] + h3[ne, k] -
                                h1[n, k] - h2[nw, k] - h3[sw, k]);
                        }
        }

        // FINAL DOS METODOS
    }
}
////////////////////////////////////////////////////////////////////////////

ARQUIVO 7: Saida.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Globalization;

namespace progCsharpe_Hex3d_matriz2d_float_1b
{
    class Saida
    {
        public static void saida1(string txsaida1, int Nx, int Ny, int PEZ, int pz, float[,] ez, float[,]
ez_pxy)
        {
            // PÓS-PROCESSAMENTO DO CAMPO Ez 3D:para plano xy: campo ez
            for (int p = 1; p < PEZ + 1; p++)
            {
                int i, j;
```

```
            i = Ax.ez_i(p, Nx);
            j = Ax.ez_j(p, Nx);
            ez_pxy[i, j] = ez[p, pz];
         }
         // ESCREVENDO NO PRIMEIRO ARQUIVO TEXTO E LENDO DESTE ARQUIVO -
funcionou
         using (StreamWriter saida_1 = new StreamWriter(txsaida1))
         {
            for (int i = 0; i < Nx + 1; i++)
            {

               for (int j = 0; j < Ny + 1; j++)
               {

                  saida_1.WriteLine(ez_pxy[i,j].ToString("E18",
CultureInfo.CreateSpecificCulture("en-US")));
               }

            }
         } // FINAL DA ESCRITA NO ARQUIVO TEXTO
      } // FINAL METODO SAIDA1

      public static void saida2(string txsaida2, int Nx, int Ny, int Nz, int PEZ, int py, float[,] ez,
float[,] ez_pxz)
      {
         // para plano xz: campo ez

         for (int i = 1; i < Nx + 1; i++)
            for (int k = 1; k < Nz + 1; k++)
               if ((py % 2 == 1 && i % 2 == 1 && i <= Nx) ||
                  (py % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
               {
                  int p = Ax.p2b(i, py, Nx, Ny, PEZ);
                  ez_pxz[i, k] = ez[p, k];
               }
         // ESCREVENDO NO SEGUNDO ARQUIVO TEXTO E LENDO DESTE ARQUIVO -
funcionou
         using (StreamWriter saida_2 = new StreamWriter(txsaida2))
         {
            for (int i = 0; i < Nx + 1; i++)
            {

               for (int j = 0; j < Nz + 1; j++)
               {

                  saida_2.WriteLine(ez_pxz[i,j].ToString("E18",
CultureInfo.CreateSpecificCulture("en-US")));
               }

            }
         } // FINAL DA ESCRITA NO ARQUIVO TEXTO
      } // FINAL METODO SAIDA2
   }
}
// FINAL APENDICE 1 ////////////////////////////////////////////////////////////////////////////////////////////
```

APÊNDICE 2. PROGRAMA C# PARA O MÉTODO FDTD COM GRADE DE PRISMAS HEXAGONAIS (COM PMLs E REGIÕES DE CAMPOS TOTAL E ESPALHADO)

ESTE APÊNDICE CONTÉM 14 ARQUIVOS

ARQUIVO 1: entrada3.txt

```
349                 Nx ( estes comentários deve ser apagados para usar este arquivo)
601                  Ny
349                 Nz
10                   ppw (número de pontos por comprimento de onda)
631                 tempomax
1                   tipo_fonte (0 = pulso, 1 = senoide)
1                   HARD_SOFT ( 0 = hard, 1 = soft)
1.0                  ds (tamanho do lado do hexágono maior)
1.1547              rdsdz = R = ds/dz
3.3334               fator_cdtds -> S_CFL = 1 / fator_cdtds^1/2
0.5                  fator_at
1                    delta_passo
678                 tempomax2
679                 tempomax3
1                    CPML ( 0 = desativa, 1 = ativa )
20                  nxx ( nyy = 3*nxx +2) define espessuras PMLs: nxx*dx; nyy*dy; nxx*dz
1.0e-6                Reflexao
1.0e-6                Reflexao2
1.9                   me
3.9                   me2
3                     numero_materiais ( >= 1 )
3                      numero_objetos ( >= 0 )
2                      xpml ( 0 = sem saída medidas de reflexão nas PMLs)
3                        ypml (junto com xpml define posições de medida de reflexão)
1                     TFSF ( 0 = desativa, 1 = ativa )
14                     deltax (define inicio região TFST: nxx + deltax)
15                    deltay (define inicio região TFST: nyy + deltay)
450  (DISP=0  sem compensação; DISP=1 comp. simples; DISP=450 comp. otimizada)
1.00456               disp_cte (usada para DISP=1 compensação simples)
//////////////////////////////////////////
```

ARQUIVO 2: materiais.txt (estes comentários deve ser apagados para usar este arquivo)

```
0.0          condutividade elétrica (material = 0 espaço livre ou vácuo)
1.0           permissividade elétrica relativa
0.0           condutividade magnética
1.0          permeabilidade magnética relativa
6.0e14       idem material = 1 (condutor elétrico perfeito)
1.0
0.0
1.0
2.0e -3      idem material = 2 (dielétrico com perdas elétricas)
3.2
0.0
1.0

//////////////////////////////////////
```

ARQUIVO 3: objetos.txt (estes comentários deve ser apagados para usar este arquivo)
```
1         placa retangular = 1 ( se 0 este objeto não é usado)
1        material = 1 (condutor elétrico perfeito)
40        posição x (em metros) do canto esquerdo inferior no plano x-y
45       posição y (em metros) do canto esquerdo inferior no plano x-y
50     posição z (em metros) do canto esquerdo inferior no plano x-z
20     largura x da placa retangular (somada a posição x)
21    largura y da placa retangular (somada a posição y)
22    largura z da placa retangular (somada a posição z)
2          esfera = 2
1        material = 1 (condutor elétrico perfeito)
60       posição x (em metros) do centro da esfera no plano x-y
65       posição y (em metros) do centro da esfera no plano x-y
70      posição z (em metros) do centro da esfera no plano x-z
25       raio (em metros) da esfera
0.0     não usado
0.0     não usado
3         cilindro = 3
2         material = 2 (dielétrico com perdas elétricas)
80       posição x (em metros) do centro do circulo no plano x-y
75       posição y (em metros) do centro do circulo no plano x-y
55      posição z (em metros) da base inicia do cilindro
35      raio (em metros) do circulo no plano x-y
68      largura z (ou altura) do cilindro (somada  a posição z)
0.0     não usado
```

//

ARQUIVO 4: Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Globalization;

namespace progCsharpe_Hex3d_matriz2d_float_2c
{
   class Program
   {
      static void Main(string[] args)
      {
         // INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS

         DateTime temp_inicio = DateTime.Now;

         int Nx = 120; // numero de pontos na direção x
         int Ny = 130;  // numero de pontos na direção y
         int Nz = 140; // numero de pontos na direção z
         int ppw = 15; // numero de pontos por comprimento de onda
         int tempomax = 3;
         int tipo_fonte = 1;  // 0=pulso ricker ou 1=senoide
         int HARD_SOFT = 1;  // fonte: 0=hard ou 1=soft
```

```
float ds = 1.0f; // dimensão dos lados do hexagono maior 2D
float rdsdz = 1.0f;  // razão ds/dz
float fator_cdtds = 2.0f; // em comprimentos de onda para dipolo_1
float fator_at = 0.5f;
int delta_passo = 1;
int tempomax2 = 4;
int tempomax3 = 5;
int CPML = 1;
int nxx = 20; // numero par
double Reflexao = 1.0e-6;
double Reflexao2 = 1.0e-6;
double me = 2.5;
double me2 = 2.5;
int numero_materiais = 1;
int numero_objetos = 1;
int ajuste_smax = 0;
int dinicial = 0;
int xpml = 5; // novas variaveis no lugar de ajuste_smax e dinicial
int ypml = 6; // para medida de coeficiente de reflexão nas PMLs
int TFSF = 1;
int deltax = 10; // número par: dimensão para região TFSF
int deltay = 10;
int DISP = 0;
double disp_cte = 1.0;
// FINAL DAS VARIÁVEIS MAIS IMPORTANTES

//////////////////////////////////////////////////
// ENTRADA DE VALORES PARA O ARQUIVO DE ENTRADA3
using (StreamReader entrada = new StreamReader(@"entrada3.txt"))
{
   // INICIO DA LEITURA DO ARQUIVO
   //CultureInfo.CreateSpecificCulture("en-US");
   Nx = int.Parse(entrada.ReadLine());
   Ny = int.Parse(entrada.ReadLine());
   Nz = int.Parse(entrada.ReadLine());
   ppw = int.Parse(entrada.ReadLine());
   tempomax = int.Parse(entrada.ReadLine());
   tipo_fonte = int.Parse(entrada.ReadLine());
   HARD_SOFT = int.Parse(entrada.ReadLine());
   ds = float.Parse(entrada.ReadLine());
   rdsdz = float.Parse(entrada.ReadLine());
   fator_cdtds = float.Parse(entrada.ReadLine());
   fator_at = float.Parse(entrada.ReadLine());
   delta_passo = int.Parse(entrada.ReadLine());
   tempomax2 = int.Parse(entrada.ReadLine());
   tempomax3 = int.Parse(entrada.ReadLine());
   CPML = int.Parse(entrada.ReadLine());
   nxx = int.Parse(entrada.ReadLine());
   Reflexao = double.Parse(entrada.ReadLine());
   Reflexao2 = double.Parse(entrada.ReadLine());
   me = double.Parse(entrada.ReadLine());
   me2 = double.Parse(entrada.ReadLine());
   numero_materiais = int.Parse(entrada.ReadLine());
   numero_objetos = int.Parse(entrada.ReadLine());
   xpml = int.Parse(entrada.ReadLine());
```

```
    ypml = int.Parse(entrada.ReadLine());
    TFSF = int.Parse(entrada.ReadLine());
    deltax = int.Parse(entrada.ReadLine());
    deltay = int.Parse(entrada.ReadLine());
    DISP = int.Parse(entrada.ReadLine());
    disp_cte = double.Parse(entrada.ReadLine());
}// FINAL DA LEITURA DO ARQUIVO: entrada3.txt

// ENTRADA DE VALORES PARA OS ARQUIVOS DE MATERIAIS E OBJETOS
int m_coluna = 4;
int o_coluna = 8;
float[,] materiais = new float[numero_materiais, m_coluna];
float[,] objetos = new float[numero_objetos, o_coluna];
// ENTRADA DE VALORES PARA O ARQUIVO DE MATERIAIS
using (StreamReader material = new StreamReader(@"materiais.txt"))
{
    for (int i = 0; i < numero_materiais; i++)
        for (int j = 0; j < m_coluna; j++)
        {
            materiais[i, j] = float.Parse(material.ReadLine());
        }
}
// ENTRADA DE VALORES PARA O ARQUIVO DE OBJETOS
using (StreamReader objeto = new StreamReader(@"objetos.txt"))
{
    for (int i = 0; i < numero_objetos; i++)
        for (int j = 0; j < o_coluna; j++)
        {
            objetos[i, j] = float.Parse(objeto.ReadLine());
        }
} // FINAL DA LEITURA DOS ARQUIVOS: materiais.txt e objetos.txt

//////////////////////////////////////////////////////
// INICIALIZANDO VARIÁVEIS PARA COMPENSAÇÃO DE DISPERSÃO
int Ndisp = 45;
if (DISP > 1) Ndisp = DISP;
float[] velnorm = new float[Ndisp + 1];
float[] fatorvel = new float[Ndisp + 1];

if (DISP <= 1)
{
    for (int i = 0; i < Ndisp + 1; i++)
    {
        if (DISP == 0)
        {
            fatorvel[i] = 1.0f;
        }
        if (DISP == 1)
        {
            fatorvel[i] = (float) disp_cte;
        }
    }
} // final if externo

// Entrada de valores para arquivo de velocidade normalizada
```

```
        if (DISP > 1)
        {
           using (StreamReader velocidade = new StreamReader(@"velocidade_hex3d.txt"))
           {
              for (int i = 0; i < Ndisp + 1; i++)
              {
                 velnorm[i] = float.Parse(velocidade.ReadLine());
              }
           }
           for (int i = 0; i < Ndisp + 1; i++)
           {
              fatorvel[i] = 2.0f / (velnorm[i] + velnorm[0]);
           }
        } // final if externo
        ////////////////////////////////////////////////////////

        // VALIDANDO VALORES
        if (Nx % 2 == 0) Nx = Nx + 1; // Nx, Ny e Nz são sempre impares para funcionar corretamente
        if (Ny % 2 == 0) Ny = Ny + 1;
        if (Nz % 2 == 0) Nz = Nz + 1;
        if (nxx % 2 == 1) nxx = nxx + 1; // nxx é sempre par (largura da camada CPML)
        if (dinicial <= 0) dinicial = 0; // ajuste fino do cinicial nos campos H1, H2 e Hz (lado direito)
        if (dinicial >= 1) dinicial = 1;
        float cdtds;
        cdtds = (float)(1.0 / Math.Sqrt(fator_cdtds)); // número de Courant
        ////////////////////////////////////////////////////

        // MOSTRANDO ALGUNS VALORES NA TELA DO COMPUTADOR
        //CultureInfo.CreateSpecificCulture("en-US");

        Console.WriteLine("INICIO DO PROGRAMA FDTD ( VERSAO HEX 3D MATRIZ 2D: float )\n");
        Console.WriteLine();
        Console.WriteLine("TEMPO INICIAL = {0}", temp_inicio);
        Console.WriteLine();
        Console.WriteLine("Nx = {0}", Nx);
        Console.WriteLine("Ny = {0}", Ny);
        Console.WriteLine("Nz = {0}", Nz);
        Console.WriteLine();
        Console.WriteLine("Pontos por comprimento de onda (ppw)   = {0}", ppw);
        Console.WriteLine("Tempo (numero de passos)               = {0}", tempomax);
        Console.WriteLine();
        Console.WriteLine("Tipo de Sinal ( Pulso = 0 ou Senoide = 1 )  = {0}", tipo_fonte);
        Console.WriteLine("Tipo de Fonte ( Hard  = 0 ou Soft    = 1 )  = {0}", HARD_SOFT);
        Console.WriteLine();
        Console.WriteLine("Numero de Courant: 1/raiz( {0} )  = {1}", fator_cdtds, cdtds);
        Console.WriteLine("Razao ds / dz             = {0}", rdsdz);
        Console.WriteLine("Dimensão ds               = {0}", ds);
        Console.WriteLine("Fator atenuacao (senoide) = {0}", fator_at);
        Console.WriteLine();
        Console.WriteLine("Camada CPML (Ativada = 1) = {0}", CPML);
        Console.WriteLine("Largura camada CPMLx      = {0}", nxx);
        Console.WriteLine("Largura camada CPMLy      = {0}", 3*nxx + 2);
```

```
Console.WriteLine("Largura camada CPMLz      = {0}", nxx);
Console.WriteLine("Reflexão camada CPML      = {0}", Reflexao);
Console.WriteLine("Coeficiente (me) CPML     = {0}", me);
Console.WriteLine("Reflexão2 camada CPML_z   = {0}", Reflexao2);
Console.WriteLine("Coeficiente (me2) CPML_z  = {0}", me2);
Console.WriteLine();
Console.WriteLine("Região TFSF (Ativada = 1) = {0}", TFSF);
Console.WriteLine("Número de materiais       = {0}", numero_materiais);
Console.WriteLine("Número de objetos         = {0}", numero_objetos);
Console.WriteLine();
Console.WriteLine("Medida de reflexão (Desativada: xpml = 0) = {0}", xpml);
Console.WriteLine("Comp. dispersão (sem = 0, cte = 1, otimizada > 1) = {0}", DISP);
Console.WriteLine();
//////////////////////////////////////////////////////////////////////

// OUTRAS VARIÁVEIS INTERNAS DO PROGRAMA
float dx, dy, dz, Lx, Ly, Lz;
dx = (float)(ds * Math.Sqrt(3.0) / 2.0);// tamanho da célula na direção x
dy = ds / 2.0f;  // tamanho da célula na direção y
dz = ds / rdsdz;
Lx = (Nx) * dx; // comprimento na direção x
Ly = (Ny) * dy; // comprimento na direção y
Lz = (Nz) * dz; // comprimento na direção z
float HARD_SOFT2 = (float)HARD_SOFT;

//////////////////////////////////////////////////////
// calculando dimensões das matrizes para os campos E e H
int ix, rx, jy, ry, jy2, ry2;
ix = Nx / 2;
rx = Nx % 2;
jy = Ny / 2;
ry = Ny % 2;
jy2 = (Ny - 1) / 2;
ry2 = (Ny - 1) % 2;

int PEZ, PH1, PH2, PH3, PH4;
PEZ = (ix + rx) * (jy + ry) + ix * jy;
PH2 = ix * (jy + ry) + (ix + rx) * jy;
PH1 = ix * jy + ix * jy; // mudei, mas é equivalente ao anterior
PH3 = PH1;
PH4 = ix * (jy + ry) + ix * jy; // nova variável

// DIMENSÕES MÁXIMAS PARA VARIAVEIS DA CAMADA CPML
int nyy = 3*nxx + 2;
int cinicial = 10; // usado na CPML (modo TMz)
int cfinal = cinicial + 2 * nyy; // modificado
int cinicial2 = cinicial; // usado na CPML (modo TEz)
int cfinal2 = cinicial2 + 2 * nyy;
int cinicial3 = cinicial; // usado na CPML (direção z)
int cfinal3 = cinicial3 + 2 * nxx + 1;
int i1 = nxx + 1; // inicio e final da camada CPML
int i2 = Nx - nxx - 1;
int j1 = nyy + 1;
int j2 = Ny - nyy; // modificado
int k1 = nxx + 1;
```

```
int k2 = Nz - nxx;

// VARIAVEIS PARA REGIÃO TFSF
// deltax esta agora na entrada.txt
int NLOSS = 20; // para grade auxiliar 1D
int M = Nx + 1;
int Mx = M + NLOSS;
float max_loss = 0.35f; // fator de perda
// calculos para inicializar dimesões da região TFSF
//int deltay = deltax;  // numero par
int iniciox = nxx + deltax; // numero par
int inicioy = nyy + deltay; // numero par
int inicioz = nxx + deltax; // numero par

int ncelx = (Nx - 1 - 2*(nxx + deltax))/2 + 1;
if (ncelx % 2 == 0) ncelx = ncelx - 1;
int ncely = (Ny - 1 - 2*(nyy + deltay))/3 + 1;
if (ncely % 2 == 0) ncely = ncely - 1;

int finalx = iniciox + 2*(ncelx - 1);
int finaly = inicioy + 3 * (ncely - 1);
int finalz = Nz - inicioz;


// VALORES MAXIMOS PARA REDUZIR USO DE MEMORIA
int PEZX, PH2X, PH1X, PE1X, PHZX, PE2X, PE3X;    // lembrar PH3 = PH1
PEZX = PEZ - Ax.fsomaez(j2 + 1, i1, i2, j1, j2);
PH2X = PH2 - Ax.fsomah1(j2 + 1, i1, i2, j1, j2);
PH1X = PH1 - Ax.fsomah23(j2 + 1, i1, i2, j1, j2);
PE1X = PH4 - Ax.fsomaez(j2 + 1, i1, i2, j1, j2);
PE2X = PH1 - Ax.fsomae2(j2 + 1, i1, i2, j1, j2);
PE3X = PH1 - Ax.fsomae3(j2 + 1, i1, i2, j1, j2);
PHZX = PH4 - Ax.fsomahz(j2 + 1, i1, i2, j1, j2);

// posição da fonte (pulso ricker ou senóide) Talvez colacar no arquivo de entrada3.txt
int px = Nx / 2 + 1;
int py = Ny / 2 + 1;
int pz = Nz / 2 + 1;
int pcentro = Ax.p2b(px, py, Nx, Ny, PEZ);

// posições dos quatro pontos para medida de reflexão nas PMLs
int pax = px + xpml;
int pby = py + ypml;
int pa = Ax.p2b(pax, py, Nx, Ny, PEZ);
int pb = Ax.p2b(px, pby, Nx, Ny, PEZ);
int pc = Ax.p2b(pax, pby, Nx, Ny, PEZ);
int pd = pz + xpml;
//////////////////////////////////////////////////////

//////////////////////////////////////////////////////////
Console.WriteLine("INICIALIZACAO - ALOCACAO DINAMICA DE MEMORIA:
MATRIZES 2D float)\n");
// ALOCAÇÃO DINAMICA DE MEMÓRIA
// ARRAY 2D
// CAMPOS ELETRICOS E MAGNETICOS
```

```
float[,] ez = new float[PEZ + 1, Nz + 1];
float[,] e3 = new float[PH1 + 1, Nz + 1];
float[,] h1 = new float[PH2 + 1, Nz + 1];
float[,] e2 = new float[PH1 + 1, Nz + 1];
float[,] h2 = new float[PH1 + 1, Nz + 1];
float[,] e1 = new float[PH4 + 1, Nz + 1];
float[,] h3 = new float[PH1 + 1, Nz + 1];
float[,] hz = new float[PH4 + 1, Nz + 1];
float[,] ez_pxy = new float[Nx + 1, Ny + 1];
float[,] ez_pxz = new float[Nx + 1, Nz + 1];

// COEFIENTES PARA EQUAÇÕES DO METODO FDTD
float[] ch10 = new float[numero_materiais];
float[] ch20 = new float[numero_materiais];
float[] ch30 = new float[numero_materiais];
float[] chz0 = new float[numero_materiais];
float[] ce10 = new float[numero_materiais];
float[] ce20 = new float[numero_materiais];
float[] ce30 = new float[numero_materiais];
float[] cez0 = new float[numero_materiais];
float[] chze2 = new float[numero_materiais];
float[] cezh2 = new float[numero_materiais];
float[] cha = new float[numero_materiais];
float[] chb = new float[numero_materiais];
float[] cea = new float[numero_materiais];
float[] ceb = new float[numero_materiais];

// NOVAS VARIAVEIS PARA CAMADA CPML
byte[, ,] matgrade = new byte[Nx + 1, Ny + 1, Nz + 1]; // materiais da grade 3D
int[,] matez = new int[Nx + 1, Ny + 1]; // materiais para camadas CPML
int[,] math1 = new int[Nx + 1, Ny + 1];
int[,] math23 = new int[Nx + 1, Ny + 1];
int[,] mate23 = new int[Nx + 1, Ny + 1];
int[,] mathz = new int[Nx + 1, Ny + 1];
int[,] mate1 = new int[Nx + 1, Ny + 1];
int[] matztm = new int[Nz + 1];
int[] matzte = new int[Nz + 1];

float[,] ph1ez = new float[PH2X + 1, Nz + 1]; // campo H1
float[,] ph1e2 = new float[PH2 + 1, 2 * nxx + 3];
float[,] ph1e3 = new float[PH2 + 1, 2 * nxx + 3];
float[,] ph2ez = new float[PH1X + 1, Nz + 1]; // campo H2
float[,] ph2e1 = new float[PH1 + 1, 2 * nxx + 3];
float[,] ph2e3 = new float[PH1 + 1, 2 * nxx + 3];
float[,] ph3ez = new float[PH1X + 1, Nz + 1]; // campo H3
float[,] ph3e1 = new float[PH1 + 1, 2 * nxx + 3];
float[,] ph3e2 = new float[PH1 + 1, 2 * nxx + 3];
float[,] pezh1 = new float[PEZX + 1, Nz + 1]; // campo Ez
float[,] pezh2 = new float[PEZX + 1, Nz + 1];
float[,] pezh3 = new float[PEZX + 1, Nz + 1];
float[,] pe1hz = new float[PE1X + 1, Nz + 1]; // campo E1
float[,] pe1h2 = new float[PH4 + 1, 2 * nxx + 2];
float[,] pe1h3 = new float[PH4 + 1, 2 * nxx + 2];
float[,] pe2hz = new float[PE2X + 1, Nz + 1]; // campo E2
float[,] pe2h1 = new float[PH1 + 1, 2 * nxx + 2];
```

```
        float[,] pe2h3 = new float[PH1 + 1, 2 * nxx + 2];
        float[,] pe3hz = new float[PE3X + 1, Nz + 1]; // campo E3
        float[,] pe3h1 = new float[PH1 + 1, 2 * nxx + 2];
        float[,] pe3h2 = new float[PH1 + 1, 2 * nxx + 2];
        float[,] phze1 = new float[PHZX + 1, Nz + 1]; // campo Hz
        float[,] phze2 = new float[PHZX + 1, Nz + 1];
        float[,] phze3 = new float[PHZX + 1, Nz + 1];
        float[] sh = new float[2 * nyy + 1]; // condutividades das camadas CPML (modo TMz)
        float[] am = new float[2 * nyy + 1]; // novo tamanho
        float[] bm = new float[2 * nyy + 1];
        float[] sh2 = new float[2 * nyy + 1]; // condutividades das camadas CPML (modo TEz)
??
        float[] am2 = new float[2 * nyy + 1];
        float[] bm2 = new float[2 * nyy + 1];
        float[] shz = new float[2 * nxx + 2]; // condutividades das camadas CPML (direção z)
        float[] amz = new float[2 * nxx + 2];
        float[] bmz = new float[2 * nxx + 2];

        int[] byez = new int[nxx + 2]; // linhas verticais
        int[] byh23 = new int[nxx + 2];
        int[] bye23e = new int[nxx + 2];
        int[] bye23d = new int[nxx + 2];
        int[] byhze = new int[nxx + 2];
        int[] byhzd = new int[nxx + 2];
        int[] bxez = new int[nyy + 2]; // linhas horizontais
        int[] bxh23 = new int[nyy + 2];
        int[] bxe23e = new int[nyy + 2];
        int[] bxe23d = new int[nyy + 2];
        int[] bxhze = new int[nyy + 2];
        int[] bxhzd = new int[nyy + 2];

        // ALOCANDO MEMORIA PARA GRADE 1D AUXILIAR (REGIAO TFSF)
        float[] hy1 = new float[Mx];
        float[] chyh1 = new float[Mx];
        float[] chye1 = new float[Mx];
        float[] ez1 = new float[Mx];
        float[] cezh1 = new float[Mx];
        float[] ceze1 = new float[Mx];

        // VARIAVEIS PARA MEDIDA DE REFLEXÃO DAS PMLs
        float[] eza = new float[tempomax];
        float[] ezb = new float[tempomax];
        float[] ezc = new float[tempomax];
        float[] ezd = new float[tempomax];

        // VARIAVEIS PARA COMPENSAÇÃO DE DISPERSÃO
        float[,] cdisp = new float[Nx + 1, Ny + 1];

        ////////////////////////////////////////////////////////////////////
        // Calculando memória total necessária para campos eletricos e magneticos
        long memoria = (PEZ + 1) * (Nz + 1) + 4 * (PH1 + 1) * (Nz + 1) + 2 * (PH4 + 1) * (Nz +
1) +
            (PH2 + 1) * (Nz + 1) + (Nx + 1) * (Ny + 1) + (Nx + 1) * (Nz + 1);
        memoria = memoria * sizeof(float);
        // Calculando memória para as variaveis inteiras novas
```

```
        long memoria2 = 2 * (Nz + 1) + 6 * (Nx + 1) * (Ny + 1);
        long memoria2b = (Nx + 1) * (Ny + 1) * (Nz + 1);
        memoria2 = memoria2 * sizeof(int);
        memoria2b = memoria2b * sizeof(byte);
        // Calculando memória para as variaveis float novas
        long memoria3 = (PH2X + 1) * (Nz + 1) + 2 * (PH1X + 1) * (Nz + 1) + 3 * (PEZX + 1) *
(Nz + 1) +
           (PE1X + 1) * (Nz + 1) + (PE2X + 1) * (Nz + 1) + (PE3X + 1) * (Nz + 1) + 9 * (2 * nxx
+ 2) +
           3 * (PHZX + 1) * (Nz + 1) + 14 * numero_materiais + 4 * (PH1 + 1) * (2 * nxx + 3) +
           2 * (PH1 + 1) * (2 * nxx + 2) + 2 * (PH4 + 1) * (2 * nxx + 2) +
           2 * (PH2 + 1) * (2 * nxx + 3) + (Nx + 1) * (Ny + 1);
        memoria3 = memoria3 * sizeof(float);
        long memoria_total = memoria + memoria2 + memoria2b + memoria3;
        float mem23 = 100.0f * (memoria2 + memoria2b + memoria3) / memoria_total;

        ////////////////////////////////////////////////////////////////////////
        Console.WriteLine("MEMÓRIA ALOCADA PARA CADA float:          {0} BYTES",
sizeof(float));
        Console.WriteLine("MEMÓRIA ALOCADA PARA CADA int:            {0} BYTES",
sizeof(int));
        Console.WriteLine("MEMÓRIA ALOCADA PARA CADA byte:            {0} BYTE",
sizeof(byte));
        Console.WriteLine("MEMÓRIA ALOCADA PARA ARRAYS 2D CAMPOS: {0} BYTES",
memoria);
        Console.WriteLine("MEMÓRIA ALOCADA PARA ARRAYS INT:          {0} BYTES",
memoria2);
        Console.WriteLine("MEMÓRIA ALOCADA PARA ARRAYS BYTE:         {0} BYTES",
memoria2b);
        Console.WriteLine("MEMÓRIA ALOCADA PARA ARRAYS 2D CPML:    {0} BYTES",
memoria3);
        Console.WriteLine("MEMÓRIA ALOCADA PARA TODOS OS ARRAYS:  {0} BYTES",
memoria_total);
        Console.WriteLine("MEMÓRIA ALOCADA EXTRA ( EM % ):         {0} %", mem23);
        Console.WriteLine("FINAL INICIALIZACAO: hz[p,k] = {0}\n", hz[PH1, Nz]);
        Console.WriteLine("MATRIZES DE SAIDA: FLOAT SCIENTIFIC, MANTISSA 18
DIGITOS ( VERSÃO 8_D )\n");
        Console.WriteLine("A PARTIR DA VERSÃO 7B: USANDO REGIÃO CAMPO TOTAL /
CAMPO ESPALLHADO\n");
        Console.WriteLine("A PARTIR DA VERSÃO 6: COM MATRIZ EXPANDIDA DE
CONDUTIVIDADES PARA CAMADA CPML\n");
        Console.WriteLine("E ESPESSURAS OTIMIZADAS DAS CAMADAS DE
CONDUTIVIDADE CPML NAS DIREÇÕES X E Y\n");
        Console.WriteLine("COM SINAIS CORRIGIDOS PARA O MODO TE E 4 SAIDAS
PARA MEDIDA DE REFLEXÃO NAS PMLs\n");
        Console.WriteLine("COM OBJETOS: PLACA RETANGULAR (1), ESFERA (2) E
CILINDRO (3)\n");
        Console.WriteLine("E OBJETOS COM PRECISÃO NO CENTRO: PLACA
RETANGULAR (4) E ESFERA (5)\n");
        Console.WriteLine("COM xpml = 0: SEM SAÍDA PARA MEDIDA DE REFLEXÃO\n");
        Console.WriteLine("COM COMPENSAÇÃO DE DISPERSÃO (DISP): SEM (0) CTE (1)
E OTIMIZADA (>1)\n");
        //Console.WriteLine("FINAL INICIALIZACAO: materiais = {0},{1},{2},{3}\n", materiais[1,
0], materiais[1, 1],
        //  materiais[1, 2], materiais[1, 3]);
```

```
        //Console.WriteLine("FINAL INICIALIZACAO: ch10 = {0}, {1}\n", cha[0], cha[1]);

        ////////////////////////////////////////////////////////////////////

        // INICIALIZANDO COEFICIENTES EM FUNÇÃO DOS TIPOS DE MATERIAIS
        MATx.materialx(materiais, numero_materiais, ch10, ch20,
        ch30, chz0, ce10, ce20, ce30, cez0, chze2,
        cezh2, cha, chb, cea, ceb, cdtds, rdsdz, ds);
        /*
        Console.WriteLine("FINAL INICIALIZACAO: ch10 = {0}, {1}\n", ch10[0], ch10[1]);
        Console.WriteLine("FINAL INICIALIZACAO: ce10 = {0}, {1}\n", ce10[0], ce10[1]);
        Console.WriteLine("FINAL INICIALIZACAO: cha  = {0}, {1}\n", cha[0], cha[1]);
        Console.WriteLine("FINAL INICIALIZACAO: chb  = {0}, {1}\n", chb[0], chb[1]);
        Console.WriteLine("FINAL INICIALIZACAO: cezh2 = {0}, {1}\n", cezh2[0], cezh2[1]);
        */

        // INICIALIZANDO OBJETOS NA GRADE 3D
        OBJx.objetox(objetos, numero_objetos, matgrade, dx, dy, dz);

        // INICIALIZANDO GRADE AUXILIAR 1D PARA REGIÃO TFSF
        double imp0a = (120.0 * Math.PI);// impedância característica no vácuo
        float imp0b = (float)imp0a;
        TS.inicializando_grade_1d(ceze1, cezh1, chyh1, chye1, Mx, NLOSS, max_loss,
cdtds,imp0b);

        // INICIALIZANDO AJUSTES PARA LINHAS VERTICAIS E HORIZONTAIS DA
CAMADA CPML
        double dx2 = Math.Sqrt(3.0) * ds / 2.0;
        double dy2 = ds / 2.0;
        double tancpml = Math.Sqrt(3.0);
        Bx.func_by(nxx, nyy, Ny, 0.0, 0.0, 0, tancpml, dx2, dy2, byez); // linhas verticais
        Bx.func_by(nxx, nyy, Ny, 0.5, 0.5, 0, tancpml, dx2, dy2, byh23);
        Bx.func_by(nxx, nyy, Ny, 1.0 / 6.0, 0.5, 0, tancpml, dx2, dy2, bye23e);
        Bx.func_by(nxx, nyy, Ny, 5.0 / 6.0, 0.5, 0, tancpml, dx2, dy2, bye23d);
        Bx.func_by(nxx, nyy, Ny, 2.0 / 3.0, 0.0, 0, tancpml, dx2, dy2, byhze);
        Bx.func_by(nxx, nyy, Ny, 1.0 / 3.0, 0.0, 0, tancpml, dx2, dy2, byhzd);
        Bx.func_bx(nxx, nyy, Nx, 0.0, 0.0, 1, tancpml, dx2, dy2, bxez); // linhas horizontais
        Bx.func_bx(nxx, nyy, Nx, 0.5, 0.5, 0, tancpml, dx2, dy2, bxh23);
        Bx.func_bx(nxx, nyy, Nx, 1.0 / 6.0, 0.5, 0, tancpml, dx2, dy2, bxe23e);
        Bx.func_bx(nxx, nyy, Nx, 5.0 / 6.0, 0.5, 0, tancpml, dx2, dy2, bxe23d);
        Bx.func_bx(nxx, nyy, Nx, 2.0 / 3.0, 0.0, 0, tancpml, dx2, dy2, bxhze);
        Bx.func_bx(nxx, nyy, Nx, 1.0 / 3.0, 0.0, 0, tancpml, dx2, dy2, bxhzd);

        // INICIALIZANDO VETORES DE CONDUTIVIDADE COM A ESPESSURA PML
DESEJADA
        double bs = ds / Math.Sqrt(3.0);
        double dsmax = (nyy - ajuste_smax) * bs; // modos TMz e TEz
        double imp0 = (120.0 * Math.PI);// impedância característica no vácuo
        double smax = (-1.0 * (me + 1.0) * Math.Log(Reflexao) / (2.0 * imp0 * dsmax));

        //GERANDO VETOR CONDUTIVIDADE NO PLANO XY
        for (int i = 0; i < (2 * nyy + 1); i++)
        {
           if (i < 2*nyy - 2*ajuste_smax + 1)
           {
```

```
            sh[i] = (float)(Math.Pow(i * 0.5 * bs / dsmax, me) * smax);
        }
        else
        {
            sh[i] = (float) smax;
        }

        am[i] = (float)Ax.func_fia(sh[i], cdtds, imp0, ds);
        bm[i] = (float)Ax.func_fib(sh[i], cdtds, imp0, ds);
        am2[i] = am[i];
        bm2[i] = bm[i];

    }

    // DIREÇÃO Z NO PLANO XZ
    double dzmax = (nxx + 0.5) * dz; // na direção z
    double szmax = (-1.0 * (me2 + 1.0) * Math.Log(Reflexao2) / (2.0 * imp0 * dzmax));
    for (int i = 0; i < 2 * nxx + 2; i++)
    {
        shz[i] = (float)(Math.Pow(i * 0.5 * dz / dzmax, me2) * szmax);
        amz[i] = (float)Ax.func_fia(shz[i], cdtds, imp0, ds); // ds ou dz ?? DUVIDA
IMPORTANTE
        bmz[i] = (float)Ax.func_fib(shz[i], cdtds, imp0, ds);
    }
    ////////////////////////////////////////////////////////////////////////

    // GERANDO MATRIZ PARA COMPENSAÇÃO DE DISPERSÃO
    // Aplicada nas classes TM.cs e TE.cs, mas ainda não na TS.cs
    double fator_disp = 1.0 * Ndisp / 45.0;
    double altura = 1.0 * (Nz - pz) * dz;
    for (int i = 0; i < Nx + 1; i++ )
        for (int j = 0; j < Ny + 1; j++)
        {
            int kteta;
            double x = (i - px)*dx;
            double y = (j - py)*dy;
            double r =  Math.Sqrt(x * x + y * y);
            if (r <= 1.0e-26 || r <= altura)
            {
                kteta = Ndisp;
            }
            else
            {
                double teta = Math.Atan(altura / r);
                kteta = (int)(fator_disp * teta * 180.0 / Math.PI);
            }
            cdisp[i, j] = fatorvel[kteta];
        }
    ////////////////////////////////////////////////////////////////////////
        // TESTANDO VARIAVEIS
        /*
        for (int i = 0; i < 30; i++)
        {
            Console.WriteLine(" fatorvel[{0}] = {1}\n",i ,fatorvel[i]);
        }
```

```
        */
        ////////////////////////////////////////////////////////////////////

        // CONTORNO MAIS SIMPLES PARA INCLUIR CAMPOS DO MODO TEz
        // GERA LINHAS VERTICAIS QUE DIMINUEM NA DIREÇÃO CENTRAL DA GRADE (CAMADA CPML)
        for (int a = 1; a < nxx + 2; a++)
        {
            // campos Ez e H1
            for (int j = 3 * a - 2; j < Ny - 3 * a + 4; j++)
            {
                matez[a, j] = cfinal - 6 * a + 6;
                matez[Nx + 1 - a, j] = cfinal - 6 * a + 6;
                math1[a, j] = cfinal - 6 * a + 6;
                math1[Nx + 1 - a, j] = cfinal - 6 * a + 6;
                //Console.WriteLine(" math1[{0},{1}] = {2}\n", a, j, math1[a, j]);
                //Console.WriteLine(" math1[{0},{1}] = {2}\n", Nx + 1 - a, j, math1[Nx + 1 - a, j]);
            }

            // campos H2 e H3
            for (int j = 3 * a - 2; j < Ny - 3 * a + 2; j++)
            {

                if (a <= nxx)
                {
                    math23[a, j] = cfinal - 6 * a + 3;
                    math23[Nx - a, j] = cfinal - 6 * a + 3;
                }
                if (a == nxx + 1)
                {
                    math23[a, j] = cinicial + dinicial;
                    math23[Nx - a, j] = cinicial + dinicial;
                }
            }

            // campos E2 e E3 (lado esquerdo)
            for (int j = 3 * a - 2; j < Ny - 3 * a + 3; j++)
            {
                mate23[a, j] = cfinal - 6 * a + 5;
            }

            // campos E2 e E3 (lado direito)
            for (int j = 3 * a; j < Ny - 3 * a + 1; j++)
            {
                if (a <= nxx)
                {
                    mate23[Nx - a, j] = cfinal - 6 * a + 1;
                }
            }

            // campos Hz e E1 (lado esquerdo)
            for (int j = 3 * a; j < Ny - 3 * a + 2; j++)
            {
                if (a <= nxx)
                {
```

```
                    mathz[a, j] = cfinal - 6 * a + 2;
                    mate1[a, j] = cfinal - 6 * a + 2;
                }
                if (a == nxx + 1)
                {
                    mathz[a, j] = cinicial;

                }
            }

            // campos Hz e E1 (lado direito)
            for (int j = 3 * a - 1; j < Ny - 3 * a + 3; j++)
            {
                if (a <= nxx)
                {
                    mathz[Nx - a, j] = cfinal - 6 * a + 4;
                }
                if (a == nxx + 1)
                {
                    mathz[Nx - a, j] = cinicial + 2 * dinicial;
                }
                mate1[Nx - a, j] = cfinal - 6 * a + 4;
            }
        }

    // GERA LINHAS HORIZONTAIS QUE DIMINUEM NA DIREÇÃO CENTRAL DA GRADE (CAMADA CPML)
    for (int a = 1; a < nyy + 2; a++)
    {
        // campos Ez e H1
        for (int i = 1 + bxez[a]; i < Nx + 1 - bxez[a]; i++)
        {
            if (a <= nyy)
            {
                matez[i, a] = cfinal - 2 * a + 2;
                matez[i, Ny + 1 - a] = cfinal - 2 * a + 2;
                math1[i, a] = cfinal - 2 * a + 2;
                math1[i, Ny + 1 - a] = cfinal - 2 * a + 2;
            }
            if (a == (nyy + 1))
            {
                matez[i, a] = cinicial;
                matez[i, Ny + 1 - a] = cinicial;
                math1[i, a] = cinicial;
                math1[i, Ny + 1 - a] = cinicial;
            }
            //Console.WriteLine(" math1[{0},{1}] = {2} bxez[{3}] = {4}\n", i, a, math1[i, a], a, bxez[a]);

        }

        // campos H2 e H3
        if (a <= nyy)
        {
            for (int i = 1 + bxh23[a]; i < Nx - bxh23[a]; i++)
```

```
        {
            math23[i, a] = cfinal - 2 * a + 1;
            math23[i, Ny - a] = cfinal - 2 * a + 1;
        }
    }

    // campos E2 e E3
    if (a <= nyy)
    {
        for (int i = 1 + bxe23e[a]; i < Nx - bxe23d[a]; i++)
        {
            mate23[i, a] = cfinal - 2 * a + 1;
            mate23[i, Ny - a] = cfinal - 2 * a + 1;
        }
    }

    // campos Hz e E1
    for (int i = 1 + bxhze[a]; i < Nx - bxhzd[a]; i++)
    {
        if (a <= nyy)
        {
            mathz[i, a] = cfinal - 2 * a + 2;
            mathz[i, Ny + 1 - a] = cfinal - 2 * a + 2;
            mate1[i, a] = cfinal - 2 * a + 2;
            mate1[i, Ny + 1 - a] = cfinal - 2 * a + 2;
        }
        if (a == nyy + 1)
        {
            mathz[i, a] = cinicial;
            mathz[i, Ny + 1 - a] = cinicial;
            mate1[i, a] = cinicial;
            mate1[i, Ny + 1 - a] = cinicial;
        }

    }
}

// GERA LINHAS TRANSVERSAIS A DIREÇÃO Z (CAMADA CPML)
for (int a = 1; a < nxx + 2; a++)
{
    matztm[a] = cfinal3 - 2 * a + 1;
    matztm[Nz + 1 - a] = cfinal3 - 2 * a + 1;
    if (a <= nxx)
    {

        matzte[a] = cfinal3 - 2 * a;
        matzte[Nz - a] = cfinal3 - 2 * a;
        matzte[Nz] = cfinal3; // introduzido por segurança
    }
} // FINAL LINHAS PARA CAMADA CPML

////////////////////////////////////////////////////////////////
// INICIO LOOP DO TEMPO
for (int tempo = 0; tempo < tempomax; tempo++)
{
```

```
            // SEM COMPUTAÇÃO PARALELA
            DateTime temp_inicio2 = DateTime.Now;

            // ATUALIZA CAMPOS MAGNÉTICOS
            TM.atualiza_tmz_h1(cha, chb, ez, e2, e3, h1, Nx, Ny, Nz, PH1, PH2, PH3, PEZ,
              i1, i2, j1, j2, cinicial, CPML, math1, am, bm, amz, bmz, ph1ez, ph1e2, ph1e3,
matgrade,
              ch10, matztm, cinicial3, k1, k2,cdisp);
            TM.atualiza_tmz_h2(cha, chb, ez, e1, e3, h2, Nx, Ny, Nz, PH1, PEZ, PH4,
              i1, i2, j1, j2, cinicial, CPML, math23, am, bm, amz, bmz, ph2ez, ph2e1, ph2e3,
matgrade,
              ch20, matztm, cinicial3, k1, k2, cdisp);
            TM.atualiza_tmz_h3(cha, chb, ez, e1, e2, h3, Nx, Ny, Nz, PH1, PH3, PEZ, PH4,
              i1, i2, j1, j2, cinicial, CPML, math23, am, bm, amz, bmz, ph3ez, ph3e1, ph3e2,
matgrade,
              ch30, matztm, cinicial3, k1, k2, cdisp);
            TE.atualiza_tez_hz(chze2, e1, e2, e3, hz, Nx, Ny, Nz, PH1, PH3, PH4,
              i1, i2, j1, j2, cinicial2, CPML, mathz, am2, bm2, phze1, phze2, phze3, matgrade,
chz0, cdisp);

            // ATUALIZA CAMPOS ELETRICOS
            TE.atualiza_tez_e1(cea, ceb, hz, h2, h3, e1, Nx, Ny, Nz, PH1, PH3, PH4,
              i1, i2, j1, j2, cinicial2, CPML, mate1, am2, bm2, amz, bmz, pe1hz, pe1h2, pe1h3,
matgrade,
              ce10, matzte, cinicial3, k1, k2, cdisp);
            TE.atualiza_tez_e2(cea, ceb, hz, h1, h3, e2, Nx, Ny, Nz, PH2, PH3, PH4,
              i1, i2, j1, j2, cinicial2, CPML, mate23, am2, bm2, amz, bmz, pe2hz, pe2h1, pe2h3,
matgrade,
              ce20, matzte, cinicial3, k1, k2, cdisp);
            TE.atualiza_tez_e3(cea, ceb, hz, h1, h2, e3, Nx, Ny, Nz, PH1, PH2, PH3, PH4,
              i1, i2, j1, j2, cinicial2, CPML, mate23, am2, bm2, amz, bmz, pe3hz, pe3h1, pe3h2,
matgrade,
              ce30, matzte, cinicial3, k1, k2, cdisp);

            TM.atualiza_tmz_ez(cezh2, h1, h2, h3, ez, Nx, Ny, Nz, PH1, PH2, PH3, PEZ,
              i1, i2, j1, j2, cinicial, CPML, matez, am, bm, pezh1, pezh2, pezh3, matgrade, cez0,
cdisp);

            if (TFSF == 1)
            {
              // USA REGIÃO CAMPO TOTAL / CAMPO ESPALHADO
              TS.atualiza_tfsf_h(h1, h2, h3, cha, ez1, matgrade, iniciox, finalx, inicioy, finaly,
inicioz,
                finalz, Nx, Ny, PH1, PH2, PH4);
              TS.atualiza_tfsf_ez(ez, cezh2, hy1, matgrade, iniciox, finalx, inicioy, finaly, inicioz,
finalz, Nx, Ny, PEZ);
              TS.atualiza_tfsf_e123(e1, e2, e3, ceb, hy1, matgrade, iniciox, finalx, inicioy, finaly,
inicioz,
                finalz, Nx, Ny, PH1, PH3, PH4);
              TS.atualiza_tfsf_grade_1d(tempo, cdtds, ppw, tipo_fonte, fator_at, ez1, hy1,
ceze1, cezh1, chyh1, chye1, Mx);
            }
            else
            {
              // APLICA A FONTE PONTUAL NO PONTO ESCOLHIDO
```

```
                ez[pcentro, pz] = HARD_SOFT2 * ez[pcentro, pz] + Ax.fonte_1(tempo, 0.0f, cdtds,
ppw, tipo_fonte, fator_at);
            }

            // atualizando os tres pontos para medida de reflexão nas PMLs
            if (xpml > 0)
            {
                eza[tempo] = ez[pa, pz];
                ezb[tempo] = ez[pb, pz];
                ezc[tempo] = ez[pc, pz];
                ezd[tempo] = ez[pcentro, pd];
            }

            if (tempo == tempomax2 - 1)
            {
                int nteste = tempomax2;
                string txx = nteste.ToString();
                string txsaida1 = @"ez_pxy_1_" + txx + ".txt";
                Saida.saida1(txsaida1, Nx, Ny, PEZ, pz, ez, ez_pxy);
                string txsaida2 = @"ez_pxz_2_" + txx + ".txt";
                Saida.saida2(txsaida2, Nx, Ny, Nz, PEZ, py, ez, ez_pxz);
            }
            if (tempo == tempomax3 - 1)
            {
                int nteste = tempomax3;
                string txx = nteste.ToString();
                string txsaida1 = @"ez_pxy_1_" + txx + ".txt";
                Saida.saida1(txsaida1, Nx, Ny, PEZ, pz, ez, ez_pxy);
                string txsaida2 = @"ez_pxz_2_" + txx + ".txt";
                Saida.saida2(txsaida2, Nx, Ny, Nz, PEZ, py, ez, ez_pxz);
            }

            DateTime temp_final2 = DateTime.Now;
            TimeSpan tempo_total2 = temp_final2 - temp_inicio2;
            if ((tempo + 1) % delta_passo == 0)
            {
                Console.WriteLine(" {0,8} / {1} em {2} ", tempo + 1, tempomax, tempo_total2);
            }
        } // FINAL LOOP DO TEMPO
        ////////////////////////////////////////////////////////////

        //METODOS PARA ESCRITA NOS DOIS ARQUIVOS DE SAÍDA
        int nteste2 = tempomax;
        string txx2 = nteste2.ToString();
        string txsaida1a = @"ez_pxy_1_" + txx2 + ".txt";
        Saida.saida1(txsaida1a, Nx, Ny, PEZ, pz, ez, ez_pxy);
        string txsaida2a = @"ez_pxz_2_" + txx2 + ".txt";
        Saida.saida2(txsaida2a, Nx, Ny, Nz, PEZ, py, ez, ez_pxz);

        // METODOS PARA ESCRITA DAS 4 VARIAVEIS PARA MEDIDA DE REFLEXÃO
NAS PMLs
        if (xpml > 0)
        {
            string tnxx = nxx.ToString();
            string tcpml = CPML.ToString();
```

```
            string tx_eza = @"ez_A_nxx_" + tnxx + "_CPML_" + tcpml + "_Temp_" + txx2 + ".txt";
            SaidaR.saidar(tx_eza, tempomax, eza);
            string tx_ezb = @"ez_B_nxx_" + tnxx + "_CPML_" + tcpml + "_Temp_" + txx2 + ".txt";
            SaidaR.saidar(tx_ezb, tempomax, ezb);
            string tx_ezc = @"ez_C_nxx_" + tnxx + "_CPML_" + tcpml + "_Temp_" + txx2 + ".txt";
            SaidaR.saidar(tx_ezc, tempomax, ezc);
            string tx_ezd = @"ez_D_nxx_" + tnxx + "_CPML_" + tcpml + "_Temp_" + txx2 +
".txt";
            SaidaR.saidar(tx_ezd, tempomax, ezd);
        }

        // MEDINDO TEMPO DE EXECUÇÃO DO PROGRAMA
        DateTime temp_final = DateTime.Now;
        TimeSpan tempo_total = temp_final - temp_inicio;
        Console.WriteLine("\nTEMPO INICIAL: {0}", temp_inicio);
        Console.WriteLine("\nTEMPO FINAL:   {0}", temp_final);
        Console.WriteLine("\nTEMPO TOTAL (HH : MM : SS): {0}", tempo_total);
        Console.Write("\nENTRE COM UM NÚMERO OU LETRA PARA SAIR DO
PROGRAMA: ");
        string FINAL = Console.ReadLine();
        // FINAL DO PROGRAMA
    }
  }
}
////////////////////////////////////////////////////////////////////////////////////
```

ARQUIVO 5: Ax.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace progCsharpe_Hex3d_matriz2d_float_2c
{
  class Ax
  {   // FUNÇÕES DE POSIÇÃO
      // Função para h3, hz e e2
      public static int p2a(int i, int j, int Nx, int Ny, int PH3)
      {
          int p1, p2;

          if ((i <= 0) || (i >= Nx + 1) || (j <= 0) || (j >= Ny + 1))
          {
              p2 = 0;
          }
          else
          {
              p1 = i + (j - 1) * (Nx - 1);
              p2 = (p1 + 1) / 2;
              if (p2 > PH3)
                  p2 = 0;
          }
          return p2;
```

```
}

// Função para ez
public static int p2b(int i, int j, int Nx, int Ny, int PEZ)
{
    int p1, p2;
    if ((i <= 0) || (i >= Nx + 1) || (j <= 0) || (j >= Ny + 1))
    {
        p2 = 0;
    }
    else
    {
        p1 = i + (j - 1) * (Nx);
        p2 = (p1 + 1) / 2;
        if (p2 > PEZ)
            p2 = 0;
    }
    return p2;
}

// Função para h1
public static int p2c(int i, int j, int Nx, int Ny, int PH2)
{
    int p1, p2;
    if ((i <= 0) || (i >= Nx + 1) || (j <= 0) || (j >= Ny + 1))
    {
        p2 = 0;
    }
    else
    {
        p1 = i + (j - 1) * (Nx);
        p2 = (p1) / 2;
        if (p2 > PH2)
            p2 = 0;
    }
    return p2;
}

// Função para h2, e1 e e3
public static int p2d(int i, int j, int Nx, int Ny, int PH1)
{
    int p1, p2;
    if ((i <= 0) || (i >= Nx + 1) || (j <= 0) || (j >= Ny + 1))
    {
        p2 = 0;
    }
    else
    {
        p1 = i + (j - 1) * (Nx - 1);
        p2 = (p1 + 1) / 2;
        if (p2 > PH1)
            p2 = 0;
    }
    return p2;
}
```

```
// NOVAS FUNÇÕES PARA CAMADA CPML
// FUNÇÕES PARA CAMPO Ez
public static int fsomaez( int j, int i1 , int i2, int j1, int j2)
{
   if (j >= (j2-1))  j = j2 - 1;
   int  ix2 = (i2 - i1)/2;
   int  ix1 =  ix2 + 1   ;
   int  jx2 = (j - j1)/2;
   int  jx1 = jx2 + (j - j1)%2;
   int  soma = jx1*ix1 + jx2*ix2;
   return soma;
}

public static int xp2b(int i, int j, int Nx, int Ny, int PEZ,int i1 , int i2, int j1, int j2)
{
   int p = 0;
   if (j <= j1)  p = p2b(i,j,Nx,Ny,PEZ);
   if (j > j1 && i <= i1) p = p2b(i, j, Nx, Ny, PEZ) - fsomaez(j - 1, i1, i2, j1, j2);
   if ((j > j1 && i > i2) || j >= j2) p = p2b(i, j, Nx, Ny, PEZ) - fsomaez(j, i1, i2, j1, j2);
   return p;
}

// FUNÇÕES PARA CAMPO H1
public static int fsomah1(int j, int i1, int i2, int j1, int j2)
{
   if (j >= (j2 - 1)) j = j2 - 1;
   int ix2 = (i2 - i1) / 2;
   int ix1 = ix2 + 1;
   int jx2 = (j - j1) / 2;
   int jx1 = jx2 + (j - j1) % 2;
   int soma = jx1 * ix2 + jx2 * ix1;
   return soma;
}

public static int xp2c(int i, int j, int Nx, int Ny, int PH2, int i1, int i2, int j1, int j2)
{
   int p = 0;
   if (j <= j1) p = p2c(i, j, Nx, Ny, PH2);
   if (j > j1 && i <= i1) p = p2c(i, j, Nx, Ny, PH2) - fsomah1(j - 1, i1, i2, j1, j2);
   if ((j > j1 && i > i2) || j >= j2) p = p2c(i, j, Nx, Ny, PH2) - fsomah1(j, i1, i2, j1, j2);
   return p;
}

// FUNÇÕES PARA CAMPOS H2 E H3
public static int fsomah23(int j, int i1, int i2, int j1, int j2)
{
   if (j >= (j2 - 1)) j = j2 - 1;
   int ix2 = (i2 - i1) / 2;
   int ix1 = ix2 ;
   int jx2 = (j - j1) / 2;
   int jx1 = jx2 + (j - j1) % 2;
   //int soma = jx1 * ix1 + jx2 * ix2;
   int soma = (j - j1 + 1) * ix2; // equação mais simples
   return soma;
```

```
    }
    // CAMPO H2
    public static int xp2d(int i, int j, int Nx, int Ny, int PH1, int i1, int i2, int j1, int j2)
    {
        int p = 0;
        if (j < j1) p = p2d(i, j, Nx, Ny, PH1);
        if (j >= j1 && i <= i1) p = p2d(i, j, Nx, Ny, PH1) - fsomah23(j - 1, i1, i2, j1, j2);
        if ((j >= j1 && i >= i2) || j >= j2) p = p2d(i, j, Nx, Ny, PH1) - fsomah23(j, i1, i2, j1, j2);
        return p;
    }

    // CAMPO H3
    public static int xp2a(int i, int j, int Nx, int Ny, int PH3, int i1, int i2, int j1, int j2)
    {
        int p = 0;
        if (j < j1) p = p2a(i, j, Nx, Ny, PH3);
        if (j >= j1 && i <= i1) p = p2a(i, j, Nx, Ny, PH3) - fsomah23(j - 1, i1, i2, j1, j2);
        if ((j >= j1 && i >= i2) || j >= j2) p = p2a(i, j, Nx, Ny, PH3) - fsomah23(j, i1, i2, j1, j2);
        return p;
    }
    // FUNÇÕES PARA MODO TEz
    // CAMPO E1
    public static int yp2d(int i, int j, int Nx, int Ny, int PH4, int i1, int i2, int j1, int j2)
    {
        int p = 0;
        if (j <= j1) p = p2d(i, j, Nx, Ny, PH4);//nova variavel PH4
        if (j > j1 && i < i1) p = p2d(i, j, Nx, Ny, PH4) - fsomaez(j - 1, i1, i2, j1, j2); // corrigido i <
i1
        if ((j > j1 && i >= i2) || j >= j2) p = p2d(i, j, Nx, Ny, PH4) - fsomaez(j, i1, i2, j1, j2);
        return p;
    }

    // FUNÇÕES PARA CAMPO Hz
    public static int fsomahz(int j, int i1, int i2, int j1, int j2)
    {
        if (j >= (j2 - 1)) j = j2 - 1;
        int ix2 = (i2 - i1) / 2;
        int soma = (j - j1)*ix2; // equação mais simples
        return soma;
    }
    public static int yp2a(int i, int j, int Nx, int Ny, int PH4, int i1, int i2, int j1, int j2)
    {
        int p = 0;
        if (j <= j1) p = p2a(i, j, Nx, Ny, PH4); // nova variável PH4
        if (j > j1 && i <= i1) p = p2a(i, j, Nx, Ny, PH4) - fsomahz(j - 1, i1, i2, j1, j2);
        if ((j > j1 && i >= i2) || j >= j2) p = p2a(i, j, Nx, Ny, PH4) - fsomahz(j, i1, i2, j1, j2);
        return p;
    }

    // FUNÇÕES PARA CAMPO E2
    public static int fsomae2(int j, int i1, int i2, int j1, int j2)
    {
        if (j >= (j2 - 1)) j = j2 - 1;
        int ix2 = (i2 - i1) / 2;
        int ix1 = ix2 + 1;
```

```
        int jx2 = (j - j1 + 1) / 2;
        int jx1 = jx2 + (j - j1 + 1) % 2;
        int soma = jx1 * ix2 + jx2 * ix1;//inversão nas multiplicações
        return soma;
    }
    // CAMPO E2
    public static int zp2a(int i, int j, int Nx, int Ny, int PH3, int i1, int i2, int j1, int j2)
    {
        int p = 0;
        if (j < j1) p = p2a(i, j, Nx, Ny, PH3);
        if (j >= j1 && i <= i1) p = p2a(i, j, Nx, Ny, PH3) - fsomae2(j - 1, i1, i2, j1, j2);
        if ((j >= j1 && i > i2) || j >= j2) p = p2a(i, j, Nx, Ny, PH3) - fsomae2(j, i1, i2, j1,
j2);//modificado j >= j1
        return p;
    }
    // FUNÇÕES PARA CAMPO E3
    public static int fsomae3(int j, int i1, int i2, int j1, int j2)
    {
        if (j >= (j2 - 1)) j = j2 - 1;
        int ix2 = (i2 - i1) / 2;
        int ix1 = ix2 + 1;
        int jx2 = (j - j1 + 1) / 2;
        int jx1 = jx2 + (j - j1 + 1) % 2;
        int soma = jx1 * ix1 + jx2 * ix2;
        return soma;
    }
    // CAMPO E3
    public static int zp2d(int i, int j, int Nx, int Ny, int PH1, int i1, int i2, int j1, int j2)
    {
        int p = 0;
        if (j < j1) p = p2d(i, j, Nx, Ny, PH1);
        if (j >= j1 && i <= i1) p = p2d(i, j, Nx, Ny, PH1) - fsomae3(j - 1, i1, i2, j1, j2);
        if ((j >= j1 && i > i2) || j >= j2) p = p2d(i, j, Nx, Ny, PH1) - fsomae3(j, i1, i2, j1,
j2);//modificado j >= j1
        return p;
    }

    // FUNÇÕES PARA CAMADAS CPML NA DIREÇÃO Z
    //PARA MODO TMz
    public static int pztm(int k, int Nz, int k1, int k2)
    {
        int p = 0;
        if (k <= k1) p = k;
        if (k >= k2) p = k1 + (k - k2 + 1);
        if (k > Nz) p = k1 + (Nz - k2 + 1);
        return p;
    }
    //PARA MODO TEz
    public static int pzte(int k, int Nz, int k1, int k2)
    {
        int p = 0;
        if (k <= (k1 - 1)) p = k;
        if (k >= k2) p = k1 + k - k2;
        if (k > (Nz - 1)) p = k1 + Nz - 1 - k2;
        return p;
```

```
    }

    // FUNÇÃO PARA FONTE DO TIPO PULSO RICKER OU SENOIDE

    public static float fonte_1(int tempo1, float posicao, float cdtds1, int ppw, int tipo_fonte,
float fator_at)
    {
       double arg, arg2, funcao, periodo, cte_tempo, atenuacao;
       double pi = Math.PI;
       double tempo2 = (double)tempo1;
       double ppw2 = (double)ppw;

       if (tipo_fonte == 0)
       { // pulso ricker
          arg = pi * ((cdtds1 * tempo2 - posicao) / ppw2 - 1.0);
          arg2 = arg * arg;
          funcao = (1.0 - 2.0 * arg2) * Math.Exp(-arg2);
       }

       else
       {  // senóide com amortecimento inicial
          periodo = ppw2 / cdtds1; // periodo da senóide
          // ou tempo para o pico do pulso ricker
          cte_tempo = fator_at * periodo;
          atenuacao = 1.0 - Math.Exp(-1.0 * tempo2 / cte_tempo);
          funcao = atenuacao * Math.Sin(2.0 * pi * tempo2 * cdtds1 / ppw2);
       }
       return (float)funcao;
    }
    //**********************************

    // funções para obter i,j a partir de p para campo ez
    public static int ez_i(int p2, int Nx)
    {
       int p1, i, j;
       p1 = 2 * p2 - 1;
       j = p1 / Nx + 1;
       if ((p1 % Nx == 0) && (p1 >= Nx))
          j = j - 1;
       i = p1 - (j - 1) * Nx;
       return i;
    }

    public static int ez_j(int p2, int Nx)
    {
       int p1, i, j;
       p1 = 2 * p2 - 1;
       j = p1 / Nx + 1;
       if ((p1 % Nx == 0) && (p1 >= Nx))
          j = j - 1;
       i = p1 - (j - 1) * Nx;
       return j;
    }

    // FUNÇÕES PARA CALCULAR CONDUTIVIDADE NA CAMADA CPML
```

```
        public static double func_fia(double sigma, double cdtds, double imp0, double ds)
        {
            double a = ( Math.Exp(-sigma * cdtds * imp0 * ds) - 1.0 );
            return a;
        }
        public static double func_fib(double sigma, double cdtds, double imp0, double ds)
        {
            double b = ( Math.Exp(-sigma * cdtds * imp0 * ds) );
            return b;
        }
        //FINAL DOS MÉTODOS
    }
}
////////////////////////////////////////////////////////////////////////////////
```

ARQUIVO 6: Bx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace progCsharpe_Hex3d_matriz2d_float_2c
{
    class Bx
    {         // FUNÇÕES QUE AJUNDAM NA CRIAÇÃO DAS PMLs
        public static void func_by(int nxx, int nyy, int Nyc, double dax, double day, int binicial,
double tancpml,
            double dx, double dy, int[] by)
        {
            double razao, x, y;
            int razao2, tancpml2;
            tancpml2 = (int)(tancpml * 1.0e7);

            for(int a = 1; a < nxx + 2; a++)
                for(int j = 1; j < Nyc + 1; j++)
                {
                    x = (a - 1) * dx + dax * dx;
                    y = (j - 1) * dy + day * dy;
                    if(((a - 1) == 0) && (dax <= 1e-5))
                    {
                        by[a] = binicial;
                    }
                    else
                    {
                        razao = y / x;
                        razao2 = (int)(razao * 1.0e7);
                        if(razao2 >= tancpml2)
                        {
                            by[a] = j - 1;
                            break;
                        }
                    }
                }
```

```
        } //final func_by

        public static void func_bx(int nxx, int nyy, int Nxc, double dax, double day, int binicial,
double tancpml,
            double dx, double dy, int[] bx)
        {
            double razao, x, y;
            int razao2, tancpml2_inv;
            tancpml2_inv = (int)( 1.0e7/tancpml);
            for (int a = 1; a < nyy + 2; a++)
                for (int i = 1; i < Nxc + 1; i++)
                {
                    x = (i - 1) * dx + dax * dx;
                    y = (a - 1) * dy + day * dy;
                    if (((a - 1) == 0) && (day <= 1e-5))
                    {
                        bx[a] = binicial;                    }
                    else
                    {
                        razao = x / y;
                        razao2 = (int)(razao * 1.0e7);
                        if (razao2 > tancpml2_inv)
                        {
                            bx[a] = i - 1;
                            break;
                        }
                    }
                }
        } //final func_bx
    }
}
////////////////////////////////////////////////////////////////////////////
```

ARQUIVO 7: MATx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace progCsharpe_Hex3d_matriz2d_float_2c
{
    class MATx
    {
        public static void materialx(float[,] materiais, int n_materiais, float[] ch10, float[] ch20,
            float[] ch30, float[] chz0, float[] ce10, float[] ce20, float[] ce30, float[] cez0, float[] chze2,
            float[] cezh2, float[] cha, float[] chb, float[] cea, float[] ceb, float cdtds, float rdsdz, float
ds)
        {
            for(int i=0; i < n_materiais; i++)
            {
                double s = (double)materiais[i, 0];
                double er = (double)materiais[i, 1];
                double sm = (double)materiais[i, 2];
```

```
            double ur = (double)materiais[i, 3];
            double eo = 1.0e-9/(36.0*Math.PI);
            double uo = 4.0e-7 * Math.PI;
            double e = er * eo;
            double u = ur * uo;
            double co = 1.0 / Math.Sqrt(uo * eo);
            double dt = cdtds * ds / co;
            ch10[i] = (float) ( (2.0 * u - sm * dt) / (2.0 * u + sm * dt) );
            ch20[i] = ch10[i];
            ch30[i] = ch10[i];
            chz0[i] = ch10[i];
            ce10[i] = (float)((2.0 * e - s * dt) / (2.0 * e + s * dt));
            ce20[i] = ce10[i];
            ce30[i] = ce10[i];
            cez0[i] = ce10[i];
            chze2[i] = (float)( (4.0 * dt) / (3*ds*(2.0 * u + sm * dt))  );
            cezh2[i] = (float)((4.0 * dt) / (3 * ds * (2.0 * e + s * dt)));
            cha[i] = (float)((2.0 * dt) / ( ds * (2.0 * u + sm * dt)));
            cea[i] = (float)((2.0 * dt) / (ds * (2.0 * e + s * dt)));
            chb[i] = (float)((2.0 * rdsdz * dt) / (Math.Sqrt(3.0) * ds * (2.0 * u + sm * dt)));
            ceb[i] = (float)((2.0 * rdsdz * dt) / (Math.Sqrt(3.0) * ds * (2.0 * e + s * dt)));
        }
      }
   }
}
////////////////////////////////////////////////////////////////////////////////
```

ARQUIVO 8: OBJx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace progCsharpe_Hex3d_matriz2d_float_2c
{
   class OBJx
   {
      public static void objetox(float[,] objetos, int n_objetos, byte[,,] matgrade, float dx, float dy,
float dz)
      {
         for(int i=0; i < n_objetos; i++)
         {
            int tipo_objeto = (int) objetos[i, 0];
            if(tipo_objeto == 1) // PLACA RETANGULAR
            {
               int material1 = (int) objetos[i, 1];
               int x = (int) (objetos[i, 2]/dx);
               int y = (int)(objetos[i, 3] / dy);
               int z = (int)(objetos[i, 4] / dz);
               int lx = (int)(objetos[i, 5] / dx);
               int ly = (int)(objetos[i, 6] / dy);
               int lz = (int)(objetos[i, 7] / dz);
               for(int m = x; m < x + lx; m++)
```

```
        for(int n = y ; n < y + ly; n++)
            for(int p = z; p < z + lz; p++)
            {
                matgrade[m,n,p] = (byte) material1;
            }
}// FINAL OBJETO placa retangular

if (tipo_objeto == 2) // ESFERA
{
    int material1 = (int)objetos[i, 1];
    int x = (int)(objetos[i, 2] / dx);
    int y = (int)(objetos[i, 3] / dy);
    int z = (int)(objetos[i, 4] / dz);
    int rx = (int)(objetos[i, 5] / dx);
    int ry = (int)(objetos[i, 5] / dy);
    int rz = (int)(objetos[i, 5] / dz);
    float raio = (float)(objetos[i, 5]);
    int iniciox = x - rx;
    int finalx = x + rx;
    int inicioy = y - ry;
    int finaly = y + ry;
    int inicioz = z - rz;
    int finalz = z + rz;
    if (iniciox <= 2) iniciox = 2; // FALTA LIMITES COM Nx, Ny e Nz
    for (int m = iniciox; m < finalx; m++)
        for (int n = inicioy; n < finaly; n++)
            for (int p = inicioz; p < finalz; p++)
            {
                float di = (m - x) * dx;
                float dj = (n - y) * dy;
                float dk = (p - z) * dz;
                if((di*di + dj*dj + dk*dk) <= (raio*raio))
                {
                    matgrade[m, n, p] = (byte)material1;
                }

            }
}// FINAL OBJETO esfera

if (tipo_objeto == 3) // CILINDRO
{
    int material1 = (int)objetos[i, 1];
    int x = (int)(objetos[i, 2] / dx);
    int y = (int)(objetos[i, 3] / dy);
    int z = (int)(objetos[i, 4] / dz);
    int rx = (int)(objetos[i, 5] / dx);
    int ry = (int)(objetos[i, 5] / dy);
    int lz = (int)(objetos[i, 6] / dz);
    float raio = (float)(objetos[i, 5]);
    int iniciox = x - rx;
    int finalx = x + rx;
    int inicioy = y - ry;
    int finaly = y + ry;
    int inicioz = z ;
    int finalz = z + lz;
```

```
    if (iniciox <= 2) iniciox = 2;
    for (int m = iniciox; m < finalx; m++)
        for (int n = inicioy; n < finaly; n++)
            for (int p = inicioz; p < finalz; p++)
            {
                float di = (m - x) * dx;
                float dj = (n - y) * dy;

                if ((di * di + dj * dj) <= (raio * raio))
                {
                    matgrade[m, n, p] = (byte)material1;
                }

            }
}// FINAL OBJETO cilindro

if (tipo_objeto == 4) // PLACA RETANGULAR 2 (posição central com precisão)
{
    int material1 = (int)objetos[i, 1];
    int x = (int)(objetos[i, 2] );
    int y = (int)(objetos[i, 3] );
    int z = (int)(objetos[i, 4] );
    int lx = (int)(objetos[i, 5] / dx);
    int ly = (int)(objetos[i, 6] / dy);
    int lz = (int)(objetos[i, 7] / dz);
    ly = ly / 2;
    lz = lz / 2;

    for (int m = x; m < x + lx; m++)
        for (int n = y - ly; n < y + ly; n++)
            for (int p = z - lz; p < z + lz; p++)
            {
                matgrade[m, n, p] = (byte)material1;
            }
}// FINAL OBJETO placa retangular 2

if (tipo_objeto == 5) // ESFERA 2 (posição central com precisão)
{
    int material1 = (int)objetos[i, 1];
    int x = (int)(objetos[i, 2] );
    int y = (int)(objetos[i, 3] );
    int z = (int)(objetos[i, 4] );
    int rx = (int)(objetos[i, 5] / dx);
    int ry = (int)(objetos[i, 5] / dy);
    int rz = (int)(objetos[i, 5] / dz);
    float raio = (float)(objetos[i, 5]);
    int iniciox = x - rx;
    int finalx = x + rx;
    int inicioy = y - ry;
    int finaly = y + ry;
    int inicioz = z - rz;
    int finalz = z + rz;
    if (iniciox <= 2) iniciox = 2; // FALTA LIMITES COM Nx, Ny e Nz
    for (int m = iniciox; m < finalx; m++)
        for (int n = inicioy; n < finaly; n++)
```

```
                    for (int p = inicioz; p < finalz; p++)
                    {
                        float di = (m - x) * dx;
                        float dj = (n - y) * dy;
                        float dk = (p - z) * dz;
                        if ((di * di + dj * dj + dk * dk) <= (raio * raio))
                        {
                            matgrade[m, n, p] = (byte)material1;
                        }

                    }
                }// FINAL OBJETO esfera 2
            }
        }
    }
}
////////////////////////////////////////////////////////////////////////////
```

ARQUIVO 9: TE.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace progCsharpe_Hex3d_matriz2d_float_2c
{
    class TE
    {
        // ATUALIZANDO MODO TEz COM CAMADAS CPML

        public static void atualiza_tez_e1(float[] cea, float[] ceb,
            float[,] hz, float[,] h2, float[,] h3,
            float[,] e1, int Nx, int Ny, int Nz, int PH1, int PH3, int PH4, //novas variaveis
            int i1, int i2, int j1, int j2, int cinicial2, int CPML, int[,] mate1, float[] am2, float[] bm2,
            float[] amz, float[] bmz, float[,] pe1hz, float[,] pe1h2, float[,] pe1h3, byte[, ,] matgrade,
            float[] ce10, int[] matzte, int cinicial3, int k1, int k2, float[,] cdisp)
        {
            int p, s, n, nw, sw;
            // MODO TEz
            // campo e1
            for (int j = 1; j < Ny + 1; j++)
                for (int i = 1; i < Nx + 1; i++)
                    for (int k = 1; k < Nz; k++)
                        if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
                            (j % 2 == 0 && i % 2 == 1 && i <= Nx - 2))
                        {
                            p = Ax.p2d(i, j, Nx, Ny, PH4);//correção PH4
                            int gp = (int) matgrade[i, j, k]; //nova variavel
                            float f = cdisp[i, j];

                            s = Ax.p2a(i, j - 1, Nx, Ny, PH4);
                            n = Ax.p2a(i, j + 1, Nx, Ny, PH4);
                            nw = Ax.p2d(i, j, Nx, Ny, PH1);
```

```
                    sw = Ax.p2a(i, j - 1, Nx, Ny, PH3);
                    e1[p, k] = ce10[gp]*e1[p, k] -
                       f * cea[gp] * (hz[s, k] - hz[n, k]) -  // sem troca de sinal
                       f * ceb[gp] * (h2[nw, k + 1] + h3[sw, k + 1] -
                       h2[nw, k] - h3[sw, k]);

                    // camada CPML
                    if ((i < i1 || i >= i2 || j <= j1 || j >= j2) && CPML == 1)
                    {
                       int ip =  mate1[i, j] - cinicial2;
                       int xp =  Ax.yp2d(i, j, Nx, Ny, PH4, i1, i2, j1, j2);
                       pe1hz[xp, k] = bm2[ip] * pe1hz[xp, k] + am2[ip] * (hz[s, k] - hz[n, k]);
                       e1[p, k] = e1[p, k] - f * cea[gp] * pe1hz[xp, k];
                    }

                    if ((k < k1 || k >= k2) && CPML == 1)
                    {
                       int ip =  matzte[k] - cinicial3;
                       int zp = Ax.pzte(k, Nz, k1, k2);
                       pe1h2[p, zp] = bmz[ip] * pe1h2[p, zp] + amz[ip] * (h2[nw, k + 1] - h2[nw,
k]);
                       pe1h3[p, zp] = bmz[ip] * pe1h3[p, zp] + amz[ip] * (h3[sw, k + 1] - h3[sw,
k]);
                       e1[p, k] = e1[p, k] - f * ceb[gp] * (pe1h2[p, zp] + pe1h3[p, zp]);
                    }
                 }
        }

        public static void atualiza_tez_e2(float[] cea, float[] ceb,
           float[,] hz, float[,] h1, float[,] h3,
           float[,] e2, int Nx, int Ny, int Nz, int PH2, int PH3, int PH4, //novas variaveis
           int i1, int i2, int j1, int j2, int cinicial2, int CPML, int[,] mate23, float[] am2, float[] bm2,
           float[] amz, float[] bmz, float[,] pe2hz, float[,] pe2h1, float[,] pe2h3, byte[, ,] matgrade,
           float[] ce20, int[] matzte, int cinicial3, int k1, int k2, float[,] cdisp)
        {
           int p, se, nw, e, nw2;

           // campo e2
           for (int j = 1; j < Ny; j++)
              for (int i = 1; i < Nx + 1; i++)
                 for (int k = 1; k < Nz; k++)
                    if ((j % 2 == 1 && i % 2 == 1 && i <= Nx - 2) ||
                       (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
                    {
                       p = Ax.p2a(i, j, Nx, Ny, PH3);
                       int gp = (int) matgrade[i, j, k]; //nova variavel
                       float f = cdisp[i, j];

                       se = Ax.p2a(i, j, Nx, Ny, PH4);//correção PH4
                       nw = Ax.p2a(i - 1, j + 1, Nx, Ny, PH4);
                       e = Ax.p2a(i, j, Nx, Ny, PH3);
                       nw2 = Ax.p2c(i, j + 1, Nx, Ny, PH2);
                       e2[p, k] = ce20[gp]*e2[p, k] -
                          f * cea[gp] * (hz[se, k] - hz[nw, k]) -
                          f * ceb[gp] * (h3[e, k + 1] - h1[nw2, k + 1] -
```

```
                    h3[e, k] + h1[nw2, k]);

                // camada CPML
                if ((i <= i1 || i > i2 || j < j1 || j >= j2) && CPML == 1)
                {
                    int ip = mate23[i, j] - cinicial2;
                    int xp = Ax.zp2a(i, j, Nx, Ny, PH3, i1, i2, j1, j2);
                    pe2hz[xp, k] = bm2[ip] * pe2hz[xp, k] + am2[ip] * (hz[se, k] - hz[nw, k]);
                    e2[p, k] = e2[p, k] - f * cea[gp] * pe2hz[xp, k];
                }

                if ((k < k1 || k >= k2) && CPML == 1)
                {
                    int ip = matzte[k] - cinicial3;
                    int zp = Ax.pzte(k, Nz, k1, k2);
                    pe2h1[p, zp] = bmz[ip] * pe2h1[p, zp] + amz[ip] * (-h1[nw2, k + 1] + h1[nw2,
k]);
                    pe2h3[p, zp] = bmz[ip] * pe2h3[p, zp] + amz[ip] * (h3[e, k + 1] - h3[e, k]);
                    e2[p, k] = e2[p, k] - f * ceb[gp] * (pe2h1[p, zp] + pe2h3[p, zp]);
                }
            }
        }


        public static void atualiza_tez_e3(float[] cea, float[] ceb,
            float[,] hz, float[,] h1, float[,] h2,
            float[,] e3, int Nx, int Ny, int Nz, int PH1, int PH2, int PH3, int PH4, //novas variaveis
            int i1, int i2, int j1, int j2, int cinicial2, int CPML, int[,] mate23, float[] am2, float[] bm2,
            float[] amz, float[] bmz, float[,] pe3hz, float[,] pe3h1, float[,] pe3h2, byte[, ,] matgrade,
            float[] ce30, int[] matzte, int cinicial3, int k1, int k2, float[,] cdisp)
        {
            int p, ne, sw, e, sw2;

            // campo e3
            for (int j = 1; j < Ny; j++)
                for (int i = 1; i < Nx + 1; i++)
                    for (int k = 1; k < Nz; k++)
                        if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
                            (j % 2 == 0 && i % 2 == 1 && i <= Nx - 2))
                        {
                            p = Ax.p2d(i, j, Nx, Ny, PH1);
                            int gp = (int) matgrade[i, j, k]; //nova variavel
                            float f = cdisp[i, j];

                            ne = Ax.p2a(i, j + 1, Nx, Ny, PH4); //correção PH4
                            sw = Ax.p2a(i - 1, j, Nx, Ny, PH4);
                            e = Ax.p2d(i, j, Nx, Ny, PH1);
                            sw2 = Ax.p2c(i, j, Nx, Ny, PH2);
                            e3[p, k] = ce30[gp]*e3[p, k] -
                                f * cea[gp] * (hz[ne, k] - hz[sw, k]) -  // sem troca de sinal
                                f * ceb[gp] * (-1.0f * h2[e, k + 1] - h1[sw2, k + 1] +
                                h2[e, k] + h1[sw2, k]);

                            // camada CPML
                            if ((i <= i1 || i > i2 || j < j1 || j >= j2) && CPML == 1)
```

```
                    {
                        int ip = mate23[i, j] - cinicial2;
                        int xp = Ax.zp2d(i, j, Nx, Ny, PH1, i1, i2, j1, j2); //corrigido
                        pe3hz[xp, k] = bm2[ip] * pe3hz[xp, k] + am2[ip] * (hz[ne, k] - hz[sw, k]);
                        e3[p, k] = e3[p, k] - f * cea[gp] * pe3hz[xp, k];
                    }

                    if ((k < k1 || k >= k2) && CPML == 1)
                    {
                        int ip =  matzte[k] - cinicial3;
                        int zp = Ax.pzte(k, Nz, k1, k2);
                        pe3h1[p, zp] = bmz[ip] * pe3h1[p, zp] + amz[ip] * (-h1[sw2, k + 1] + h1[sw2,
k]);
                        pe3h2[p, zp] = bmz[ip] * pe3h2[p, zp] + amz[ip] * (-1.0f * h2[e, k + 1] +
h2[e, k]);
                        e3[p, k] = e3[p, k] - f * ceb[gp] * (pe3h1[p, zp] + pe3h2[p, zp]);
                    }
                }
    }


    public static void atualiza_tez_hz(float[] chze2,
        float[,] e1, float[,] e2, float[,] e3,
        float[,] hz, int Nx, int Ny, int Nz, int PH1, int PH3, int PH4, //novas variaveis
        int i1, int i2, int j1, int j2, int cinicial2, int CPML, int[,] mathz, float[] am2, float[] bm2,
        float[,] phze1, float[,] phze2, float[,] phze3, byte[, ,] matgrade, float[] chz0, float[,] cdisp)
    {
        int p, s, n, se, nw, ne, sw;

        // campo hz
        for (int j = 1; j < Ny + 1; j++)
            for (int i = 1; i < Nx + 1; i++)
                for (int k = 1; k < Nz + 1; k++) // verificar ??
                    if ((j % 2 == 1 && i % 2 == 1 && i <= Nx - 2) ||
                        (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
                    {
                        p = Ax.p2a(i, j, Nx, Ny, PH4);//correção PH4
                        int gp = (int) matgrade[i, j, k]; //nova variavel
                        float f = cdisp[i, j];

                        s = Ax.p2d(i, j - 1, Nx, Ny, PH4);
                        n = Ax.p2d(i, j + 1, Nx, Ny, PH4);
                        se = Ax.p2a(i + 1, j - 1, Nx, Ny, PH3);
                        nw = Ax.p2a(i, j, Nx, Ny, PH3);
                        ne = Ax.p2d(i + 1, j, Nx, Ny, PH1);
                        sw = Ax.p2d(i, j - 1, Nx, Ny, PH1);
                        hz[p, k] = chz0[gp]*hz[p, k] -
                            f * chze2[gp] * (e1[s, k] + e2[se, k] + e3[ne, k] -
                            e1[n, k] - e2[nw, k] - e3[sw, k]);

                        // camada CPML
                        if ((i <= i1 || i >= i2 || j <= j1 || j >= j2) && CPML == 1)
                        {
                            int ip =  mathz[i, j] - cinicial2;
                            int xp = Ax.yp2a(i, j, Nx, Ny, PH4, i1, i2, j1, j2);
```

```
                        phze1[xp, k] = bm2[ip] * phze1[xp, k] + am2[ip] * (e1[s, k] - e1[n, k]);
                        hz[p, k] = hz[p, k] - f * chze2[gp] * phze1[xp, k];
                    }
                    if ((i <= i1 || i >= i2 || j <= j1 || j >= j2) && CPML == 1)
                    {
                        int ip = mathz[i, j] - cinicial2;
                        int xp =  Ax.yp2a(i, j, Nx, Ny, PH4, i1, i2, j1, j2);
                        phze2[xp, k] = bm2[ip] * phze2[xp, k] + am2[ip] * (e2[se, k] - e2[nw, k]);
                        phze3[xp, k] = bm2[ip] * phze3[xp, k] + am2[ip] * (e3[ne, k] - e3[sw, k]);
                        hz[p, k] = hz[p, k] - f * chze2[gp] * (phze2[xp, k] + phze3[xp, k]);
                    }
                }
        }

        // FINAL DOS MÉTODOS
    }
}
////////////////////////////////////////////////////////////////////////////
```

ARQUIVO 10: TM.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace progCsharpe_Hex3d_matriz2d_float_2c
{
    class TM
    {
        // ATUALIZANDO MODO TMz COM CAMADAS CPML

        public static void atualiza_tmz_h1(float[] cha, float[] chb,
             float[,] ez, float[,] e2, float[,] e3,
            float[,] h1, int Nx, int Ny, int Nz, int PH1, int PH2, int PH3, int PEZ, //novas variaveis
            int i1, int i2, int j1, int j2, int cinicial, int CPML, int[,] math1, float[] am, float[] bm,
            float[] amz, float[] bmz, float[,] ph1ez, float[,] ph1e2, float[,] ph1e3, byte[,,] matgrade,
            float[] ch10, int[] matztm, int cinicial3, int k1, int k2, float[,] cdisp)
        {
            int p, s, n, se, ne;
            // MODO TMz
            // campo h1
            for (int j = 1; j < Ny + 1; j++)
                for (int i = 1; i < Nx + 1; i++)
                    for (int k = 1; k < Nz + 1; k++)
                        if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
                            (j % 2 == 0 && i % 2 == 1 && i <= Nx))
                        {
                            p = Ax.p2c(i, j, Nx, Ny, PH2);
                            int gp = (int) matgrade[i, j, k]; //nova variavel
                            float f = cdisp[i, j];

                            s = Ax.p2b(i, j - 1, Nx, Ny, PEZ);
                            n = Ax.p2b(i, j + 1, Nx, Ny, PEZ);
```

```
                se = Ax.p2a(i, j - 1, Nx, Ny, PH3);
                ne = Ax.p2d(i, j, Nx, Ny, PH1);
                h1[p, k] = ch10[gp]*h1[p, k] +
                  f * cha[gp] * (ez[s, k] - ez[n, k]) +
                   f * chb[gp] * (e2[se, k] + e3[ne, k] -
                   e2[se, k - 1] - e3[ne, k - 1]);

                // camada CPML
                if ((i <= i1 || i > i2 || j <= j1 || j >= j2) && CPML == 1)
                {
                    int ip =  math1[i, j] - cinicial;
                    int xp = Ax.xp2c(i, j, Nx, Ny, PH2, i1, i2, j1, j2);
                    ph1ez[xp, k] = bm[ip] * ph1ez[xp, k] + am[ip] * (ez[s, k] - ez[n, k]);
                    h1[p, k] = h1[p, k] + f * cha[gp] * ph1ez[xp, k];
                }

                if ((k <= k1 || k >= k2) && CPML == 1)
                {
                    int ip = matztm[k] - cinicial3;
                    int zp = Ax.pztm(k, Nz, k1, k2);
                    ph1e2[p, zp] = bmz[ip] * ph1e2[p, zp] + amz[ip] * (e2[se, k] - e2[se, k - 1]);
                    ph1e3[p, zp] = bmz[ip] * ph1e3[p, zp] + amz[ip] * (e3[ne, k] - e3[ne, k - 1]);
                    h1[p, k] = h1[p, k] + f * chb[gp] * (ph1e2[p, zp] + ph1e3[p, zp]);
                }

            }
}


public static void atualiza_tmz_h2(float[] cha, float[] chb,
    float[,] ez, float[,] e1, float[,] e3,
   float[,] h2, int Nx, int Ny, int Nz, int PH1, int PEZ, int PH4, //novas variaveis
   int i1, int i2, int j1, int j2, int cinicial, int CPML, int[,] math23, float[] am, float[] bm,
   float[] amz, float[] bmz, float[,] ph2ez, float[,] ph2e1, float[,] ph2e3, byte[, ,] matgrade,
   float[] ch20, int[] matztm, int cinicial3, int k1, int k2, float[,] cdisp)
{
   int p, se, nw, se2, w;

   // campo h2
   for (int j = 1; j < Ny; j++)
      for (int i = 1; i < Nx + 1; i++)
         for (int k = 1; k < Nz + 1; k++)
            if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
               (j % 2 == 0 && i % 2 == 1 && i <= Nx - 2))
            {
               p = Ax.p2d(i, j, Nx, Ny, PH1);
               int gp = (int) matgrade[i, j, k]; //nova variavel
               float f = cdisp[i, j];

               se = Ax.p2b(i + 1, j, Nx, Ny, PEZ);
               nw = Ax.p2b(i, j + 1, Nx, Ny, PEZ);
               se2 = Ax.p2d(i, j, Nx, Ny, PH4); //correção PH4
               w = Ax.p2d(i, j, Nx, Ny, PH1);

               h2[p, k] = ch20[gp]*h2[p, k] +
```

```
                    f * cha[gp] * (ez[se, k] - ez[nw, k]) + // sem troca de sinal
                    f * chb[gp] * (e3[w, k] - e1[se2, k] -
                    e3[w, k - 1] + e1[se2, k - 1]);

                // camada CPML
                if ((i <= i1 || i >= i2 || j < j1 || j >= j2) && CPML == 1)
                {
                    int ip = math23[i, j] - cinicial;
                    int xp = Ax.xp2d(i, j, Nx, Ny, PH1, i1, i2, j1, j2);
                    ph2ez[xp, k] = bm[ip] * ph2ez[xp, k] + am[ip] * (ez[se, k] - ez[nw, k]);
                    h2[p, k] = h2[p, k] + f * cha[gp] * ph2ez[xp, k];
                }

                if ((k <= k1 || k >= k2) && CPML == 1)
                {
                    int ip = matztm[k] - cinicial3;
                    int zp = Ax.pztm(k, Nz, k1, k2);
                    ph2e1[p, zp] = bmz[ip] * ph2e1[p, zp] + amz[ip] * (-e1[se2, k] + e1[se2, k -
1]);
                    ph2e3[p, zp] = bmz[ip] * ph2e3[p, zp] + amz[ip] * (e3[w, k] - e3[w, k - 1]);
                    h2[p, k] = h2[p, k] + f * chb[gp] * (ph2e1[p, zp] + ph2e3[p, zp]);
                }
            }
        }

        public static void atualiza_tmz_h3(float[] cha, float[] chb,
            float[,] ez, float[,] e1, float[,] e2,
            float[,] h3, int Nx, int Ny, int Nz, int PH1, int PH3, int PEZ, int PH4, //novas variaveis
            int i1, int i2, int j1, int j2, int cinicial, int CPML, int[,] math23, float[] am, float[] bm,
            float[] amz, float[] bmz, float[,] ph3ez, float[,] ph3e1, float[,] ph3e2, byte[, ,] matgrade,
            float[] ch30, int[] matztm, int cinicial3, int k1, int k2, float[,] cdisp)
        {
            int p, ne, sw, ne2, w;

            // campo h3
            for (int j = 1; j < Ny; j++)
                for (int i = 1; i < Nx + 1; i++)
                    for (int k = 1; k < Nz + 1; k++)
                        if ((j % 2 == 1 && i % 2 == 1 && i <= Nx - 2) ||
                            (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
                        {
                            p = Ax.p2a(i, j, Nx, Ny, PH3);
                            int gp = (int) matgrade[i, j, k]; //nova variavel
                            float f = cdisp[i, j];

                            ne = Ax.p2b(i + 1, j + 1, Nx, Ny, PEZ);
                            sw = Ax.p2b(i, j, Nx, Ny, PEZ);
                            ne2 = Ax.p2d(i, j + 1, Nx, Ny, PH4);//correção PH4
                            w = Ax.p2a(i, j, Nx, Ny, PH3);
                            h3[p, k] = ch30[gp]*h3[p, k] +
                                f * cha[gp] * (ez[ne, k] - ez[sw, k]) +   //erro  sinal  corrigido
                                f * chb[gp] * (-1.0f * e1[ne2, k] - e2[w, k] +
                                e1[ne2, k - 1] + e2[w, k - 1]);

                            // camada CPML
```

```
                    if ((i <= i1 || i >= i2 || j < j1 || j >= j2) && CPML == 1)
                    {
                        int ip = math23[i, j] - cinicial;
                        int xp = Ax.xp2a(i, j, Nx, Ny, PH3, i1, i2, j1, j2);
                        ph3ez[xp, k] = bm[ip] * ph3ez[xp, k] + am[ip] * (ez[ne, k] - ez[sw, k]);
                        h3[p, k] = h3[p, k] + f * cha[gp] * ph3ez[xp, k];
                    }

                    if ((k <= k1 || k >= k2) && CPML == 1)
                    {
                        int ip = matztm[k] - cinicial3;
                        int zp = Ax.pztm(k, Nz, k1, k2);
                        ph3e1[p, zp] = bmz[ip] * ph3e1[p, zp] + amz[ip] * (-1.0f * e1[ne2, k] +
e1[ne2, k - 1]);
                        ph3e2[p, zp] = bmz[ip] * ph3e2[p, zp] + amz[ip] * (-e2[w, k] + e2[w, k - 1]);
                        h3[p, k] = h3[p, k] + f * chb[gp] * (ph3e1[p, zp] + ph3e2[p, zp]);
                    }
                }
    }

    public static void atualiza_tmz_ez(float[] cezh2,
        float[,] h1, float[,] h2, float[,] h3,
        float[,] ez, int Nx, int Ny, int Nz, int PH1, int PH2, int PH3, int PEZ, //novas variaveis
        int i1, int i2, int j1, int j2, int cinicial, int CPML, int[,] matez, float[] am, float[] bm,
        float[,] pezh1, float[,] pezh2, float[,] pezh3, byte[, ,] matgrade, float[] cez0, float[,] cdisp)
    {
        int p, s, n, se, nw, ne, sw;

        // campo ez
        for (int j = 1; j < Ny + 1; j++)
            for (int i = 1; i < Nx + 1; i++)
                for (int k = 1; k < Nz + 1; k++)
                    if ((j % 2 == 1 && i % 2 == 1 && i <= Nx) ||
                        (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
                    {
                        p = Ax.p2b(i, j, Nx, Ny, PEZ);
                        int gp = (int) matgrade[i, j, k]; //nova variavel
                        float f = cdisp[i, j];

                        s = Ax.p2c(i, j - 1, Nx, Ny, PH2);
                        n = Ax.p2c(i, j + 1, Nx, Ny, PH2);
                        se = Ax.p2d(i, j - 1, Nx, Ny, PH1);
                        nw = Ax.p2d(i - 1, j, Nx, Ny, PH1);
                        ne = Ax.p2a(i, j, Nx, Ny, PH3);
                        sw = Ax.p2a(i - 1, j - 1, Nx, Ny, PH3);
                        ez[p, k] = cez0[gp]*ez[p, k] +
                            f * cezh2[gp] * (h1[s, k] + h2[se, k] + h3[ne, k] -
                            h1[n, k] - h2[nw, k] - h3[sw, k]);

                        // camada CPML
                        if ((i <= i1 || i > i2 || j <= j1 || j >= j2) && CPML == 1)
                        {
                            int ip = matez[i, j] - cinicial;
                            int xp = Ax.xp2b(i, j, Nx, Ny, PEZ, i1, i2, j1, j2);
                            pezh1[xp, k] = bm[ip] * pezh1[xp, k] + am[ip] * (h1[s, k] - h1[n, k]);
```

```
                        ez[p, k] = ez[p, k] + f * cezh2[gp] * pezh1[xp, k];
                    }
                    if ((i <= i1 || i > i2 || j <= j1 || j >= j2) && CPML == 1)
                    {
                        int ip = matez[i, j] - cinicial;
                        int xp = Ax.xp2b(i, j, Nx, Ny, PEZ, i1, i2, j1, j2);
                        pezh2[xp, k] = bm[ip] * pezh2[xp, k] + am[ip] * (h2[se, k] - h2[nw, k]);
                        pezh3[xp, k] = bm[ip] * pezh3[xp, k] + am[ip] * (h3[ne, k] - h3[sw, k]);
                        ez[p, k] = ez[p, k] + f * cezh2[gp] * (pezh2[xp, k] + pezh3[xp, k]);
                    }
                }
        }

        // FINAL DOS METODOS
    }
}
////////////////////////////////////////////////////////////////////////////
```

ARQUIVO 11: TS.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace progCsharpe_Hex3d_matriz2d_float_2c
{
    class TS
    {
        // ATUALIZA REGIÃO TFSF PARA H1, H2 E H3
        public static void atualiza_tfsf_h(float[,] h1, float[,] h2, float[,] h3, float[] cha, float[] ez1,
byte[,,] matgrade,
            int iniciox, int finalx, int inicioy,int finaly, int inicioz, int finalz, int Nx, int Ny, int PH1, int
PH2, int PH4)
        {
            int i, j, p, gp;
            float kxc = 1.0f;
            // corrige H2, H3 e H1 no lado esquerdo (campo espalhado)

    for(int k = inicioz; k < finalz + 1; k++)
      for(int jc = inicioy; jc < finaly+1; jc = jc + 3)
      {
          if (jc % 2 == 0)
          {
            i = iniciox - 2;
            j = jc - 1; // H2 abaixo
            if ((j%2==1 && i%2==0 && i<=Nx-1) ||
               (j%2==0 && i%2==1 && i<=Nx-2))
            {
                p = Ax.p2d(i,j,Nx,Ny,PH1);
                gp = (int) matgrade[i,j,k];
                h2[p,k] -= cha[gp]*kxc*ez1[i+1];
            }
            j = jc + 1; // H2 acima
```

```
    if ((j%2==1 && i%2==0 && i<=Nx-1) ||
       (j%2==0 && i%2==1 && i<=Nx-2))
    {
          p = Ax.p2d(i,j,Nx,Ny,PH1);
          gp = (int) matgrade[i,j,k];
        h2[p,k] -= cha[gp]*kxc*ez1[i+1];
    }
    j = jc - 2 ;// H3 abaixo
    if ((j%2==1 && i%2==1 && i<=Nx-2) ||
       (j%2==0 && i%2==0 && i<=Nx-1))
    {
        p = Ax.p2a(i,j,Nx,Ny,PH1);
        gp = (int) matgrade[i,j,k];
        h3[p,k] -= cha[gp]*kxc*ez1[i+1];
    }
    j = jc ;   // H3 acima
    if ((j%2==1 && i%2==1 && i<=Nx-2) ||
       (j%2==0 && i%2==0 && i<=Nx-1))
    {
        p = Ax.p2a(i,j,Nx,Ny,PH1);
        gp = (int) matgrade[i,j,k];
        h3[p,k] -= cha[gp]*kxc*ez1[i+1];
    }
    i = iniciox - 1;
    j = jc - 2 ; // H1 abaixo
    if ((j%2==1 && i%2==0 && i<=Nx-1) ||
       (j%2==0 && i%2==1 && i<=Nx))
    {
        p = Ax.p2c(i,j,Nx,Ny,PH2);
        gp = (int) matgrade[i,j,k];
        h1[p,k] += cha[gp]*kxc*ez1[i];
    }
    j = jc + 2;  // H1 acima
    if ((j%2==1 && i%2==0 && i<=Nx-1) ||
       (j%2==0 && i%2==1 && i<=Nx))
    {
        p = Ax.p2c(i,j,Nx,Ny,PH2);
        gp = (int) matgrade[i,j,k];
        h1[p,k] -= cha[gp]*kxc*ez1[i];
    }

  }
if (jc % 2 == 1)
{
    i = iniciox - 1;
    j = jc - 1; // H2 abaixo e interno
    if ((j%2==1 && i%2==0 && i<=Nx-1) ||
       (j%2==0 && i%2==1 && i<=Nx-2))
    {
        p = Ax.p2d(i,j,Nx,Ny,PH1);
        gp = (int) matgrade[i,j,k];
        h2[p,k] -= cha[gp]*kxc*ez1[i+1];
    }
    j = jc;    // H3 acima e interno
    if ((j%2==1 && i%2==1 && i<=Nx-2) ||
```

```
          (j%2==0 && i%2==0 && i<=Nx-1))
        {
           p = Ax.p2a(i,j,Nx,Ny,PH1);
           gp = (int) matgrade[i,j,k];
           h3[p,k] -= cha[gp]*kxc*ez1[i+1];
        }
     }
  }

// corrige H2, H3 e H1 no lado direito (campo espalhado)
for(int k = inicioz; k < finalz + 1; k++)
  for(int jc = inicioy; jc < finaly+1; jc = jc + 3)
  {
    if (jc % 2 == 0)
    {
        i = finalx + 1;
        j = jc - 2; // H2 abaixo
        if ((j%2==1 && i%2==0 && i<=Nx-1) ||
           (j%2==0 && i%2==1 && i<=Nx-2))
        {
           p = Ax.p2d(i,j,Nx,Ny,PH1);
           gp = (int) matgrade[i,j,k];
           h2[p,k] += cha[gp]*kxc*ez1[i];
        }
        j = jc;  // H2 acima
        if ((j%2==1 && i%2==0 && i<=Nx-1) ||
           (j%2==0 && i%2==1 && i<=Nx-2))
        {
           p = Ax.p2d(i,j,Nx,Ny,PH1);
           gp = (int) matgrade[i,j,k];
           h2[p,k] += cha[gp]*kxc*ez1[i];
        }
        j = jc - 1; // H3 abaixo
        if ((j%2==1 && i%2==1 && i<=Nx-2) ||
           (j%2==0 && i%2==0 && i<=Nx-1))
        {
           p = Ax.p2a(i,j,Nx,Ny,PH1);
           gp = (int) matgrade[i,j,k];
           h3[p,k] += cha[gp]*kxc*ez1[i];
        }
        j = jc + 1; // H3 acima
        if ((j%2==1 && i%2==1 && i<=Nx-2) ||
           (j%2==0 && i%2==0 && i<=Nx-1))
        {
           p = Ax.p2a(i,j,Nx,Ny,PH1);
           gp = (int) matgrade[i,j,k];
           h3[p,k] += cha[gp]*kxc*ez1[i];
        }
        j = jc - 2 ; // H1 abaixo
        if ((j%2==1 && i%2==0 && i<=Nx-1) ||
           (j%2==0 && i%2==1 && i<=Nx))
        {
           p = Ax.p2c(i,j,Nx,Ny,PH2);
           gp = (int) matgrade[i,j,k];
           h1[p,k] += cha[gp]*kxc*ez1[i];
```

```
        }
        j = jc + 2;  // H1 acima
        if ((j%2==1 && i%2==0 && i<=Nx-1) ||
           (j%2==0 && i%2==1 && i<=Nx))
        {
            p = Ax.p2c(i,j,Nx,Ny,PH2);
            gp = (int) matgrade[i,j,k];
            h1[p,k] -= cha[gp]*kxc*ez1[i];
        }
    }
    if (jc % 2 == 1)
    {
        i = finalx;
        j = jc;    // H2 acima e interno
        if ((j%2==1 && i%2==0 && i<=Nx-1) ||
           (j%2==0 && i%2==1 && i<=Nx-2))
        {
            p = Ax.p2d(i,j,Nx,Ny,PH1);
            gp = (int) matgrade[i,j,k];
            h2[p,k] += cha[gp]*kxc*ez1[i];
        }
        j = jc - 1; // H3 abaixo e interno
        if ((j%2==1 && i%2==1 && i<=Nx-2) ||
           (j%2==0 && i%2==0 && i<=Nx-1))
        {
            p = Ax.p2a(i,j,Nx,Ny,PH1);
            gp = (int) matgrade[i,j,k];
            h3[p,k] += cha[gp]*kxc*ez1[i];
        }
     }
  }
// corrige H2, H3 e H1 na base (campo espalhado)
for(int k = inicioz; k < finalz + 1; k++)
 for(int ic = inicix; ic < finalx + 1; ic++)
 {
    if (ic % 2 == 0)
    {
        j = inicioy - 3;
        i = ic - 1;  // H3 a esquerda
        if ((j%2==1 && i%2==1 && i<=Nx-2) ||
           (j%2==0 && i%2==0 && i<=Nx-1))
        {
            p = Ax.p2a(i,j,Nx,Ny,PH1);
            gp = (int) matgrade[i,j,k];
            h3[p,k] -= cha[gp]*kxc*ez1[i+1];
        }
        i = ic;   // H2 a direita
        if ((j%2==1 && i%2==0 && i<=Nx-1) ||
           (j%2==0 && i%2==1 && i<=Nx-2))
        {
            p = Ax.p2d(i,j,Nx,Ny,PH1);
            gp = (int) matgrade[i,j,k];
            h2[p,k] += cha[gp]*kxc*ez1[i];
        }
        i = ic;    // H1 mais abaixo
```

```
        if ((j%2==1 && i%2==0 && i<=Nx-1) ||
           (j%2==0 && i%2==1 && i<=Nx))
        {
            p = Ax.p2c(i,j,Nx,Ny,PH2);
            gp = (int) matgrade[i,j,k];
            h1[p,k] += cha[gp]*kxc*ez1[i];
        }
    }
    if (ic % 2 == 1)
    {
        j = inicioy - 2;
        i = ic;   // H1 menos abaixo
        if ((j%2==1 && i%2==0 && i<=Nx-1) ||
           (j%2==0 && i%2==1 && i<=Nx))
        {
            p = Ax.p2c(i,j,Nx,Ny,PH2);
            gp = (int) matgrade[i,j,k];
            h1[p,k] += cha[gp]*kxc*ez1[i];
        }
    }

}
 // corrige H2, H3 e H1 no topo (campo espalhado)
for(int k = inicioz; k < finalz + 1; k++)
    for(int ic = iniciox; ic < finalx + 1; ic++)
    {
    if (ic % 2 == 0)
    {
        j = finaly + 2;
        i = ic;     // H3 a direita
        if ((j%2==1 && i%2==1 && i<=Nx-2) ||
           (j%2==0 && i%2==0 && i<=Nx-1))
        {
            p = Ax.p2a(i,j,Nx,Ny,PH1);
            gp = (int) matgrade[i,j,k];
            h3[p,k] += cha[gp]*kxc*ez1[i];
        }
        i = ic - 1;  // H2 a esquerda
        if ((j%2==1 && i%2==0 && i<=Nx-1) ||
           (j%2==0 && i%2==1 && i<=Nx-2))
        {
            p = Ax.p2d(i,j,Nx,Ny,PH1);
            gp = (int) matgrade[i,j,k];
            h2[p,k] -= cha[gp]*kxc*ez1[i+1];
        }
        j = finaly + 3;
        i = ic;     // H1 mais acima
        if ((j%2==1 && i%2==0 && i<=Nx-1) ||
           (j%2==0 && i%2==1 && i<=Nx))
        {
            p = Ax.p2c(i,j,Nx,Ny,PH2);
            gp = (int) matgrade[i,j,k];
            h1[p,k] -= cha[gp]*kxc*ez1[i];
        }
```

```
        }
        if (ic % 2 == 1)
        {
            j = finaly + 2;
            i= ic ;     // H1 menos acima
            if ((j%2==1 && i%2==0 && i<=Nx-1) ||
               (j%2==0 && i%2==1 && i<=Nx))
            {
                p = Ax.p2c(i,j,Nx,Ny,PH2);
                gp = (int) matgrade[i,j,k];
                h1[p,k] -= cha[gp]*kxc*ez1[i];
            }

        }
       }
      }
       // ATUALIZA REGIÃO TFSF PARA Ez
       public static void atualiza_tfsf_ez(float[,] ez, float[] cezh2, float[] hy1, byte[,,] matgrade,
           int iniciox, int finalx, int inicioy,int finaly, int inicioz, int finalz, int Nx, int Ny, int PEZ)
       {
           // corrige campo Ez no lado esquerdo (campo total)
           int i = 1, j, p, gp;
           float c20 = (float) (Math.Cos(Math.PI / 6.0));// correção dos campos H2 e H3 em
relação a H1y
           float c30 = c20;

           for(int k = inicioz; k < finalz + 1; k++)
            for(int jc = inicioy; jc < finaly+1; jc = jc + 3)
            {
                if (jc % 2 == 0)
                {
                    i = iniciox - 1;
                    j = jc - 1;  // Ez abaixo
                    if ((j % 2 == 1 && i % 2 == 1 && i <= Nx) ||
                       (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
                    {
                        p = Ax.p2b(i, j, Nx, Ny, PEZ);
                        gp = (int)matgrade[i, j, k];
                        ez[p, k] -= cezh2[gp] * (c20 * hy1[i - 1] + c30 * hy1[i - 1]);
                    }
                    j = jc + 1;  // Ez acima
                    if ((j % 2 == 1 && i % 2 == 1 && i <= Nx) ||
                       (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
                    {
                        p = Ax.p2b(i, j, Nx, Ny, PEZ);
                        gp = (int)matgrade[i, j, k];
                        ez[p, k] -= cezh2[gp] * (c20 * hy1[i - 1] + c30 * hy1[i - 1]);
                    }
                }

                if (jc % 2 == 1)
                {
                    i = iniciox;
                    j = jc - 1;  // Ez abaixo e interno
                    if ((j%2==1 && i%2==1 && i<=Nx) ||
```

```
          (j%2==0 && i%2==0 && i<=Nx-1))
        {
           p = Ax.p2b(i,j,Nx,Ny,PEZ);
           gp = (int) matgrade[i,j,k];
           ez[p,k] -= cezh2[gp]*c20*hy1[i-1];
        }
        j = jc + 1;  // Ez acima e interno
        if ((j%2==1 && i%2==1 && i<=Nx) ||
          (j%2==0 && i%2==0 && i<=Nx-1))
        {
           p = Ax.p2b(i,j,Nx,Ny,PEZ);
           gp = (int) matgrade[i,j,k];
           ez[p,k] -= cezh2[gp]*c30*hy1[i-1];
        }

    }
  }
// corrige campo Ez no lado direito (campo total)
for(int k = inicioz; k < finalz + 1; k++)
 for(int jc = inicioy; jc < finaly+1; jc = jc + 3)
 {

     if (jc % 2 == 0)
     {
        i = finalx + 1;
        j = jc - 1;  // Ez abaixo
        if ((j % 2 == 1 && i % 2 == 1 && i <= Nx) ||
           (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
        {
           p = Ax.p2b(i, j, Nx, Ny, PEZ);
           gp = (int)matgrade[i, j, k];
           ez[p, k] += cezh2[gp] * (c20 * hy1[i] + c30 * hy1[i]);
        }

        j = jc + 1; // Ez acima
        if ((j % 2 == 1 && i % 2 == 1 && i <= Nx) ||
           (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
        {
           p = Ax.p2b(i, j, Nx, Ny, PEZ);
           gp = (int)matgrade[i, j, k];
           ez[p, k] += cezh2[gp] * (c20 * hy1[i] + c30 * hy1[i]);
        }
     }

    if (jc % 2 == 1)
    {
        i = finalx;
        j = jc - 1;  // Ez abaixo e interno
        if ((j%2==1 && i%2==1 && i<=Nx) ||
          (j%2==0 && i%2==0 && i<=Nx-1))
        {
           p = Ax.p2b(i,j,Nx,Ny,PEZ);
           gp = (int) matgrade[i,j,k];
           ez[p,k] += cezh2[gp]*c30*hy1[i];
        }
```

```
                j = jc + 1;  // Ez acima e interno
                if ((j%2==1 && i%2==1 && i<=Nx) ||
                   (j%2==0 && i%2==0 && i<=Nx-1))
                {
                    p = Ax.p2b(i,j,Nx,Ny,PEZ);
                    gp = (int) matgrade[i,j,k];
                    ez[p,k] += cezh2[gp]*c20*hy1[i];
                }

            }
          }
          // corrige Ez na base (campo total)
          for(int k = inicioz; k < finalz + 1; k++)
          {
          j = inicioy - 2;
          for( i = iniciox; i < finalx+1; i = i + 2)
          {
             if ((j%2==1 && i%2==1 && i<=Nx) ||
                (j%2==0 && i%2==0 && i<=Nx-1))
             {
                p = Ax.p2b(i,j,Nx,Ny,PEZ);
                gp = (int)matgrade[i, j, k];
                ez[p,k] += cezh2[gp]*(c20*hy1[i] - c30*hy1[i-1]);
             }

          }
          // corrige Ez no topo (campo total)
          j = finaly + 2;
          for(i = iniciox; i < finalx+1; i = i + 2)
          {
             if ((j%2==1 && i%2==1 && i<=Nx) ||
                (j%2==0 && i%2==0 && i<=Nx-1))
             {
                p = Ax.p2b(i,j,Nx,Ny,PEZ);
                gp = (int)matgrade[i, j, k];
                ez[p, k] += cezh2[gp] * (c30 * hy1[i] - c20 * hy1[i - 1]);
             }

          }
        }
      }
      // ATUALIZA REGIÃO TFSF PARA E1, E2 E E3
      public static void atualiza_tfsf_e123(float[,] e1, float[,] e2, float[,] e3, float[] ceb, float[] hy1,
byte[, ,] matgrade,
          int iniciox, int finalx, int inicioy, int finaly, int inicioz, int finalz, int Nx, int Ny, int PH1, int
PH3, int PH4)
      {
          int i, j, p, p2, gp;
          float c20 = (float)(Math.Cos(Math.PI / 6.0));// correção dos campos H2 e H3 em relação
a H1y
          float c30 = c20;
          int k1 = inicioz - 1;
          int k2 = finalz;
          for(int jc = inicioy; jc < finaly+1; jc = jc + 3)
```

```
{
    if (jc % 2 == 0)
    {
        i = iniciox - 1; // lado esquerdo
        j = jc - 2; // e1 e e3 abaixo
        if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
            (j % 2 == 0 && i % 2 == 1 && i <= Nx - 2))
        {
            p = Ax.p2d(i, j, Nx, Ny, PH4);
            p2 = Ax.p2d(i, j, Nx, Ny, PH1);
            gp = (int)matgrade[i, j, k1];
            e1[p, k1] += ceb[gp] * c20 * hy1[i];
            e3[p2, k1] -= ceb[gp] * c20 * hy1[i];
            gp = (int)matgrade[i, j, k2];
            e1[p, k2] -= ceb[gp] * c20 * hy1[i];
            e3[p2, k2] += ceb[gp] * c20 * hy1[i];
        }
        j = jc - 1; // e2 abaixo
        if ((j % 2 == 1 && i % 2 == 1 && i <= Nx - 2) ||
            (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
        {
            p = Ax.p2a(i, j, Nx, Ny, PH3);
            gp = (int)matgrade[i, j, k1];
            e2[p, k1] += ceb[gp] * c20 * hy1[i];
            gp = (int)matgrade[i, j, k2];
            e2[p, k2] -= ceb[gp] * c20 * hy1[i];
        }
        j = jc; // e1 e e3 acima
        if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
            (j % 2 == 0 && i % 2 == 1 && i <= Nx - 2))
        {
            p = Ax.p2d(i, j, Nx, Ny, PH4);
            p2 = Ax.p2d(i, j, Nx, Ny, PH1);
            gp = (int)matgrade[i, j, k1];
            e1[p, k1] += ceb[gp] *2.0f* c20 * hy1[i];
            e3[p2, k1] -= ceb[gp] * c20 * hy1[i];
            gp = (int)matgrade[i, j, k2];
            e1[p, k2] -= ceb[gp] * 2.0f * c20 * hy1[i];
            e3[p2, k2] += ceb[gp] * c20 * hy1[i];
        }
        j = jc + 1; // e2 acima
        if ((j % 2 == 1 && i % 2 == 1 && i <= Nx - 2) ||
            (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
        {
            p = Ax.p2a(i, j, Nx, Ny, PH3);
            gp = (int)matgrade[i, j, k1];
            e2[p, k1] += ceb[gp] * c20 * hy1[i];
            gp = (int)matgrade[i, j, k2];
            e2[p, k2] -= ceb[gp] * c20 * hy1[i];
        }
        j = jc + 2; // e1 acima
        if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
            (j % 2 == 0 && i % 2 == 1 && i <= Nx - 2))
        {
            p = Ax.p2d(i, j, Nx, Ny, PH4);
```

```
            gp = (int)matgrade[i, j, k1];
            e1[p, k1] += ceb[gp] * c20 * hy1[i];
            gp = (int)matgrade[i, j, k2];
            e1[p, k2] -= ceb[gp] * c20 * hy1[i];
        }
    }
}
for (int jc = inicioy; jc < finaly + 1; jc = jc + 3)
{
    if (jc % 2 == 0)
    {
        i = finalx; // lado direito
        j = jc - 2; // e2 abaixo
        if ((j % 2 == 1 && i % 2 == 1 && i <= Nx - 2) ||
            (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
        {
            p = Ax.p2a(i, j, Nx, Ny, PH3);
            gp = (int)matgrade[i, j, k1];
            e2[p, k1] += ceb[gp] * c20 * hy1[i];
            gp = (int)matgrade[i, j, k2];
            e2[p, k2] -= ceb[gp] * c20 * hy1[i];
        }
        j = jc - 1; // e1 e e3 abaixo
        if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
            (j % 2 == 0 && i % 2 == 1 && i <= Nx - 2))
        {
            p = Ax.p2d(i, j, Nx, Ny, PH4);
            p2 = Ax.p2d(i, j, Nx, Ny, PH1);
            gp = (int)matgrade[i, j, k1];
            e1[p, k1] += ceb[gp] *2.0f* c20 * hy1[i];
            e3[p2, k1] -= ceb[gp] * c20 * hy1[i];
            gp = (int)matgrade[i, j, k2];
            e1[p, k2] -= ceb[gp]*2.0f * c20 * hy1[i];
            e3[p2, k2] += ceb[gp] * c20 * hy1[i];
        }

        j = jc; // e2 acima
        if ((j % 2 == 1 && i % 2 == 1 && i <= Nx - 2) ||
            (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
        {
            p = Ax.p2a(i, j, Nx, Ny, PH3);
            gp = (int)matgrade[i, j, k1];
            e2[p, k1] += ceb[gp] * c20 * hy1[i];
            gp = (int)matgrade[i, j, k2];
            e2[p, k2] -= ceb[gp] * c20 * hy1[i];
        }
        j = jc + 1; // e1 e e3 acima
        if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
            (j % 2 == 0 && i % 2 == 1 && i <= Nx - 2))
        {
            p = Ax.p2d(i, j, Nx, Ny, PH4);
            p2 = Ax.p2d(i, j, Nx, Ny, PH1);
            gp = (int)matgrade[i, j, k1];
            e1[p, k1] += ceb[gp] * 2.0f * c20 * hy1[i];
            e3[p2, k1] -= ceb[gp] * c20 * hy1[i];
```

```
            gp = (int)matgrade[i, j, k2];
            e1[p, k2] -= ceb[gp] * 2.0f * c20 * hy1[i];
            e3[p2, k2] += ceb[gp] * c20 * hy1[i];
        }

    }
}
// campo espalhado e1 no centro
for(i = iniciox; i < finalx; i++)
{
    if (i % 2 == 0)
    {
        for(j = inicioy - 1; j < finaly + 2; j++)
        {
            if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
            (j % 2 == 0 && i % 2 == 1 && i <= Nx - 2))
            {
                p = Ax.p2d(i, j, Nx, Ny, PH4);
                gp = (int)matgrade[i, j, k1];
                e1[p, k1] += ceb[gp]*2.0f * c20 * hy1[i];
                gp = (int)matgrade[i, j, k2];
                e1[p, k2] -= ceb[gp]*2.0f * c20 * hy1[i];
            }
        }
    }
    if (i % 2 == 1)
    {
        for (j = inicioy - 2; j < finaly + 3; j++)
        {
            if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) ||
            (j % 2 == 0 && i % 2 == 1 && i <= Nx - 2))
            {
                if (j == (inicioy - 2) || j == (finaly + 2))
                {
                    p = Ax.p2d(i, j, Nx, Ny, PH4);
                    gp = (int)matgrade[i, j, k1];
                    e1[p, k1] += ceb[gp] * c20 * hy1[i];
                    gp = (int)matgrade[i, j, k2];
                    e1[p, k2] -= ceb[gp] * c20 * hy1[i];
                }
                else
                {
                    p = Ax.p2d(i, j, Nx, Ny, PH4);
                    gp = (int)matgrade[i, j, k1];
                    e1[p, k1] += ceb[gp] * 2.0f * c20 * hy1[i];
                    gp = (int)matgrade[i, j, k2];
                    e1[p, k2] -= ceb[gp] * 2.0f * c20 * hy1[i];
                }

            }
        }
    }
}
// campo espalhado e2 e e3 no centro
for(i = iniciox; i < finalx; i++)
```

```
            for (j = inicioy - 2; j < finaly + 2; j++)
            {
               if ((j % 2 == 1 && i % 2 == 1 && i <= Nx - 2) || // campo e2
                  (j % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
               {
                  p = Ax.p2a(i, j, Nx, Ny, PH3);
                  gp = (int)matgrade[i, j, k1];
                  e2[p, k1] += ceb[gp] * c20 * hy1[i];
                  gp = (int)matgrade[i, j, k2];
                  e2[p, k2] -= ceb[gp] * c20 * hy1[i];
               }
               if ((j % 2 == 1 && i % 2 == 0 && i <= Nx - 1) || // campo e3
                  (j % 2 == 0 && i % 2 == 1 && i <= Nx - 2))
               {
                  p = Ax.p2d(i, j, Nx, Ny, PH1);
                  gp = (int)matgrade[i, j, k1];
                  e3[p, k1] -= ceb[gp] * c20 * hy1[i];
                  gp = (int)matgrade[i, j, k2];
                  e3[p, k2] += ceb[gp] * c20 * hy1[i];
               }
            }

      }

      // INICIALIZA GRADE AUXILIAR 1D
      public static void inicializando_grade_1d(float[] ceze1, float[] cezh1, float[] chyh1, float[]
chye1,int Mx,
         int NLOSS, float max_loss, float cdtds, float imp0)
      {
         for(int m = 0; m < Mx-1; m++)
         {
            float cos30 = (float)(Math.Cos(Math.PI / 6.0));
            float ka = 1.0f / cos30;
            float kb = ka;
         if (m < (Mx-1-NLOSS))
         {
            ceze1[m] = 1.0f;
            cezh1[m] = kb*cdtds*imp0;
            chyh1[m] = 1.0f;
            chye1[m] = ka*cdtds/imp0;
         }
         else
         {
            float depth_layer = m - (Mx - 1 - NLOSS) + 0.5f;
            float loss_factor = (float) (max_loss * Math.Pow(depth_layer / NLOSS, 2) );
            ceze1[m] = (1.0f - loss_factor) / (1.0f + loss_factor);
            cezh1[m] = kb*cdtds*imp0/(1.0f + loss_factor);
            depth_layer += 0.5f;
            loss_factor = (float) ( max_loss * Math.Pow(depth_layer / NLOSS, 2) );
            chyh1[m] = (1.0f - loss_factor)/(1.0f + loss_factor);
            chye1[m] = (ka * cdtds / imp0) / (1.0f + loss_factor);
         }
        }
      }
```

```
        // ATUALIZA GRADE 1D PARA REGIÃO TFSF
        public static void atualiza_tfsf_grade_1d(int tempo,float cdtds, int  ppw, int tipo_fonte, float
fator_at,
            float[] ez1,float[] hy1, float[] ceze1, float[] cezh1, float[] chyh1, float[] chye1,int Mx)
        {
            for(int m = 0; m < Mx-1; m++)
            {
                hy1[m] = chyh1[m] * hy1[m] + chye1[m] * (ez1[m + 1] - ez1[m]);
            }
            for(int m = 1; m < Mx-1; m++)
            {
                ez1[m] = ceze1[m] * ez1[m] + cezh1[m] * (hy1[m] - hy1[m - 1]);
            }
            // aplica fonte senoidal ou pulso para grade auxiliar 1D
            ez1[0] = Ax.fonte_1(tempo, 0.0f, cdtds, ppw, tipo_fonte, fator_at);
        }
    }
}
//////////////////////////////////////////////////////////////////////////////
ARQUIVO 12: Saida.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Globalization;


namespace progCsharpe_Hex3d_matriz2d_float_2c
{
    class Saida
    {
        public static void saida1(string txsaida1, int Nx, int Ny, int PEZ, int pz, float[,] ez, float[,]
ez_pxy)
        {
            // PÓS-PROCESSAMENTO DO CAMPO Ez 3D:para plano xy: campo ez
            for (int p = 1; p < PEZ + 1; p++)
            {
                int i, j;
                i = Ax.ez_i(p, Nx);
                j = Ax.ez_j(p, Nx);
                ez_pxy[i, j] = ez[p, pz];
            }
            // ESCREVENDO NO PRIMEIRO ARQUIVO TEXTO E LENDO DESTE ARQUIVO -
funcionou
            using (StreamWriter saida_1 = new StreamWriter(txsaida1))
            {
                for (int i = 0; i < Nx + 1; i++)
                {

                    for (int j = 0; j < Ny + 1; j++)
                    {
```

```
                    saida_1.WriteLine(ez_pxy[i,j].ToString("E18",
CultureInfo.CreateSpecificCulture("en-US")));
                }

            }
        } // FINAL DA ESCRITA NO ARQUIVO TEXTO
    } // FINAL METODO SAIDA1

    public static void saida2(string txsaida2, int Nx, int Ny, int Nz, int PEZ, int py, float[,] ez,
float[,] ez_pxz)
    {
        // para plano xz: campo ez

        for (int i = 1; i < Nx + 1; i++)
            for (int k = 1; k < Nz + 1; k++)
                if ((py % 2 == 1 && i % 2 == 1 && i <= Nx) ||
                    (py % 2 == 0 && i % 2 == 0 && i <= Nx - 1))
                {
                    int p = Ax.p2b(i, py, Nx, Ny, PEZ);
                    ez_pxz[i, k] = ez[p, k];
                }
        // ESCREVENDO NO SEGUNDO ARQUIVO TEXTO E LENDO DESTE ARQUIVO -
funcionou
        using (StreamWriter saida_2 = new StreamWriter(txsaida2))
        {
            for (int i = 0; i < Nx + 1; i++)
            {

                for (int j = 0; j < Nz + 1; j++)
                {

                    saida_2.WriteLine(ez_pxz[i,j].ToString("E18",
CultureInfo.CreateSpecificCulture("en-US")));
                }

            }
        } // FINAL DA ESCRITA NO ARQUIVO TEXTO
    } // FINAL METODO SAIDA2
  }
}
//////////////////////////////////////////////////////////
```

ARQUIVO 13: SaidaR.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Globalization;


namespace progCsharpe_Hex3d_matriz2d_float_2c
{
```

```
   class SaidaR
   {
      public static void saidar(string txsaidar, int tempomax, float[] ezr)
      {
        // ESCREVENDO
         using (StreamWriter saida_r = new StreamWriter(txsaidar))
         {
            for (int j = 0; j < tempomax; j++)
               {

                  saida_r.WriteLine(ezr[j].ToString("E18",
CultureInfo.CreateSpecificCulture("en-US")));
               }


         } // FINAL DA ESCRITA NO ARQUIVO TEXTO
      } // FINAL METODO saidar
   }
}
////////////////////////////////////////////////////////////////////////
```

ARQUIVO 14: Matrix.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace progCsharpe_Hex3d_matriz2d_double_1a
{
   class Matrix
      // ESTA CLASSE FOI OBTIDA DE SITE DE AJUDA A PROGRAMADORES
      // PERMITE USAR MATRIZES MUITO GRANDES NO WINDOWS
      // NO ARQUIVO PROPRIEDADES DO PROJETO C# USAR 64 BITS
      // CLASSE PARA MATRIZES 2D USANDO LISTAS DE LISTAS DOUBLE
      // TAMBÉM PODE SER MODIFICADA PARA USAR FLOAT
      // M2 (linhas) ou M (colunas) pode ser maior que 16384

   {
      private List<List<double>> matrix;
      public double this[int p, int k]
      {
         get { return matrix[p][k]; }
         set { matrix[p][k] = value; }
      }

      public Matrix(int M2 = 2, int M = 2)
      {
         List<List<double>> pontos = new List<List<double>>(M2);
         for (int i = 0; i < M2; i++)
         {
            List<double> linha = new List<double>(M);// INICIALIZÇÃO CORRETA
            for (int j = 0; j < M; j++)
            {
```

```
                linha.Add(0.0);// INICIALIZA COM VALORES 0.0
            }
            pontos.Add(linha);// INICIALIZA
        }
        matrix = pontos;
    }
  }// FINAL CLASSE
}// FINAL NAMESPACE

// FUNCIONA COMO UMA MATRIZ NORMAL QUE PODE SER USADA DO
// LADO ESQUERDO OU DIREITO DO SINAL DE ATRIBUIÇÃO (=)
/*
// EXEMPLO
int M = 100;
int M2 = 200;
Console.Write("INICIALIZANDO LISTAS DE LISTAS");
Matrix ez = new Matrix(M2, M); // M2 = M pode ser > 16384
Matrix h1 = new Matrix(M2, M);

int Na = 2;
double soma3 = 0.0;
for (int i = 0; i < M2; i++)
  {

   for (int j = 0; j < M; j++)
    {

      ez[i,j] = 1.0*Na *(i * M + j + 1);// FUNCIONOU
      h1[i, j] = 1.0*Na * (i * M + j + 1);// FUNCIONOU
      double tempx3 = ez[i, j] + h1[i, j]; // FUNCIONOU
      soma3 += tempx3; // TESTANDO SOMA
      //Console.WriteLine("Ez[{0},{1}] = {2}",i,j,tempx3);
    }

  }
*/
// FINAL DO APÊNDICE 2
////////////////////////////////////////////////////////////////////////
```

APÊNDICE 3. PROGRAMA C# PARA O MÉTODO FDTD YEE (COM PMLs E REGIÕES DE CAMPOS TOTAL E ESPALHADO)

ESTE APÊNDICE CONTÉM 11 SAÍDAS
ARQUIVO 1: entrada4.txt

```
121                 Nx ( estes comentários deve ser apagados para usar este arquivo)
121                 Ny
121                 Nz
10                  ppw (número de pontos por comprimento de onda)
534                 tempomax
0                   tipo_fonte (0 = pulso, 1 = senoide)
0                   HARD_SOFT ( 0 = hard, 1 = soft)
1.0                 ds = dx = dy = dz
1.0                 rdsdz = R = ds/dz (usar sempre 1.0)
3.0                 fator_cdtds
0.5                 fator_at
1                   delta_passo
134                 tempomax2
135                 tempomax3
1                   CPML ( 0 = desativa, 1 = ativa )
20                  nxx (define espessura das PMLs; nxx*dx = nxx*dy = nxx*dz)
1.0e-6              Reflexao
1.0e-6              Reflexao2
3.9                 me
3.9                 me2
2                   numero_materiais
0                   numero_objetos
2                   xpml ( 0 = sem saída medidas de reflexão nas PMLs)
3                   ypml (junto com xpml define posições de medida de reflexão)
1                   TFSF ( 0 = desativa, 1 = ativa )
4           deltax ( (nxx + deltax) define posição da região TFSF nas direções x, y, z )
1.0                 f_disp (fator de compensação simples de dispersão numérica)
////////////////////////////////////
```

ARQUIVO 2: materiais.txt (ver Apêndice 2)

```
0.0
1.0
0.0
1.0
6.0e14
1.0
0.0
1.0
////////////////////////////////////
```

ARQUIVO 3: objetos.txt (ver Apêndice 2)

```
1
1
40
45
0
20
21
22
```

```
////////////////////////////////////

ARQUIVO 4: Program.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Globalization;

namespace progCsharpe_Yee3d_matriz2d_float_2b
{
   class Program
   {
      static void Main(string[] args)
      {
         // INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS

         DateTime temp_inicio = DateTime.Now;

         int Nx = 120; // numero de pontos na direção x
         int Ny = 130;  // numero de pontos na direção y
         int Nz = 140; // numero de pontos na direção z
         int ppw = 15; // numero de pontos por comprimento de onda
         int tempomax = 3;
         int tipo_fonte = 1;  // 0=pulso ricker ou 1=senoide
         int HARD_SOFT = 1;  // fonte: 0=hard ou 1=soft
         float ds = 1.0f; // dimensão dos lados do hexagono maior 2D
         float rdsdz = 1.0f;  // razão ds/dz
         float fator_cdtds = 2.0f; // em comprimentos de onda para dipolo_1
         float fator_at = 0.5f;
         int delta_passo = 1;
         int tempomax2 = 4;
         int tempomax3 = 5;
         int CPML = 1;
         int nxx = 20; // numero par
         double Reflexao = 1.0e-6;
         double Reflexao2 = 1.0e-6;
         double me = 2.5;
         double me2 = 2.5;
         int numero_materiais = 1;
         int numero_objetos = 1;
         int ajuste_smax = 0;
         int dinicial = 0;
         int xpml = 5;
         int ypml = 6;
         int TFSF = 1;
         int deltax = 10; // número par: dimensão para região TFSF
         float f_disp = 1.0f;
         // FINAL DAS VARIÁVEIS MAIS IMPORTANTES

         ////////////////////////////////////////////////
         // ENTRADA DE VALORES PARA O ARQUIVO DE ENTRADA3
```

```
using (StreamReader entrada = new StreamReader(@"entrada4.txt"))
{
   // INICIO DA LEITURA DO ARQUIVO
   //CultureInfo.CreateSpecificCulture("en-US");
   Nx = int.Parse(entrada.ReadLine());
   Ny = int.Parse(entrada.ReadLine());
   Nz = int.Parse(entrada.ReadLine());
   ppw = int.Parse(entrada.ReadLine());
   tempomax = int.Parse(entrada.ReadLine());
   tipo_fonte = int.Parse(entrada.ReadLine());
   HARD_SOFT = int.Parse(entrada.ReadLine());
   ds = float.Parse(entrada.ReadLine());
   rdsdz = float.Parse(entrada.ReadLine());
   fator_cdtds = float.Parse(entrada.ReadLine());
   fator_at = float.Parse(entrada.ReadLine());
   delta_passo = int.Parse(entrada.ReadLine());
   tempomax2 = int.Parse(entrada.ReadLine());
   tempomax3 = int.Parse(entrada.ReadLine());
   CPML = int.Parse(entrada.ReadLine());
   nxx = int.Parse(entrada.ReadLine());
   Reflexao = double.Parse(entrada.ReadLine());
   Reflexao2 = double.Parse(entrada.ReadLine());
   me = double.Parse(entrada.ReadLine());
   me2 = double.Parse(entrada.ReadLine());
   numero_materiais = int.Parse(entrada.ReadLine());
   numero_objetos = int.Parse(entrada.ReadLine());
   xpml = int.Parse(entrada.ReadLine());
   ypml = int.Parse(entrada.ReadLine());
   TFSF = int.Parse(entrada.ReadLine());
   deltax = int.Parse(entrada.ReadLine());
   f_disp = float.Parse(entrada.ReadLine());
}// FINAL DA LEITURA DO ARQUIVO: entrada3.txt

// ENTRADA DE VALORES PARA OS ARQUIVOS DE MATERIAIS E OBJETOS
int m_coluna = 4;
int o_coluna = 8;
float[,] materiais = new float[numero_materiais, m_coluna];
float[,] objetos = new float[numero_objetos, o_coluna];
// ENTRADA DE VALORES PARA O ARQUIVO DE MATERIAIS
using (StreamReader material = new StreamReader(@"materiais.txt"))
{
   for (int i = 0; i < numero_materiais; i++)
      for (int j = 0; j < m_coluna; j++)
      {
         materiais[i, j] = float.Parse(material.ReadLine());
      }
}
// ENTRADA DE VALORES PARA O ARQUIVO DE OBJETOS
using (StreamReader objeto = new StreamReader(@"objetos.txt"))
{
   for (int i = 0; i < numero_objetos; i++)
      for (int j = 0; j < o_coluna; j++)
      {
         objetos[i, j] = float.Parse(objeto.ReadLine());
      }
```

```
        } // FINAL DA LEITURA DOS ARQUIVOS: materiais.txt e objetos.txt

        ////////////////////////////////////////////////////

        // VALIDANDO VALORES
        //if (Nx % 2 == 1) Nx = Nx + 1; // Nx, Ny e Nz são sempre pares para funcionar corretamente
        //if (Ny % 2 == 1) Ny = Ny + 1;
        //if (Nz % 2 == 1) Nz = Nz + 1;
        //if (nxx % 2 == 1) nxx = nxx + 1; // nxx é sempre par (largura da camada CPML)
        if (dinicial <= 0) dinicial = 0; // ajuste fino do cinicial nos campos H1, H2 e Hz (lado direito)
        if (dinicial >= 1) dinicial = 1;
        float cdtds;
        cdtds = (float)(1.0 / Math.Sqrt(fator_cdtds)); // número de Courant
        ////////////////////////////////////////////////////

        // MOSTRANDO ALGUNS VALORES NA TELA DO COMPUTADOR
        //CultureInfo.CreateSpecificCulture("en-US");

        Console.WriteLine("INICIO DO PROGRAMA FDTD ( VERSAO YEE 3D MATRIZ 2D: float )\n");
        Console.WriteLine();
        Console.WriteLine("TEMPO INICIAL = {0}", temp_inicio);
        Console.WriteLine();
        Console.WriteLine("Nx = {0}", Nx);
        Console.WriteLine("Ny = {0}", Ny);
        Console.WriteLine("Nz = {0}", Nz);
        Console.WriteLine();
        Console.WriteLine("Pontos por comprimento de onda (ppw)   = {0}", ppw);
        Console.WriteLine("Tempo (numero de passos)               = {0}", tempomax);
        Console.WriteLine();
        Console.WriteLine("Tipo de Sinal ( Pulso = 0 ou Senoide = 1 )  = {0}", tipo_fonte);
        Console.WriteLine("Tipo de Fonte ( Hard  = 0 ou Soft    = 1 )  = {0}", HARD_SOFT);
        Console.WriteLine("Numero de Courant: 1/raiz( {0} )     = {1}", fator_cdtds, cdtds);
        Console.WriteLine();
        Console.WriteLine("Dimensôes    dx = dy = dz = {0}", ds);
        Console.WriteLine("Camada CPML (Ativada = 1) = {0}", CPML);
        Console.WriteLine("Largura camada CPML       = {0}", nxx);
        Console.WriteLine("Reflexão camada CPML      = {0}", Reflexao);
        Console.WriteLine("Coeficiente (me) CPML     = {0}", me);
        Console.WriteLine("Reflexão2 camada CPML_z   = {0}", Reflexao2);
        Console.WriteLine("Coeficiente (me2) CPML_z  = {0}", me2);
        Console.WriteLine();
        Console.WriteLine("Fator atenuacao (senoide) = {0}", fator_at);
        Console.WriteLine("Número de materiais       = {0}", numero_materiais);
        Console.WriteLine("Número de objetos         = {0}", numero_objetos);
        Console.WriteLine("Região TFSF (Ativada = 1) = {0}", TFSF);
        Console.WriteLine();
        ////////////////////////////////////////////////////////////////////

        // OUTRAS VARIÁVEIS INTERNAS DO PROGRAMA
        float dx, dy, dz, Lx, Ly, Lz;
        dx = ds ;// tamanho da célula na direção x
        dy = ds ;  // tamanho da célula na direção y
```

```
dz = ds ;
Lx = (Nx) * dx; // comprimento na direção x
Ly = (Ny) * dy; // comprimento na direção y
Lz = (Nz) * dz; // comprimento na direção z
float HARD_SOFT2 = (float)HARD_SOFT;

//////////////////////////////////////////////////////
// calculando dimensões das matrizes para os campos E e H
int PEZ = Nx * Ny;
int PHX = Nx * (Ny - 1);
int PHY = (Nx - 1) * Ny ;
int PHZ = (Nx - 1) * (Ny - 1);

// DIMENSÕES MÁXIMAS PARA VARIAVEIS DA CAMADA CPML
int nyy = nxx;
int cinicial = 10; // usado na CPML (modo TMz)
int cfinal = cinicial + 2 * nxx; // modificado
int cinicial2 = cinicial; // usado na CPML (modo TEz)
int cfinal2 = cinicial2 + 2 * nyy;
int cinicial3 = cinicial; // usado na CPML (direção z)
int cfinal3 = cinicial3 + 2 * nxx;
int i1 = nxx; // inicio e final da camada CPML
int i2 = Nx - nxx - 1;
int j1 = nyy;
int j2 = Ny - nyy - 1;
int k1 = nxx;
int k2 = Nz - nxx - 1;

// VARIAVEIS PARA REGIÃO TFSF
// deltax definido no arquivo de entrada
int NLOSS = 20; // para grade auxiliar 1D
int M = Nx + 1;
int Mx = M + NLOSS;
float max_loss = 0.35f; // fator de perda
// calculos para inicializar dimesões da região TFSF
int deltay = deltax;  // numero par
int iniciox = nxx + deltax; // numero par
int inicioy = nyy + deltay; // numero par
int inicioz = nxx + deltax; // numero par

int finalx = Nx - iniciox;
int finaly = Ny - inicioy;
int finalz = Nz - inicioz;


// VALORES MAXIMOS PARA REDUZIR USO DE MEMORIA
int PEZ2, PHX2, PHY2, PHZ2;
PEZ2 = PEZ - Ay.fsomaez(j2 + 1, i1, i2, j1, j2);
PHX2 = PHX - Ay.fsomahx(j2 + 1, i1, i2, j1, j2);
PHY2 = PHY - Ay.fsomahy(j2 + 1, i1, i2, j1, j2);
PHZ2 = PHZ - Ay.fsomahz(j2 + 1, i1, i2, j1, j2);

// posição da fonte (pulso ricker ou senóide) Talvez colacar no arquivo de entrada3.txt
int px = Nx / 2 + 1;
int py = Ny / 2 + 1;
```

```
        int pz = Nz / 2 + 1;
        int pcentro = Ay.p2a(px, py, Nx);

        // posições para os 4 pontos para medida de reflexão das PMLs
        int pax = px + xpml;
        int pby = py + ypml;
        int pa = Ay.p2a(pax, py, Nx);
        int pb = Ay.p2a(px, pby, Nx);
        int pc = Ay.p2a(pax, pby, Nx);
        int pd = pz + xpml;
        ////////////////////////////////////////////////////

        Console.WriteLine("INICIALIZACAO - ALOCACAO DINAMICA DE MEMORIA:
MATRIZES 2D float)\n");
        // ALOCAÇÃO DINAMICA DE MEMÓRIA
        // ARRAY 2D
        // CAMPOS ELETRICOS E MAGNETICOS
        float[,] ez = new float[PEZ, Nz - 1];
        float[,] hx = new float[PHX, Nz - 1];
        float[,] hy = new float[PHY, Nz - 1];
        float[,] hz = new float[PHZ, Nz];
        float[,] ex = new float[PHY, Nz];
        float[,] ey = new float[PHX, Nz];

        float[,] ez_pxy = new float[Nx, Ny];
        float[,] ez_pxz = new float[Nx, Nz - 1];

        // COEFIENTES PARA EQUAÇÕES DO METODO FDTD
        float[] cha = new float[numero_materiais];
        float[] chb = new float[numero_materiais];
        float[] cea = new float[numero_materiais];
        float[] ceb = new float[numero_materiais];

        // NOVAS VARIAVEIS PARA CAMADA CPML
        byte[, ,] matgrade = new byte[Nx, Ny, Nz]; // materiais da grade 3D
        byte[,] matez = new byte[Nx, Ny]; // materiais para camadas CPML
        byte[,] mathx = new byte[Nx, Ny - 1];
        byte[,] mathy = new byte[Nx - 1, Ny];
        byte[,] mathz = new byte[Nx - 1, Ny - 1];
        byte[] matztm = new byte[Nz-1];
        byte[] matzte = new byte[Nz];
        byte[] matxa = new byte[Nx]; // novas variaveis
        byte[] matxb = new byte[Nx-1];
        byte[] matya = new byte[Ny];
        byte[] matyb = new byte[Ny-1];

        float[,] phxez = new float[PHX2, Nz - 1]; // campo Hx
        float[,] phxey = new float[PHX, 2 * nxx ];
        float[,] phyez = new float[PHY2, Nz - 1]; // campo Hy
        float[,] phyex = new float[PHY, 2 * nxx];
        float[,] pezhx = new float[PEZ2, Nz - 1]; // campo Ez
        float[,] pezhy = new float[PEZ2, Nz - 1];
        float[,] peyhz = new float[PHX2, Nz]; // campo Ey
        float[,] peyhx = new float[PHX, 2 * nxx + 2];
        float[,] pexhz = new float[PHY2, Nz]; // campo Ex
```

```
        float[,] pexhy = new float[PHY, 2 * nxx + 2];
        float[,] phzex = new float[PHZ2, Nz]; // campo Hz
        float[,] phzey = new float[PHZ2, Nz];

        float[] sh = new float[2 * nxx + 1]; // condutividades das camadas CPML (modo TMz)
        float[] am = new float[2 * nxx + 1]; // novo tamanho
        float[] bm = new float[2 * nxx + 1];
        float[] sh2 = new float[2 * nxx + 1]; // condutividades das camadas CPML (modo TEz)
??
        float[] am2 = new float[2 * nxx + 1];
        float[] bm2 = new float[2 * nxx + 1];
        float[] shz = new float[2 * nxx + 2]; // condutividades das camadas CPML (direção z)
        float[] amz = new float[2 * nxx + 2];
        float[] bmz = new float[2 * nxx + 2];

        // ALOCANDO MEMORIA PARA GRADE 1D AUXILIAR (REGIAO TFSF)
        float[] hy1 = new float[Mx];
        float[] chyh1 = new float[Mx];
        float[] chye1 = new float[Mx];
        float[] ez1 = new float[Mx];
        float[] cezh1 = new float[Mx];
        float[] ceze1 = new float[Mx];

         // VARIAVEIS PARA MEDIDA DE REFLEXÃO DAS PMLs
        float[] eza = new float[tempomax];
        float[] ezb = new float[tempomax];
        float[] ezc = new float[tempomax];
        float[] ezd = new float[tempomax];

        ////////////////////////////////////////////////////////////////////////
        // Calculando memória total necessária para campos eletricos e magneticos
        // FALTA COMPLETAR
        long memoria = (PEZ) * (Nz - 1) +  (PHX) * (Nz - 1) +  (PHY) * (Nz - 1) +
              (PEZ) * (Nz) +  (PHX) * (Nz) +  (PHY) * (Nz) +(Nx) * (Ny ) + (Nx) * (Nz - 1);
        memoria = memoria * sizeof(float);
        // Calculando memória para as variaveis inteiras novas
        long memoria2 = (Nx) * (Ny) + (Nx-1) * (Ny) + (Nx) * (Ny-1) + (Nx-1) * (Ny-1);
        long memoria2b = (Nx ) * (Ny ) * (Nz );
        memoria2 = memoria2 * sizeof(byte);
        memoria2b = memoria2b * sizeof(byte);
        // Calculando memória para as variaveis float novas
        long memoria3 = (PHX2) * (Nz - 1) +  (PHY2) * (Nz - 1) + (PEZ2) * (Nz - 1) +
           (PHX2) * (Nz ) +  (PHY2) * (Nz) +  (PHZ2) * (Nz);
        memoria3 = memoria3 * sizeof(float);
        long memoria_total = memoria + memoria2 + memoria2b + memoria3;
        float mem23 = 100.0f * (memoria2 + memoria2b + memoria3) / memoria_total;

        ////////////////////////////////////////////////////////////////////////
        Console.WriteLine("MEMÓRIA ALOCADA PARA CADA float:          {0} BYTES",
sizeof(float));
        Console.WriteLine("MEMÓRIA ALOCADA PARA CADA int:            {0} BYTES",
sizeof(int));
        Console.WriteLine("MEMÓRIA ALOCADA PARA CADA byte:           {0} BYTE",
sizeof(byte));
```

```
        Console.WriteLine("MEMÓRIA ALOCADA PARA ARRAYS 2D CAMPOS: {0} BYTES",
memoria);
        Console.WriteLine("MEMÓRIA ALOCADA PARA ARRAYS BYTE: {0}
BYTES",memoria2 + memoria2b);
        Console.WriteLine("MEMÓRIA ALOCADA PARA ARRAYS 2D CPML: {0} BYTES",
memoria3);
        Console.WriteLine("MEMÓRIA ALOCADA PARA TODOS OS ARRAYS: {0} BYTES",
memoria_total);
        Console.WriteLine("MEMÓRIA ALOCADA EXTRA ( EM % ): {0} %", mem23);
        Console.WriteLine("FINAL INICIALIZACAO: hz[p,k] = {0}\n", hz[PHZ-2, Nz-2]);
        Console.WriteLine("MATRIZES DE SAIDA: FLOAT SCIENTIFIC, MANTISSA 18
DIGITOS ( VERSÃO 3_C YEE 3D )\n");
        Console.WriteLine("USANDO FONTE PONTUAL OU REGIÃO CAMPO TOTAL /
CAMPO ESPALHADO\n");
        Console.WriteLine("COM MATRIZES SIMPLIFICADAS DE CONDUTIVIDADES
PARA CAMADAS CPML\n");
        Console.WriteLine("E ESPESSURAS IGUAIS DAS CAMADAS DE CONDUTIVIDADE
CPML NAS DIREÇÕES X E Y\n");
        Console.WriteLine("COM 3 OBJETOS: PLACA RETANGULAR (1), ESFERA (2) E
CILINDRO (1)\n");
        Console.WriteLine("E COM 4 PONTOS DE MEDIDA DE REFLEXÃO NAS PMLs\n");
        Console.WriteLine("E OBJETOS COM PRECISÃO NO CENTRO: PLACA
RETANGULAR (4) E ESFERA (5)\n");
        Console.WriteLine("E COM xpml = 0: SEM SAIDA MEDIDAS DE REFLEXÃO\n");
        Console.WriteLine("E COM COMPENSAÇÃO DE DISPERSÃO\n");

        //Console.WriteLine("FINAL INICIALIZACAO: materiais = {0},{1},{2},{3}\n", materiais[1,
0], materiais[1, 1],
        //  materiais[1, 2], materiais[1, 3]);
        //Console.WriteLine("FINAL INICIALIZACAO: ch10 = {0}, {1}\n", cha[0], cha[1]);

        ////////////////////////////////////////////////////////////////////////

        // INICIALIZANDO COEFICIENTES EM FUNÇÃO DOS TIPOS DE MATERIAIS
        MATx.materialx(materiais, numero_materiais, cha, chb, cea, ceb, cdtds, ds);
        /*
        Console.WriteLine("FINAL INICIALIZACAO: cha  = {0}, {1}\n", cha[0], cha[1]);
        Console.WriteLine("FINAL INICIALIZACAO: chb  = {0}, {1}\n", chb[0], chb[1]);
        */

        // INICIALIZANDO OBJETOS NA GRADE 3D
        OBJx.objetox(objetos, numero_objetos, matgrade, dx, dy, dz);

        // INICIALIZANDO GRADE AUXILIAR 1D PARA REGIÃO TFSF
        double imp0a = (120.0 * Math.PI);// impedância característica no vácuo
        float imp0b = (float)imp0a;
        TS.inicializando_grade_1d(ceze1, cezh1, chyh1, chye1, Mx, NLOSS, max_loss,
cdtds,imp0b);

         // INICIALIZANDO VETORES DE CONDUTIVIDADE COM A ESPESSURA PML
DESEJADA
        double ds2 = ds; // lembrar que dx = dy = dz = ds
        double dsmax = (nxx - ajuste_smax) * ds2; // modos TMz e TEz
        double imp0 = (120.0 * Math.PI);// impedância característica no vácuo
        double smax = (-1.0 * (me + 1.0) * Math.Log(Reflexao) / (2.0 * imp0 * dsmax));
```

```
        //GERANDO VETOR CONDUTIVIDADE NO PLANO XY
        for (int i = 0; i < (2 * nxx + 1); i++)
        {
            if (i < 2*nxx - 2*ajuste_smax + 1)
            {
                sh[i] = (float)(Math.Pow(i * 0.5 * ds2 / dsmax, me) * smax);
            }
            else
            {
                sh[i] = (float) smax;
            }

            am[i] = (float)Ay.func_fia(sh[i], cdtds, imp0, ds2);
            bm[i] = (float)Ay.func_fib(sh[i], cdtds, imp0, ds2);
            am2[i] = am[i];
            bm2[i] = bm[i];

        }

        // DIREÇÃO Z NO PLANO XZ
        double dzmax = (nxx ) * dz; // na direção z
        double szmax = (-1.0 * (me2 + 1.0) * Math.Log(Reflexao2) / (2.0 * imp0 * dzmax));
        for (int i = 0; i < 2 * nxx + 2; i++)
        {
            shz[i] = (float)(Math.Pow(i * 0.5 * dz / dzmax, me2) * szmax);
            amz[i] = (float)Ay.func_fia(shz[i], cdtds, imp0, ds); // ds ou dz ?? DUVIDA
IMPORTANTE
            bmz[i] = (float)Ay.func_fib(shz[i], cdtds, imp0, ds);
        }

        ////////////////////////////////////////////////////////////////////////
        // TESTANDO VARIAVEIS
        /*
        for (int i = 0; i < numero_materiais; i++)
        {
            Console.WriteLine(" sh[{0}] = {1}\n",i ,cezh2[i]);
        }
        */
        ////////////////////////////////////////////////////////////////////////

         // GERA LINHAS VERTICAIS QUE DIMINUEM NA DIREÇÃO CENTRAL DA GRADE
(CAMADA CPML)
        for (int a = 0; a < nxx + 1; a++)
        {
            // campo Ez
            for (int j = a; j < Ny - a; j++)
            {
                matez[a, j] = (byte)(cfinal - 2 * a);
                matez[Nx - 1 - a, j] = (byte)(cfinal - 2 * a);
            }
            // campos Hx e Ey
            for (int j = a; j < Ny - 1 - a; j++)
            {
                mathx[a, j] = (byte)(cfinal - 2 * a);
```

```
            mathx[Nx - 1 - a, j] = (byte)(cfinal - 2 * a);
        }

    }

    for (int a = 0; a < nxx; a++)
    {
        // campos Hy e Ex
        for (int j = a + 1; j < Ny - 1 - a; j++)
        {
            mathy[a, j] = (byte)(cfinal - 2 * a - 1);
            mathy[Nx - 2 - a, j] = (byte)(cfinal - 2 * a - 1);
        }
        // campo Hz
        for (int j = a; j < Ny - 1 - a; j++)
        {
            mathz[a, j] = (byte)(cfinal - 2 * a - 1);
            mathz[Nx - 2 - a, j] = (byte)(cfinal - 2 * a - 1);
        }
    }

    // GERA LINHAS HORIZONTAIS QUE DIMINUEM NA DIREÇÃO CENTRAL DA
GRADE (CAMADA CPML)
    for (int a = 0; a < nxx + 1; a++)
    {
        // campo Ez
        for (int i = a; i < Nx - a; i++)
        {
            matez[i,a] = (byte)(cfinal - 2 * a);
            matez[i,Ny - 1 - a] = (byte)(cfinal - 2 * a);
        }
        // campos Hy e Ex
        for (int i = a; i < Nx - 1 - a; i++)
        {
            mathy[i,a] = (byte)(cfinal - 2 * a);
            mathy[i,Ny - 1 - a] = (byte)(cfinal - 2 * a);
        }

    }

    for (int a = 0; a < nxx; a++)
    {
        // campos Hx e Ey
        for (int i = a + 1; i < Nx - 1 - a; i++)
        {
            mathx[i,a] = (byte)(cfinal - 2 * a - 1);
            mathx[i,Ny - 2 - a] = (byte)(cfinal - 2 * a - 1);
        }
        // campo Hz
        for (int i = a + 1; i < Nx - 2 - a; i++)
        {
            mathz[i,a] = (byte)(cfinal - 2 * a - 1);
            mathz[i,Ny - 2 - a] = (byte)(cfinal - 2 * a - 1);
        }
    }
```

```
// GERA LINHAS TRANSVERSAIS A DIREÇÃO Z (CAMADA CPML)
for (int a = 0; a < nxx + 1; a++)
{
   matzte[a] = (byte) (cfinal3 - 2 * a);
   matzte[Nz - 1 - a] = (byte)(cfinal3 - 2 * a);
   if( a <= (nxx - 1) )
   {
      matztm[a] = (byte)(cfinal3 - 2 * a - 1);
      matztm[Nz - 2 - a] = (byte)(cfinal3 - 2 * a - 1);

   }
} // FINAL LINHAS PARA CAMADA CPML

// GERA LINHAS TRANSVERSAIS AS DIREÇÕES X E Z (CAMADA CPML)
for (int a = 0; a < nxx + 1; a++)
{
   matxa[a] = (byte)(cfinal - 2 * a);
   matxa[Nx - 1 - a] = (byte)(cfinal - 2 * a);
   matya[a] = (byte)(cfinal - 2 * a);
   matya[Ny - 1 - a] = (byte)(cfinal - 2 * a);
   if (a <= (nxx - 1))
   {
      matxb[a] = (byte)(cfinal - 2 * a - 1);
      matxb[Nx - 2 - a] = (byte)(cfinal - 2 * a - 1);
      matyb[a] = (byte)(cfinal - 2 * a - 1);
      matyb[Ny - 2 - a] = (byte)(cfinal - 2 * a - 1);

   }
} // FINAL LINHAS PARA CAMADA CPML

//////////////////////////////////////////////////////////////////////
// TESTANDO VARIAVEIS
/*
//for (int p = 0; p < PHX2; p++)
for (int j = 1; j < Ny-1; j++)
   for (int i = 1; i < Nx-1; i++)
{
   //int ip = (int)matxa[i] - cinicial;
   int xp = Ay.zp2b(i, j, Nx, i1, i2, j1, j2);
   Console.WriteLine(" {0} = {1}\n", xp,PHZ);
}
*/
//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////
// INICIO LOOP DO TEMPO
for (int tempo = 0; tempo < tempomax; tempo++)
{
   // SEM COMPUTAÇÃO PARALELA
   DateTime temp_inicio2 = DateTime.Now;

   // ATUALIZA CAMPOS MAGNÉTICOS
   TEM.atualiza_hx( cha, chb,ey, ez, hx,phxez, phxey,am, bm,am2, bm2, Nx,Ny,Nz,
i1, i2,
```

```
                    j1, j2, k1, k2, CPML, cinicial, cinicial3, matgrade, mathx, matztm, matyb,
f_disp);
            TEM.atualiza_hy(cha, chb, ex, ez, hy, phyez, phyex, am, bm, am2, bm2, Nx, Ny, Nz,
i1, i2,
                    j1, j2, k1, k2, CPML, cinicial, cinicial3, matgrade, mathy, matztm, matxb,
f_disp);
            TEM.atualiza_hz(cha, chb, ex, ey, hz, phzex, phzey, am, bm, Nx, Ny, Nz, i1, i2,
                    j1, j2, CPML, cinicial, matgrade, mathz, matxb, matyb, f_disp);

            // ATUALIZA TFSF
            if (TFSF == 1)
            {
               // USA REGIÃO CAMPO TOTAL / CAMPO ESPALHADO
               TS.atualiza_tfsf_h(hx, hy, chb, ez1, matgrade, iniciox, finalx, inicioy, finaly, inicioz,
finalz, Nx);
               TS.atualiza_tfsf_grade_1d(tempo, cdtds, ppw, tipo_fonte, fator_at, ez1, hy1,
ceze1, cezh1, chyh1, chye1, Mx);
               TS.atualiza_tfsf_ez(ez, ex, ceb, hy1, matgrade, iniciox, finalx, inicioy, finaly,
inicioz, finalz, Nx);

            }

            // ATUALIZA CAMPOS ELETRICOS
            TEM.atualiza_ex(cea, ceb, hy, hz, ex, pexhy, pexhz, am, bm, am2, bm2, Nx, Ny, Nz,
i1, i2,
                    j1, j2, k1, k2, CPML, cinicial, cinicial3, matgrade, mathy, matzte, matya,
f_disp);
            TEM.atualiza_ey( cea, ceb, hx, hz, ey, peyhx, peyhz, am, bm, am2,  bm2, Nx,  Ny,
Nz,  i1, i2,
                    j1, j2, k1, k2, CPML, cinicial, cinicial3, matgrade, mathx, matzte, matxa,
f_disp);
            TEM.atualiza_ez( cea, ceb, hx, hy, ez, pezhx, pezhy, am, bm,Nx, Ny, Nz, i1, i2,
                    j1, j2, CPML, cinicial, matgrade, matez, matxa, matya, f_disp);


            if (TFSF == 0)
            {
               // APLICA A FONTE PONTUAL NO PONTO ESCOLHIDO
               ez[pcentro, pz] = HARD_SOFT2 * ez[pcentro, pz] + Ay.fonte_1(tempo, 0.0f, cdtds,
ppw, tipo_fonte, fator_at);
            }

            // atualizando os tres pontos para medida de reflexão nas PMLs
            if (xpml > 0)
            {
               eza[tempo] = ez[pa, pz];
               ezb[tempo] = ez[pb, pz];
               ezc[tempo] = ez[pc, pz];
               ezd[tempo] = ez[pcentro, pd];
            }

            if (tempo == tempomax2 - 1)
            {
               int nteste = tempomax2;
               string txx = nteste.ToString();
```

```
                string txsaida1 = @"ez_pxy_yee1_" + txx + ".txt";
                Saida.saida1(txsaida1, Nx, Ny, pz, ez, ez_pxy);
                string txsaida2 = @"ez_pxz_yee2_" + txx + ".txt";
                Saida.saida2(txsaida2, Nx, Ny, Nz, py, ez, ez_pxz);
            }
            if (tempo == tempomax3 - 1)
            {
                int nteste = tempomax3;
                string txx = nteste.ToString();
                string txsaida1 = @"ez_pxy_yee1_" + txx + ".txt";
                Saida.saida1(txsaida1, Nx, Ny, pz, ez, ez_pxy);
                string txsaida2 = @"ez_pxz_yee2_" + txx + ".txt";
                Saida.saida2( txsaida2, Nx, Ny, Nz, py, ez, ez_pxz);
            }

            DateTime temp_final2 = DateTime.Now;
            TimeSpan tempo_total2 = temp_final2 - temp_inicio2;
            if ((tempo + 1) % delta_passo == 0)
            {
                Console.WriteLine(" {0,8} / {1} em {2} ", tempo + 1, tempomax, tempo_total2);
            }
        } // FINAL LOOP DO TEMPO
        ////////////////////////////////////////////////////////////
        ////////////////////////////////////////////////////////////////////
        // TESTANDO VARIAVEIS
        /*
        for (int p = 0; p < PHX2; p++)
            for (int i = 0; i < 1; i++)
            {
                int ip = (int)matxa[i] - cinicial;
                Console.WriteLine(" {0} = {1}\n", p, phxez[p, pz]);
            }
        */
        ////////////////////////////////////////////////////////////////////

        //METODOS PARA ESCRITA NOS DOIS ARQUIVOS DE SAÍDA
        int nteste2 = tempomax;
        string txx2 = nteste2.ToString();
        string txsaida1a = @"ez_pxy_yee1_" + txx2 + ".txt";
        Saida.saida1(txsaida1a, Nx, Ny, pz, ez, ez_pxy);
        string txsaida2a = @"ez_pxz_yee2_" + txx2 + ".txt";
        Saida.saida2(txsaida2a, Nx, Ny, Nz, py, ez, ez_pxz);

        // METODOS PARA ESCRITA DAS 4 VARIAVEIS PARA MEDIDA DE REFLEXÃO
NAS PMLs
        if (xpml > 0)
        {
            string tnxx = nxx.ToString();
            string tcpml = CPML.ToString();
            string tx_eza = @"ez_YA_nxx_" + tnxx + "_CPML_" + tcpml + "_Temp_" + txx2 +
".txt";
            SaidaR.saidar(tx_eza, tempomax, eza);
            string tx_ezb = @"ez_YB_nxx_" + tnxx + "_CPML_" + tcpml + "_Temp_" + txx2 +
".txt";
            SaidaR.saidar(tx_ezb, tempomax, ezb);
```

```
            string tx_ezc = @"ez_YC_nxx_" + tnxx + "_CPML_" + tcpml + "_Temp_" + txx2 +
".txt";
            SaidaR.saidar(tx_ezc, tempomax, ezc);
            string tx_ezd = @"ez_YD_nxx_" + tnxx + "_CPML_" + tcpml + "_Temp_" + txx2 +
".txt";
            SaidaR.saidar(tx_ezd, tempomax, ezd);
        }

        // MEDINDO TEMPO DE EXECUÇÃO DO PROGRAMA
        DateTime temp_final = DateTime.Now;
        TimeSpan tempo_total = temp_final - temp_inicio;
        Console.WriteLine("\nTEMPO INICIAL: {0}", temp_inicio);
        Console.WriteLine("\nTEMPO FINAL:   {0}", temp_final);
        Console.WriteLine("\nTEMPO TOTAL (HH : MM : SS): {0}", tempo_total);
        Console.Write("\nENTRE COM UM NÚMERO OU LETRA PARA SAIR DO
PROGRAMA: ");
        string FINAL = Console.ReadLine();
        // FINAL DO PROGRAMA
    }
  }
}
////////////////////////////////////////////////////////////////////////////
```

ARQUIVO 5: Ay.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace progCsharpe_Yee3d_matriz2d_float_2b
{
  class Ay
  {
    // Função para ez, hx, ey
    public static int p2a(int i, int j, int Nx)
    {
      int p2;
      p2 = i + j * Nx;
      return p2;
    }
    // Função para hy, ex, hz
    public static int p2b(int i, int j, int Nx)
    {
      int p2;
      p2 = i + j * (Nx - 1);
      return p2;
    }

    // NOVAS FUNÇÕES PARA CAMADA CPML
    // FUNÇÕES PARA CAMPO Ez
    public static int fsomaez(int j, int i1, int i2, int j1, int j2)
    {
      if (j >= (j2 - 1)) j = j2 - 1;
```

```
    int ix = i2 - i1 - 1;
    int jx = j - j1;
    int soma = jx * ix;
    return soma;
}

public static int zp2a(int i, int j, int Nx, int i1, int i2, int j1, int j2)
{
    int p = 0;
    if (j <= j1) p = p2a(i, j, Nx);
    if (j > j1 && i <= i1) p = p2a(i, j, Nx) - fsomaez(j - 1, i1, i2, j1, j2);
    if ((j > j1 && i >= i2) || j >= j2) p = p2a(i, j, Nx) - fsomaez(j, i1, i2, j1, j2);
    return p;
}

// FUNÇÕES PARA CAMPO Hx e Ey
public static int fsomahx(int j, int i1, int i2, int j1, int j2)
{
    if (j >= (j2 - 1)) j = j2 - 1;
    int ix = i2 - i1 - 1;
    int jx = j - j1 + 1;
    int soma = jx * ix;
    return soma;
}

public static int xp2a(int i, int j, int Nx, int i1, int i2, int j1, int j2)
{
    int p = 0;
    if (j < j1) p = p2a(i, j, Nx);
    if (j >= j1 && i <= i1) p = p2a(i, j, Nx) - fsomahx(j - 1, i1, i2, j1, j2);
    if ((j >= j1 && i >= i2) || j >= j2) p = p2a(i, j, Nx) - fsomahx(j, i1, i2, j1, j2);
    return p;
}

// FUNÇÕES PARA CAMPO Hz
public static int fsomahz(int j, int i1, int i2, int j1, int j2)
{
    if (j >= (j2 - 1)) j = j2 - 1;
    int ix = i2 - i1 ;
    int jx = j - j1 + 1;
    int soma = jx * ix;
    return soma;
}

public static int zp2b(int i, int j, int Nx, int i1, int i2, int j1, int j2)
{
    int p = 0;
    if (j < j1) p = p2b(i, j, Nx);
    if (j >= j1 && i < i1) p = p2b(i, j, Nx) - fsomahz(j - 1, i1, i2, j1, j2);
    if ((j >= j1 && i >= i2) || j >= j2) p = p2b(i, j, Nx) - fsomahz(j, i1, i2, j1, j2);
    return p;
}

// FUNÇÕES PARA CAMPOS Hy e Ex
public static int fsomahy(int j, int i1, int i2, int j1, int j2)
```

```
    {
        if (j >= (j2 - 1)) j = j2 - 1;
        int ix = i2 - i1;
        int jx = j - j1 ;
        int soma = jx * ix;
        return soma;
    }

    public static int yp2b(int i, int j, int Nx, int i1, int i2, int j1, int j2)
    {
        int p = 0;
        if (j <= j1) p = p2b(i, j, Nx);
        if (j > j1 && i < i1) p = p2b(i, j, Nx) - fsomahy(j - 1, i1, i2, j1, j2);
        if ((j > j1 && i >= i2) || j >= j2) p = p2b(i, j, Nx) - fsomahy(j, i1, i2, j1, j2);
        return p;
    }

    // FUNÇÕES PARA CAMADAS CPML NA DIREÇÃO Z
    //PARA MODO TMz
    public static int pztm(int k, int Nz, int k1, int k2)
    {
        int p = 0;
        if ( k <= (k1 - 1) ) p = k;
        if (k >= k2) p = k1 + k - k2;
        if ( k > (Nz - 2) ) p = k1 + Nz - 2 - k2 ;
        return p;
    }
    //PARA MODO TEz
    public static int pzte(int k, int Nz, int k1, int k2)
    {
        int p = 0;
        if (k <= k1) p = k;
        if (k >= k2) p = k1 + k - k2 + 1;
        if ( k > (Nz - 1) ) p = k1 + Nz - k2;
        return p;
    }

    // FUNÇÃO PARA FONTE DO TIPO PULSO RICKER OU SENOIDE
    public static float fonte_1(int tempo1, float posicao, float cdtds1, int ppw, int tipo_fonte,
float fator_at)
    {
        double arg, arg2, funcao, periodo, cte_tempo, atenuacao;
        double pi = Math.PI;
        double tempo2 = (double)tempo1;
        double ppw2 = (double)ppw;

        if (tipo_fonte == 0)
        { // pulso ricker
            arg = pi * ((cdtds1 * tempo2 - posicao) / ppw2 - 1.0);
            arg2 = arg * arg;
            funcao = (1.0 - 2.0 * arg2) * Math.Exp(-arg2);
        }

        else
        {  // senóide com amortecimento inicial
```

```
                periodo = ppw2 / cdtds1; // periodo da senóide
                // ou tempo para o pico do pulso ricker
                cte_tempo = fator_at * periodo;
                atenuacao = 1.0 - Math.Exp(-1.0 * tempo2 / cte_tempo);
                funcao = atenuacao * Math.Sin(2.0 * pi * tempo2 * cdtds1 / ppw2);
            }
            return (float)funcao;
        }

        // FUNÇÕES PARA CALCULAR CONDUTIVIDADE NA CAMADA CPML
        public static double func_fia(double sigma, double cdtds, double imp0, double ds)
        {
            double a = (Math.Exp(-sigma * cdtds * imp0 * ds) - 1.0);
            return a;
        }
        public static double func_fib(double sigma, double cdtds, double imp0, double ds)
        {
            double b = (Math.Exp(-sigma * cdtds * imp0 * ds));
            return b;
        }
        //FINAL DOS MÉTODOS
    }
}
//////////////////////////////////////////////////////////////////
```

ARQUIVO 6: MATx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace progCsharpe_Yee3d_matriz2d_float_2b
{
    class MATx
    {
        public static void materialx(float[,] materiais, int n_materiais,  float[] cha, float[] chb,
            float[] cea, float[] ceb, float cdtds, float ds)
        {
            for (int i = 0; i < n_materiais; i++)
            {
                double s = (double)materiais[i, 0];
                double er = (double)materiais[i, 1];
                double sm = (double)materiais[i, 2];
                double ur = (double)materiais[i, 3];
                double eo = 1.0e-9 / (36.0 * Math.PI);
                double uo = 4.0e-7 * Math.PI;
                double e = er * eo;
                double u = ur * uo;
                double co = 1.0 / Math.Sqrt(uo * eo);
                double dt = cdtds * ds / co;
                cha[i] = (float)((2.0 * u - sm * dt) / (2.0 * u + sm * dt));
                cea[i] = (float)((2.0 * e - s * dt) / (2.0 * e + s * dt));
                chb[i] = (float)((2.0 * dt) / (ds * (2.0 * u + sm * dt)));
```

```
            ceb[i] = (float)((2.0 * dt) / (ds * (2.0 * e + s * dt)));
        }
      }
   }
}
//////////////////////////////////////////////////////////////////
```

ARQUIVO 7: OBJx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace progCsharpe_Yee3d_matriz2d_float_2b
{
   class OBJx
   {
      public static void objetox(float[,] objetos, int n_objetos, byte[,,] matgrade, float dx, float dy,
float dz)
      {
         for(int i = 0; i < n_objetos; i++)
         {
            int tipo_objeto = (int) objetos[i, 0];
            if(tipo_objeto == 1)
            {
               int material1 = (int) objetos[i, 1];
               int x = (int) (objetos[i, 2]/dx);
               int y = (int)(objetos[i, 3] / dy);
               int z = (int)(objetos[i, 4] / dz);
               int lx = (int)(objetos[i, 5] / dx);
               int ly = (int)(objetos[i, 6] / dy);
               int lz = (int)(objetos[i, 7] / dz);
               for(int m = x; m < x + lx; m++)
                  for(int n = y ; n < y + ly; n++)
                     for(int p = z; p < z + lz; p++)
                     {
                        matgrade[m,n,p] = (byte) material1;
                     }
            }//final objeto: placa retangular
            if (tipo_objeto == 2) // ESFERA
            {
               int material1 = (int)objetos[i, 1];
               int x = (int)(objetos[i, 2] / dx);
               int y = (int)(objetos[i, 3] / dy);
               int z = (int)(objetos[i, 4] / dz);
               int rx = (int)(objetos[i, 5] / dx);
               int ry = (int)(objetos[i, 5] / dy);
               int rz = (int)(objetos[i, 5] / dz);
               float raio = (float)(objetos[i, 5]);
               int iniciox = x - rx;
               int finalx = x + rx;
               int inicioy = y - ry;
               int finaly = y + ry;
```

```
    int inicioz = z - rz;
    int finalz = z + rz;
    if (iniciox <= 2) iniciox = 2; // FALTA LIMITES COM Nx, Ny e Nz
    for (int m = iniciox; m < finalx; m++)
        for (int n = inicioy; n < finaly; n++)
            for (int p = inicioz; p < finalz; p++)
            {
                float di = (m - x) * dx;
                float dj = (n - y) * dy;
                float dk = (p - z) * dz;
                if ((di * di + dj * dj + dk * dk) <= (raio * raio))
                {
                    matgrade[m, n, p] = (byte)material1;
                }

            }
}// FINAL OBJETO esfera

if (tipo_objeto == 3) // CILINDRO
{
    int material1 = (int)objetos[i, 1];
    int x = (int)(objetos[i, 2] / dx);
    int y = (int)(objetos[i, 3] / dy);
    int z = (int)(objetos[i, 4] / dz);
    int rx = (int)(objetos[i, 5] / dx);
    int ry = (int)(objetos[i, 5] / dy);
    int lz = (int)(objetos[i, 6] / dz);
    float raio = (float)(objetos[i, 5]);
    int iniciox = x - rx;
    int finalx = x + rx;
    int inicioy = y - ry;
    int finaly = y + ry;
    int inicioz = z;
    int finalz = z + lz;
    if (iniciox <= 2) iniciox = 2;
    for (int m = iniciox; m < finalx; m++)
        for (int n = inicioy; n < finaly; n++)
            for (int p = inicioz; p < finalz; p++)
            {
                float di = (m - x) * dx;
                float dj = (n - y) * dy;

                if ((di * di + dj * dj) <= (raio * raio))
                {
                    matgrade[m, n, p] = (byte)material1;
                }

            }
}// FINAL OBJETO cilindro

if (tipo_objeto == 4) // PLACA RETANGULAR 2 (posição central com precisão)
{
    int material1 = (int)objetos[i, 1];
    int x = (int)(objetos[i, 2]);
    int y = (int)(objetos[i, 3]);
```

```
                int z = (int)(objetos[i, 4]);
                int lx = (int)(objetos[i, 5] / dx);
                int ly = (int)(objetos[i, 6] / dy);
                int lz = (int)(objetos[i, 7] / dz);
                ly = ly / 2;
                lz = lz / 2;

                for (int m = x; m < x + lx; m++)
                    for (int n = y - ly; n < y + ly; n++)
                        for (int p = z - lz; p < z + lz; p++)
                        {
                            matgrade[m, n, p] = (byte)material1;
                        }
            }// FINAL OBJETO placa retangular 2

            if (tipo_objeto == 5) // ESFERA 2 (posição central com precisão)
            {
                int material1 = (int)objetos[i, 1];
                int x = (int)(objetos[i, 2]);
                int y = (int)(objetos[i, 3]);
                int z = (int)(objetos[i, 4]);
                int rx = (int)(objetos[i, 5] / dx);
                int ry = (int)(objetos[i, 5] / dy);
                int rz = (int)(objetos[i, 5] / dz);
                float raio = (float)(objetos[i, 5]);
                int iniciox = x - rx;
                int finalx = x + rx;
                int inicioy = y - ry;
                int finaly = y + ry;
                int inicioz = z - rz;
                int finalz = z + rz;
                if (iniciox <= 2) iniciox = 2; // FALTA LIMITES COM Nx, Ny e Nz
                for (int m = iniciox; m < finalx; m++)
                    for (int n = inicioy; n < finaly; n++)
                        for (int p = inicioz; p < finalz; p++)
                        {
                            float di = (m - x) * dx;
                            float dj = (n - y) * dy;
                            float dk = (p - z) * dz;
                            if ((di * di + dj * dj + dk * dk) <= (raio * raio))
                            {
                                matgrade[m, n, p] = (byte)material1;
                            }

                        }
            }// FINAL OBJETO esfera 2
        }
    }
  }
}
//////////////////////////////////////////////////////////////////
```

ARQUIVO 8: TEM.cs

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace progCsharpe_Yee3d_matriz2d_float_2b
{
   class TEM
   {
      // ATUALIZANDO CAMPO Hx
      public static void atualiza_hx( float[] cha, float[] chb, float[,] ey, float[,] ez,float[,] hx,float[,]
phxez,
         float[,] phxey,float[] am, float[] bm,float[] am2, float[] bm2,int Nx,int Ny,int Nz, int i1, int
i2,
         int j1, int j2, int k1, int k2, int CPML, int cinicial, int cinicial3, byte[,,] matgrade,
         byte[,] mathx, byte[] matztm, byte[] matyb, float f_disp)
      {
         int p, n, gp;
         // campo hx
         for(int i=0; i < Nx; i++)
            for(int j=0; j < Ny-1; j++)
               for(int k=0; k < Nz-1; k++)
               {
                     p = Ay.p2a(i,j,Nx);
                     n = Ay.p2a(i,j+1,Nx);
                     gp = (int) matgrade[i,j,k];

                     hx[p,k] = cha[gp]*hx[p,k] +
                              f_disp*chb[gp] * ((ey[p, k + 1] - ey[p, k]) -
                                       (ez[n,k] - ez[p,k])  );
                  // CAMADAS CPML PARA Hx
                     if (( j < j1 || j >= j2) && CPML == 1)
                  {
                     int ip = (int)matyb[j] -  cinicial;
                     int xp = Ay.xp2a(i,j,Nx,i1,i2,j1,j2);
                     phxez[xp,k] = bm[ip]*phxez[xp,k] + am[ip]*(ez[n,k] - ez[p,k]);
                     hx[p, k] = hx[p, k] - f_disp * chb[gp] * phxez[xp, k];
                  }
                  if((k < k1 || k >= k2) && CPML == 1)
                  {
                     int ip =  (int) matztm[k] - cinicial3;
                     int zp =  Ay.pztm(k, Nz, k1, k2);
                     phxey[p,zp] = bm2[ip]*phxey[p,zp] + am2[ip]*(ey[p,k+1] - ey[p,k]);
                     hx[p, k] = hx[p, k] + f_disp * chb[gp] * phxey[p, zp];
                  }
                              }// final for
        } // final da função

      // ATUALIZANDO CAMPO Hy
      public static void atualiza_hy(float[] cha, float[] chb, float[,] ex, float[,] ez, float[,] hy, float[,]
phyez,
         float[,] phyex, float[] am, float[] bm, float[] am2, float[] bm2, int Nx, int Ny, int Nz, int i1,
int i2,
         int j1, int j2, int k1, int k2, int CPML, int cinicial, int cinicial3, byte[, ,] matgrade,
         byte[,] mathy, byte[] matztm, byte[] matxb, float f_disp)
```

```
{ int p, e, w, gp;
   // campo hy
   for(int i=0; i < Nx-1; i++)
      for(int j=0; j < Ny; j++)
         for(int k=0; k < Nz-1; k++)
         {
               p = Ay.p2b(i,j,Nx);
               e = Ay.p2a(i+1,j,Nx);
               w = Ay.p2a(i, j, Nx);
               gp = (int)matgrade[i, j, k];

               hy[p,k] = cha[gp]*hy[p,k] +
                        f_disp * chb[gp] * ((ez[e, k] - ez[w, k]) -
                              (ex[p,k+1] - ex[p,k])  );
               // CAMADAS CPML PARA Hy
               if ((i < i1 || i >= i2 ) && CPML == 1)
               {
                  int ip = (int)matxb[i] - cinicial;
                  int xp = Ay.yp2b(i, j, Nx, i1, i2, j1, j2);
                  phyez[xp, k] = bm[ip] * phyez[xp, k] + am[ip] * (ez[e, k] - ez[w, k]);
                  hy[p, k] = hy[p, k] + f_disp * chb[gp] * phyez[xp, k];
               }
               if ((k < k1 || k >= k2) && CPML == 1)
               {
                  int ip = (int)matztm[k] - cinicial3;
                  int zp = Ay.pztm(k, Nz, k1, k2);
                  phyex[p, zp] = bm2[ip] * phyex[p, zp] + am2[ip] * (ex[p, k + 1] - ex[p, k]);
                  hy[p, k] = hy[p, k] - f_disp * chb[gp] * phyex[p, zp];
               }
                        }// final for
  } // final da função

 // ATUALIZANDO CAMPO Ez
 public static void atualiza_ez(float[] cea, float[] ceb, float[,] hx, float[,] hy, float[,] ez, float[,]
pezhx,
   float[,] pezhy, float[] am, float[] bm, int Nx, int Ny, int Nz, int i1, int i2,
   int j1, int j2, int CPML, int cinicial, byte[, ,] matgrade, byte[,] matez, byte[] matxa, byte[]
matya, float f_disp)
 {
   int p, s, e, w, gp;
   // campo ez
   for(int i = 1; i < Nx-1; i++)
      for(int j = 1; j < Ny-1; j++)
         for(int k = 0; k < Nz-1; k++)
         {
               p = Ay.p2a(i,j,Nx);
               s = Ay.p2a(i,j - 1,Nx);
                                    e = Ay.p2b(i,j,Nx);
               w = Ay.p2b(i - 1, j, Nx);
               gp = (int)matgrade[i, j, k];

               ez[p,k] = cea[gp]*ez[p,k] +
                        f_disp * ceb[gp] * ((hy[e, k] - hy[w, k]) -
                              (hx[p,k] - hx[s,k])  );
               // CAMADAS CPML PARA Ez
```

```
                    if (( j <= j1 || j >= j2) && CPML == 1)
                    {
                        int ip = (int)matya[j] - cinicial;
                        int xp = Ay.zp2a(i, j, Nx, i1, i2, j1, j2);
                        pezhx[xp, k] = bm[ip] * pezhx[xp, k] + am[ip] * (hx[p, k] - hx[s, k]);
                        ez[p, k] = ez[p, k] - f_disp * ceb[gp] * pezhx[xp, k];
                    }
                    if ((i <= i1 || i >= i2 ) && CPML == 1)
                    {
                        int ip = (int)matxa[i] - cinicial;
                        int xp = Ay.zp2a(i, j, Nx, i1, i2, j1, j2);
                        pezhy[xp, k] = bm[ip] * pezhy[xp, k] + am[ip] * (hy[e, k] - hy[w, k]);
                        ez[p, k] = ez[p, k] + f_disp * ceb[gp] * pezhy[xp, k];
                    }
                            }//final for
        } // final da função

        // ATUALIZANDO CAMPO Ex
      public static void atualiza_ex( float[] cea, float[] ceb, float[,] hy, float[,] hz,float[,] ex,float[,]
pexhy,
         float[,] pexhz,float[] am, float[] bm,float[] am2, float[] bm2,int Nx,int Ny,int Nz, int i1, int
i2,
         int j1, int j2, int k1, int k2, int CPML, int cinicial, int cinicial3, byte[,,] matgrade,
         byte[,] mathy, byte[] matzte, byte[] matya, float f_disp)
       { int p, s, gp;
          // campo ex
          for(int i = 0; i < Nx-1; i++)
             for(int j = 1; j < Ny-1; j++)
                for(int k = 1; k < Nz-1; k++)
                {
                        p = Ay.p2b(i,j,Nx);
                        s = Ay.p2b(i,j - 1,Nx);
                        gp = (int)matgrade[i, j, k];

                        ex[p,k] = cea[gp]*ex[p,k] +
                                f_disp * ceb[gp] * ((hz[p, k] - hz[s, k]) -
                                        (hy[p,k] - hy[p,k-1])  );
                    // CAMADAS CPML PARA Ex
                        if (( j <= j1 || j >= j2) && CPML == 1)
                        {
                            int ip = (int)matya[j] - cinicial;
                            int xp = Ay.yp2b(i, j, Nx, i1, i2, j1, j2);
                            pexhz[xp, k] = bm[ip] * pexhz[xp, k] + am[ip] * (hz[p,k] - hz[s,k]);
                            ex[p, k] = ex[p, k] + f_disp * ceb[gp] * pexhz[xp, k];
                        }
                        if ((k <= k1 || k >= k2) && CPML == 1)
                        {
                            int ip = (int)matzte[k] - cinicial3;
                            int zp = Ay.pzte(k, Nz, k1, k2);
                            pexhy[p, zp] = bm2[ip] * pexhy[p, zp] + am2[ip] * (hy[p,k] - hy[p,k-1]);
                            ex[p, k] = ex[p, k] - f_disp * ceb[gp] * pexhy[p, zp];
                        }
                                } // final for
        } // final da função
```

```
    // ATUALIZANDO CAMPO Ey
    public static void atualiza_ey(float[] cea, float[] ceb, float[,] hx, float[,] hz, float[,] ey, float[,]
peyhx,
      float[,] peyhz, float[] am, float[] bm, float[] am2, float[] bm2, int Nx, int Ny, int Nz, int i1,
int i2,
      int j1, int j2, int k1, int k2, int CPML, int cinicial, int cinicial3, byte[, ,] matgrade,
      byte[,] mathx, byte[] matzte, byte[] matxa, float f_disp)
    { int p, e, w, gp;
      // campo ey
      for(int i = 1; i < Nx-1; i++)
        for(int j = 0; j < Ny-1; j++)
          for(int k = 1; k < Nz-1; k++)
          {
              p = Ay.p2a(i,j,Nx);
              e = Ay.p2b(i,j,Nx);
              w = Ay.p2b(i - 1, j, Nx);
              gp = (int)matgrade[i, j, k];

              ey[p, k] = cea[gp] * ey[p, k] +
                      f_disp * ceb[gp] * ((hx[p, k] - hx[p, k - 1]) -
                              (hz[e,k] - hz[w,k])  );
              // CAMADAS CPML PARA Ey
              if ((i <= i1 || i >= i2 ) && CPML == 1)
              {
                 int ip = (int)matxa[i] - cinicial;
                 int xp = Ay.xp2a(i, j, Nx, i1, i2, j1, j2);
                 peyhz[xp, k] = bm[ip] * peyhz[xp, k] + am[ip] * (hz[e, k] - hz[w, k]);
                 ey[p, k] = ey[p, k] - f_disp * ceb[gp] * peyhz[xp, k];
              }
              if ((k <= k1 || k >= k2) && CPML == 1)
              {
                 int ip = (int)matzte[k] - cinicial3;
                 int zp = Ay.pzte(k, Nz, k1, k2);
                 peyhx[p, zp] = bm2[ip] * peyhx[p, zp] + am2[ip] * (hx[p, k] - hx[p, k - 1]);
                 ey[p, k] = ey[p, k] + f_disp * ceb[gp] * peyhx[p, zp];
              }
                        } // final for
    } // final da função

    // ATUALIZANDO CAMPO Hz
    public static void atualiza_hz(float[] cha, float[] chb, float[,] ex, float[,] ey, float[,] hz, float[,]
phzex,
      float[,] phzey, float[] am, float[] bm, int Nx, int Ny, int Nz, int i1, int i2,
      int j1, int j2, int CPML, int cinicial, byte[, ,] matgrade, byte[,] mathz, byte[] matxb, byte[]
matyb, float f_disp)
    { int p, n, e, w, gp;
      // campo hz
      for(int i=0; i < Nx-1; i++)
        for(int j=0; j < Ny-1; j++)
          for(int k=0; k < Nz; k++)
          {
              p = Ay.p2b(i,j,Nx);
              n = Ay.p2b(i, j + 1, Nx);
              e = Ay.p2a(i + 1, j, Nx);
              w = Ay.p2a(i, j, Nx);
```

```
                    gp = (int)matgrade[i, j, k];

                    hz[p,k] = cha[gp]*hz[p,k] +
                              f_disp * chb[gp] * ((ex[n, k] - ex[p, k]) -
                                      (ey[e,k] - ey[w,k])  );
                    // CAMADAS CPML PARA Hz
                    if (( j < j1 || j >= j2) && CPML == 1)
                    {
                        int ip = (int)matyb[j] - cinicial;
                        int xp = Ay.zp2b(i, j, Nx, i1, i2, j1, j2);
                        phzex[xp, k] = bm[ip] * phzex[xp, k] + am[ip] * (ex[n, k] - ex[p, k]);
                        hz[p, k] = hz[p, k] + f_disp * chb[gp] * phzex[xp, k];
                    }
                    if ((i < i1 || i >= i2 ) && CPML == 1)
                    {
                        int ip = (int)matxb[i] - cinicial;
                        int xp = Ay.zp2b(i, j, Nx, i1, i2, j1, j2);
                        phzey[xp, k] = bm[ip] * phzey[xp, k] + am[ip] * (ey[e, k] - ey[w, k]);
                        hz[p, k] = hz[p, k] - f_disp * chb[gp] * phzey[xp, k];
                    }
                              } // final for
        } // final da função


    }
}
////////////////////////////////////////////////////////////////////////////////
```

ARQUIVO 9: TS.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace progCsharpe_Yee3d_matriz2d_float_2b
{
    class TS
    {
        // ATUALIZA REGIÃO TFSF PARA Hx E Hy
        public static void atualiza_tfsf_h(float[,] hx, float[,] hy,  float[] chb, float[] ez1, byte[,,]
matgrade,
            int iniciox, int finalx, int inicioy,int finaly, int inicioz, int finalz, int Nx)
        {
            int i, j, k, p, gp;

            // corrige hy - ez1 no lado esquerdo (campo espalhado)
            i = iniciox;
            for(j = inicioy; j <= finaly; j++)
              for(k = inicioz; k < finalz; k++)
              {
                  p = Ay.p2b(i - 1, j, Nx);
                  gp = (int)matgrade[i, j, k];
                  hy[p, k] -= chb[gp] * ez1[i];
```

```
            }

        // corrige hy + ez1 no lado direito (campo espalhado)
        i = finalx;
        for(j = inicioy; j <= finaly; j++)
          for(k = inicioz; k < finalz; k++)
          {
             p = Ay.p2b(i, j, Nx);
             gp = (int)matgrade[i, j, k];
             hy[p, k] += chb[gp] * ez1[i];
          }

        // corrige hx + ez1 no lado inferior (campo espalhado)
        j = inicioy;
        for(i = iniciox; i <= finalx; i++)
          for(k = inicioz; k < finalz; k++)
          {
             p = Ay.p2a(i, j-1, Nx);
             gp = (int)matgrade[i, j-1, k];
             hx[p, k] += chb[gp] * ez1[i];
          }

        // corrige hx - ez1 no lado superior (campo espalhado)
        j = finaly;
        for(i = iniciox; i <= finalx; i++)
          for(k = inicioz; k < finalz; k++)
          {
             p = Ay.p2a(i, j, Nx);
             gp = (int)matgrade[i, j, k];
             hx[p, k] -= chb[gp] * ez1[i];
          }

        // faces z sem correção
      } // final da função

      // ATUALIZA REGIÃO TFSF PARA Ez e Ex
      public static void atualiza_tfsf_ez( float[,] ez, float[,] ex, float[] ceb, float[] hy1, byte[,]
matgrade,
          int iniciox, int finalx, int inicioy,int finaly, int inicioz, int finalz, int Nx)
      {

        int i, j, k, p, gp;

        // corrige ez - hy1 no lado esquerdo (campo total)
        i = iniciox;
        for (j = inicioy; j <= finaly; j++)
           for (k = inicioz; k < finalz; k++)
           {
              p = Ay.p2a(i, j, Nx);
              gp = (int)matgrade[i, j, k];
              ez[p, k] -= ceb[gp] * hy1[i-1];
           }

        // corrige ez + hy1 no lado direito (campo total)
        i = finalx;
```

```
            for (j = inicioy; j <= finaly; j++)
               for (k = inicioz; k < finalz; k++)
               {
                  p = Ay.p2a(i, j, Nx);
                  gp = (int)matgrade[i, j, k];
                  ez[p, k] += ceb[gp] * hy1[i];
               }

            // faces y sem correção

            // corrige ex + hy1 na base (campo total)
            k = inicioz;
            for (i = iniciox; i < finalx; i++)
               for (j = inicioy; j <= finaly; j++)
               {
                  p = Ay.p2b(i, j, Nx);
                  gp = (int)matgrade[i, j, k];
                  ex[p, k] += ceb[gp] * hy1[i];
               }

            // corrige ex - hy1 no topo (campo total)
            k = finalz;
            for (i = iniciox; i < finalx; i++)
               for (j = inicioy; j <= finaly; j++)
               {
                  p = Ay.p2b(i, j, Nx);
                  gp = (int)matgrade[i, j, k];
                  ex[p, k] -= ceb[gp] * hy1[i];
               }

         } // final da função

          // INICIALIZA GRADE AUXILIAR 1D
         public static void inicializando_grade_1d(float[] ceze1, float[] cezh1, float[] chyh1, float[]
chye1,int Mx,
            int NLOSS, float max_loss, float cdtds, float imp0)
         {
            for(int m = 0; m < Mx-1; m++)
            {
               //float cos30 = (float)(Math.Cos(Math.PI / 6.0));
               float ka = 1.0f;
               float kb = ka;
            if (m < (Mx-1-NLOSS))
            {
               ceze1[m] = 1.0f;
               cezh1[m] = kb*cdtds*imp0;
               chyh1[m] = 1.0f;
               chye1[m] = ka*cdtds/imp0;
            }
            else
            {
               float depth_layer = m - (Mx - 1 - NLOSS) + 0.5f;
               float loss_factor = (float) (max_loss * Math.Pow(depth_layer / NLOSS, 2) );
               ceze1[m] = (1.0f - loss_factor) / (1.0f + loss_factor);
               cezh1[m] = kb*cdtds*imp0/(1.0f + loss_factor);
```

```
                depth_layer += 0.5f;
                loss_factor = (float) ( max_loss * Math.Pow(depth_layer / NLOSS, 2) );
                chyh1[m] = (1.0f - loss_factor)/(1.0f + loss_factor);
                chye1[m] = (ka * cdtds / imp0) / (1.0f + loss_factor);
            }
           }
        } // final da função

        // ATUALIZA GRADE 1D PARA REGIÃO TFSF
        public static void atualiza_tfsf_grade_1d(int tempo,float cdtds, int  ppw, int tipo_fonte, float
fator_at,
            float[] ez1,float[] hy1, float[] ceze1, float[] cezh1, float[] chyh1, float[] chye1,int Mx)
        {
            for(int m = 0; m < Mx-1; m++)
            {
                hy1[m] = chyh1[m] * hy1[m] + chye1[m] * (ez1[m + 1] - ez1[m]);
            }
            for(int m = 1; m < Mx-1; m++)
            {
                ez1[m] = ceze1[m] * ez1[m] + cezh1[m] * (hy1[m] - hy1[m - 1]);
            }
            // aplica fonte senoidal ou pulso para grade auxiliar 1D
            ez1[0] = Ay.fonte_1(tempo, 0.0f, cdtds, ppw, tipo_fonte, fator_at);
        } // final da função
    }
}
//////////////////////////////////////////////////////////////////
```

ARQUIVO 10: Saida.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Globalization;


namespace progCsharpe_Yee3d_matriz2d_float_2b
{
    class Saida
    {
        public static void saida1(string txsaida1, int Nx, int Ny, int pz, float[,] ez, float[,] ez_pxy)
        {
            // PÓS-PROCESSAMENTO DO CAMPO Ez 3D:para plano xy: campo ez
             for(int i = 0; i < Nx; i++)
                for(int j = 0; j < Ny; j++)
                    {
                        int p = Ay.p2a(i, j, Nx);
                        ez_pxy[i, j] = ez[p, pz];
                    }

           // ESCREVENDO NO PRIMEIRO ARQUIVO TEXTO E LENDO DESTE ARQUIVO -
funcionou
```

```
          using (StreamWriter saida_1 = new StreamWriter(txsaida1))
          {
             for (int i = 0; i < Nx; i++)
             {

                for (int j = 0; j < Ny; j++)
                {

                   saida_1.WriteLine(ez_pxy[i,j].ToString("E18",
CultureInfo.CreateSpecificCulture("en-US")));
                }

             }
          } // FINAL DA ESCRITA NO ARQUIVO TEXTO
       } // FINAL METODO SAIDA1

       public static void saida2(string txsaida2, int Nx, int Ny, int Nz, int py, float[,] ez, float[,]
ez_pxz)
       {
          // para plano xz: campo ez
           for(int i = 0; i < Nx; i++)
             for (int k = 0; k < Nz - 1; k++)
                {
                   int p = Ay.p2a(i, py, Nx);
                   ez_pxz[i, k] = ez[p, k];
                }
          // ESCREVENDO NO SEGUNDO ARQUIVO TEXTO E LENDO DESTE ARQUIVO -
funcionou
          using (StreamWriter saida_2 = new StreamWriter(txsaida2))
          {
             for (int i = 0; i < Nx; i++)
             {

                for (int k = 0; k < Nz - 1; k++)
                {

                   saida_2.WriteLine(ez_pxz[i,k].ToString("E18",
CultureInfo.CreateSpecificCulture("en-US")));
                }

             }
          } // FINAL DA ESCRITA NO ARQUIVO TEXTO
       } // FINAL METODO SAIDA2
    }
}
////////////////////////////////////////////////////////////
```

ARQUIVO 11: SaidaR.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
```

```
using System.Globalization;


namespace progCsharpe_Yee3d_matriz2d_float_2b
{
   class SaidaR
   {
      public static void saidar(string txsaidar, int tempomax, float[] ezr)
      {
         // ESCREVENDO
         using (StreamWriter saida_r = new StreamWriter(txsaidar))
         {
            for (int j = 0; j < tempomax; j++)
            {

               saida_r.WriteLine(ezr[j].ToString("E18",    CultureInfo.CreateSpecificCulture("en-
US")));
            }


         } // FINAL DA ESCRITA NO ARQUIVO TEXTO
      } // FINAL METODO saidar
   }
}
// FINAL APÊNDICE 3
//////////////////////////////////////////////////////////////
```

APÊNDICE 4. PROGRAMAS PARA VISUALIZAÇÃO E COMPARAÇÃO ENTRE AMBOS OS MÉTODOS FDTD

ESTE APÊNDICE CONTÉM 6 ARQUIVOS

ARQUIVO 1: PROGRAMA PYTHON PARA GRADE DE PRIMAS HEXAGONAIS COM VISUALIZAÇÃO DE PULSO RICKER

```
# -*- coding: cp1252 -*-
#            cp1252 ou utf-8
# programa programa_fdtd_3d_h_2c_visualizacao_1c.py em python 2.7.3
#
# AUTOR: MARINOEL JOAQUIM

# PROGRAMA FDTD 3D HEXAGONAL (VISUALIZAÇÃO)
# COM GRÁFICOS 1D PARA PLANOS XY E XZ, E FONTE NO TEMPO
#*********************************************************
# BIBLIOTECAS NECESSÁRIAS

import numpy as np
import time

#*********************************************************
tempo1 = time.time()

# INICIO DAS FUNÇÕES

# Função para hz e h3
def p2a(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx-1)
        p2 = (p1 + 1)/2
        if (p2 > PH3):
            p2 = 0
    return p2

# Função para ez e e3
def p2b(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx)
        p2 = (p1 + 1)/2
        if (p2 > PEZ):
            p2 = 0
    return p2

# Função para h2 e e1
def p2c(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
```

```
        p1 = i + (j-1)*(Nx)
        p2 = (p1)/2
        if (p2 > PH2):
            p2 = 0
    return p2

# Função para h1 e e2
def p2d(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx-1)
        p2 = (p1 + 1)/2
        if (p2 > PH1):
            p2 = 0
    return p2
#*************************************

# FUNÇÃO PARA FONTE DO TIPO PULSO RICKER OU SENOIDE
def fonte_1(tempo2,posicao,tipo_fonte,fator_at):
    if tipo_fonte == 0: # pulso ricker
        arg = np.pi*( (cdtds*tempo2 - posicao)/ppw - 1.0 )
        arg2 = arg*arg
        funcao =(1.0 - 2.0*arg2)*np.exp(-arg2)

    else:  # senóide com amortecimento inicial
        periodo = ppw/cdtds # periodo da senóide
                        # ou tempo para o pico do pulso ricker
        cte_tempo = fator_at*periodo
        funcao = ( 1.0 - np.exp(-1.0*tempo2/cte_tempo) )*\
                np.sin(2*np.pi*tempo2*cdtds/ppw)
    return funcao
#***********************************

# função para obter i,j a partir de p para campo ez
def ez_ij(p2):
    p1 = 2*p2 - 1
    j = p1/Nx + 1
    if (p1%Nx==0 and p1>=Nx):
        j = j - 1
    i = p1 - (j-1)*Nx
    return i,j
#*********************************************************

# Função de interpolação para gráfico de ez no plano xy
def pteste(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        i = 0
        j = 0
    return i, j

# Função de interpolação para gráfico de ez no plano xz
def pteste2(i,k):
    if ((i<=0) or (i>=Nx+1) or (k<=0) or (k>=Nz+1)):
        i = 0
```

```
        k = 0
    return i, k
#**********************************************************

# Função para dipolo
def dipoloz(pc,kc,comprimento,ativo_passivo):
    Nk = int(comprimento/2) + 1
    if ativo_passivo == 0:
        kinicial = 0
    else:
        kinicial = 1
    for k in range(kinicial,Nk):
        cez[pc,kc+k] = 0.0
        cezh[pc,kc+k] = 0.0
        cez[pc,kc-k] = 0.0
        cezh[pc,kc-k] = 0.0
    return

# Função para desenhar dipolo
def dipoloz_grafico(pc,kc,comprimento,ativo_passivo,vminimo):
    Nk = int(comprimento/2) + 1
    if ativo_passivo == 0:
        kinicial = 0
    else:
        kinicial = 1
    ic,jc = ez_ij(pc)
    for k in range(kinicial,Nk):
        ez_pxz[ic,kc+k] = vminimo
        ez_pxz[ic,kc-k] = vminimo

    return

#*******************************************

# FINAL DAS FUNÇÕES
#**********************************************************
#**********************************************************

# INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS
Nx = 81 # numero de pontos na direção x
Ny = 141  # numero de pontos na direção y
Nz = 71 # numero de pontos na direção z
ppw = 15 # numero de pontos por comprimento de onda
tempomax = 40

tipo_fonte = 1  # 0=pulso ricker ou 1=senoide
HARD_SOFT  = 1   # fonte: 0=hard ou 1=soft
ATIVANDO_DIPOLO_1 = 0 # 1=ativa 0=desativa
ATIVANDO_DIPOLO_2 = 0 # 1=ativa 0=desativa

rdsdz = 1.0 #np.sqrt(1.0) # razão ds/dz
fator_comp = 1.4 # em comprimentos de onda para dipolo_1

arqsaida1 = "ez_pxy_1.txt"
arqsaida2 = "ez_pxz_2.txt"
```

```
fator_at = 0.5

# FINAL DAS PRINCIPAIS VARIÁVEIS
#########################################################################

# VARIÁVEIS DE CONTROLE DOS GRÁFICOS

USAR_ESTES_ARQUIVOS = 0 # usar 1 ou 2, ou 0

MOSTRAR_SOMENTE_GRAFICOS_1D = 1 # 1= graficos 1D, 0 = não mostra
MOSTRAR_SOMENTE_GRAFICOS_2D = 1 # 1= graficos 2D, 0= não mostra

# ALTERANDO ESPESSURA DA LINHA E TAMANHO DA FONTE DE LETRAS
lw = 1.6
fs1 = 15
fs2 = 17
fs3 = 17

CORRECAO = 0 #45 # correção provisória para eixo y

nivel = 1.0      # controla nível máximo graficos 2D

MAXPOLAR_AUTOMATICO = 0 # automatico=0 ou usar Max_polarx
Max_polarx = 20 # posição do máximo no eixo x (0 a Max_polarx)
porcentagemdepx = 0.5 # para obter o maxpolar automático
#*******************************************************
#########################################################################

# LEITURA DE ARQUIVO DE ENTRADA
if USAR_ESTES_ARQUIVOS == 0:
   arquivo3 = open("entrada.txt","r")

if USAR_ESTES_ARQUIVOS == 1:
   arquivo3 = open("entrada_h2ta1.txt","r")

if USAR_ESTES_ARQUIVOS == 2:
   arquivo3 = open("entrada_h2ta2.txt","r")

Nx = int(arquivo3.readline()) #funcionou
Ny =  int(arquivo3.readline())
Nz =  int(arquivo3.readline())
ppw =  int(arquivo3.readline())
tempomax =  int(arquivo3.readline())
tipo_fonte =  int(arquivo3.readline())
HARD_SOFT =  int(arquivo3.readline())
ATIVANDO_DIPOLO_1 =  int(arquivo3.readline())
ATIVANDO_DIPOLO_2 =  int(arquivo3.readline())
rdsdz =  float(arquivo3.readline())
fator_comp =  float(arquivo3.readline())
arqsaida1 = str(arquivo3.readline())
arqsaida2 = str(arquivo3.readline())
delta_passo = int(arquivo3.readline())
fator_cdtds = float(arquivo3.readline())
fator_at = float(arquivo3.readline())
fator_geometria = int(arquivo3.readline())
```

```
arquivo3.close()

# RETIRANDO "\n" DAS STRINGS
txx1 = list(arqsaida1)
txx1.pop()
arqsaida1 = "".join(txx1)

txx2 = list(arqsaida2)
txx2.pop()
arqsaida2 = "".join(txx2)

if USAR_ESTES_ARQUIVOS == 0:
   arqsaida1 = "ez_pxy_1.txt"
   arqsaida2 = "ez_pxz_2.txt"

if USAR_ESTES_ARQUIVOS == 1:
   arqsaida1 = "ez_pxy_h2ta1.txt"
   arqsaida2 = "ez_pxz_h2ta1.txt"

if USAR_ESTES_ARQUIVOS == 2:
   arqsaida1 = "ez_pxy_h2ta2.txt"
   arqsaida2 = "ez_pxz_h2ta2.txt"

cdtds = 1.0/np.sqrt(fator_cdtds) # número de Courant

# IMPRIME VALORES OBTIDOS DO ARQUIVO DE ENTRADA
print 'Nx =',Nx
print 'Ny =',Ny
print 'Nz =',Nz
print 'ppw =',ppw
print 'tempomax =',tempomax
print 'tipo_fonte =',tipo_fonte
print 'HARD_SOFT =',HARD_SOFT
print 'ATIVANDO_DIPOLO_1 =',ATIVANDO_DIPOLO_1
print 'ATIVANDO_DIPOLO_2 =',ATIVANDO_DIPOLO_2
print 'rdsdz =', rdsdz
print 'cdtds =', cdtds
print 'fator comprimento dipolo  =',fator_comp
print 'fator atenuação (senóide) =',fator_at
print 'fator geometria (1, 2, 3) =',fator_geometria
print 'arqsaida1 =',arqsaida1
print 'arqsaida2 =',arqsaida2
# FINAL DA LEITURA


#*************************************

razaodsdz = round(rdsdz,2) # para gráfico
CFL = int(cdtds*100)/100.0 # número de Courant para gráfico


ativo = 1  # tipo dipolo_1
passivo = 0
comprimento_dipolo = int(fator_comp*rdsdz*ppw)
compdipolo = int(fator_comp*100)/100.0 # para gráfico
```

```
imp0 = 120*np.pi

#************************************************


if (fator_geometria <= 2):
   ds = 1.0  # dimensão dos lados do hexagono 2D
   dx = ds*np.sqrt(3.0)/2.0 # tamanho da célula na direção x
   dy = ds/2.0   # tamanho da célula na direção y
   dz = ds/rdsdz
   Lx = 1.0*(Nx)*dx  # comprimento na direção x
   Ly = 1.0*(Ny)*dy  # comprimento na direção y
   Lz = 1.0*(Nz)*dz  # comprimento na direção z

#******************************************************

tipo_1d = 0 # grade 1D auxiliar
tipo_3d = 3 # grade 3D

# calculando dimensões das matrizes para os campos E e H
ix = Nx/2
rx = Nx%2
jy = Ny/2
ry = Ny%2
jy2 = (Ny-1)/2
ry2 = (Ny-1)%2

PEZ = (ix + rx)*(jy + ry) + (ix)*(jy)
PH2 = (ix)*(jy + ry) + (ix + rx)*(jy)
PH1 = (ix)*(jy2 + ry2) + (ix)*(jy2)
PH3 = PH1
#print 'PEZ= ',PEZ

# campos

# matrizes para os gráficos de Ez nos planos xy e xz
ez_pxy = np.zeros((Nx+1,Ny+1))
ez_pxz = np.zeros((Nx+1,Nz+1))

ez_pxy2 = np.zeros((Ny+1,Nx+1)) # transposta de ez_pxy
ez_pxz2 = np.zeros((Nz+1,Nx+1)) # transposta de ez_pxz

#************************************
px = Nx/2 + 1# posição da fonte ricker
py = Ny/2 + 1
pz = Nz/2 + 1
pcentro = p2b(px,py)
#*****************************************

# textos para gráficos
texto2 =  'dim='+str(Nx)+'x'+str(Ny)+'x'+str(Nz)+\
          '  ppw='+str(ppw)+'  CFL='+str(CFL)+\
          '  ds/dz='+str(razaodsdz)
if tipo_fonte == 0:
   texto1 = u'p_ricker '
```

```
else:
   texto1 = u'senóide '
if HARD_SOFT == 0:
   texto1 = texto1 + 'hard '
else:
   texto1 = texto1 + 'soft '
if ATIVANDO_DIPOLO_1 == 1 :
   texto3 = '  dipolo '+str(compdipolo)
else:
   texto3 = ''
textof = texto1 + texto2 + texto3
#************************************

# testando a fonte sozinha no tempo
ez_teste = np.zeros((tempomax))
tempo_teste = np.zeros((tempomax))

#******************************

# INICIO DOS CÁLCULOS PARA GRADE HEXAGONAL 3D

print u'\nINICIO FDTD'
print

# inicio loop do tempo

for tempo in range(0,tempomax):

   tempo_teste[tempo] = tempo
   ez_teste[tempo] = fonte_1(tempo,0.0,tipo_fonte,fator_at)

   #print '\n',tempo+1,'/',tempomax

# final loop do tempo
#************************************************

#*****************************************

# LEITURA DAS MATRIZES PARA PLANOS XY E XZ
arquivo4 = open(arqsaida1,"r")
#linha = None
for i in range(0,Nx+1):
    for j in range(0,Ny+1):
       linha = arquivo4.readline() #funcionou
       ez_pxy[i,j] = float(linha)
arquivo4.close()

arquivo5 = open(arqsaida2,"r")
#linha = None
for i in range(0,Nx+1):
    for j in range(0,Nz+1):
       linha = arquivo5.readline() #funcionou
       ez_pxz[i,j] = float(linha)
arquivo5.close()
```

```
#*******************************************
# eliminando valores zero das matrizes para os gráficos de ez
for i in range(1,Nx+1): # plano xy
    for j in range(1,Ny+1):
        if (j%2==0 and i%2==1 and i<=Nx) or\
           (j%2==1 and i%2==0 and i<=Nx-1):
              inx, jnx = pteste(i,j+1)
              isx, jsx = pteste(i,j-1)
              ie, je   = pteste(i+1,j)
              iw, jw   = pteste(i-1,j)
              ez_pxy[i,j] = 0.25*(ez_pxy[inx,jnx] + ez_pxy[isx,jsx] +\
                        ez_pxy[ie,je] + ez_pxy[iw,jw])
j = py
for i in range(1,Nx+1):  # plano xz
    for k in range(1,Nz+1):
        if (j%2==0 and i%2==1 and i<=Nx) or\
           (j%2==1 and i%2==0 and i<=Nx-1):
              ie, ke   = pteste2(i+1,k)
              iw, kw   = pteste2(i-1,k)
              ez_pxz[i,k] = 0.5*(ez_pxz[ie,ke] + ez_pxz[iw,kw])

#***********************************************

# gerando gráficos 1D a partir dos gráficos 2D
Nx2 = Nx - px
Ny2 = Ny - py
Nz2 = Nz - pz
ez_x1 = np.zeros((Nx2+1))
ez_y1 = np.zeros((Ny2+1))
ez_x2 = np.zeros((Nx2+1))
ez_z2 = np.zeros((Nz2+1))
x = np.zeros((Nx2+1))
y = np.zeros((Ny2+1))
z = np.zeros((Nz2+1))

for i in range(0,Nx2+1):
    x[i] = i*dx
    ez_x1[i] = ez_pxy[px+i,py] # plano xy
    ez_x2[i] = ez_pxz[px+i,pz] # plano xz

for j in range(0,Ny2+1):
    y[j] = j*dy
    ez_y1[j] = ez_pxy[px,py+j] # plano xy
    if (j > (Ny2+1 - CORRECAO)): # correção provisória
        ez_y1[j] = 0.0


for k in range(0,Nz2+1):
    z[k] = k*dz
    ez_z2[k] = ez_pxz[px,pz+k] # plano xz


if Nx < Ny: # diagonal do plano xy
    Nd = Nx2
else:
```

```
    Nd = Ny2
ez_d1 = np.zeros((Nd+1))
s = np.zeros((Nd+1))

for k in range(0,Nd+1):
    s[k] = k*ds
    ez_d1[k] = ez_pxy[px+k,py+k]

if Nx < Nz: # diagonal do plano xz
    Nd2 = Nx2
else:
    Nd2 = Nz2
ez_d2 = np.zeros((Nd2+1))
s2 = np.zeros((Nd2+1))

ds2 = np.sqrt(dx**2 + dz**2)
for k in range(0,Nd2+1):
    s2[k] = k*ds2
    ez_d2[k] = ez_pxz[px+k,pz+k]

#*****************************************
# GRAVANDO GRAFICOS 1D EM 3 SAIDAS
dados = [Nx2,Nz2,Nd2,ds,rdsdz]

arquivow = open("saida_dados_h.txt","w")
for i in range(0,5):
    arquivow.write("%f\n" % dados[i])  # dados plano xz
arquivow.close()

arquivox = open("saida_ez_x2h.txt","w")
for i in range(0,Nx2+1):
    arquivox.write("%12.11f\n" % ez_x2[i])  # plano xz
arquivox.close()

arquivoz = open("saida_ez_z2h.txt","w")
for k in range(0,Nz2+1):
    arquivoz.write("%12.11f\n" % ez_z2[k])  # plano xz
arquivoz.close()

arquivod = open("saida_ez_d2h.txt","w")
for k in range(0,Nd2+1):
    arquivod.write("%12.11f\n" % ez_d2[k])  # plano xz
arquivod.close()

#########################################################################
# desenhando dipolo no plano xz
vminimo = -abs(ez_pxz).max()
if ATIVANDO_DIPOLO_1 == 1:
    dipoloz_grafico(pcentro,pz,comprimento_dipolo,ativo,vminimo)

if ATIVANDO_DIPOLO_2 == 1:
    dipoloz_grafico(pcentro+ppw/4,pz,ppw/2+2,0,vminimo)
#*************************************************

ez_pxy2 = ez_pxy.T # transposta para corrigir imagem
```

```
ez_pxz2 = ez_pxz.T # transposta para corrigir imagem

#*********************************************************

# medindo o tempo da simulação
tempo2 = time.time()
delta_t1 = tempo2 - tempo1
delta_min = int(delta_t1)/60
delta_seg = int(delta_t1)%60
#print'Tempo =', delta_t1,'segundos'
#print'Tempo =', delta_min,'min e',delta_seg,'seg'
tempomin = delta_t1/(8.0*tempomax*Nx*Ny*Nz)
#print 'Tempo_min =',tempomin,'seg'
print
#print'Tempo estimado =', delta_min2,'min e',delta_seg2,'seg'

if ATIVA_POLAR == 1:
    print 'px = ',px
    print 'nxp = ',nxp
    print 'n = ',n
    print 'raio = ',raio
#print'\n'
print u'FINAL'
#print ez_y1

#*****************************************

# PLOTANDO CAMPOS Ez PARA O MODO 3D
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

# nearest ou bilinear - funcionou
# cm.jet ou cm.RdYlGn ou cm.winter ou cm.hot
# ficou imagem correta com shape=None e com extent=[0,Lx,0,Ly]
# mas shape inverteu eixos x e y
# no entanto usando a transposta de ez a imagem ficou com eixos corretos
fs1 = 16
fs2 = 17
fs3 = 17
fse = 13
lw = 1.6
# GRÁFICOS 2D
if MOSTRAR_SOMENTE_GRAFICOS_2D == 1:
    # plano xy
    plt.figure()
    im = plt.imshow(ez_pxy2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Ly], shape=None,
                vmax= nivel*abs(ez_pxy2).max(), vmin= -nivel*abs(ez_pxy2).max()  )

    plt.colorbar(im) # funcionou
    plt.title(u'HEX 3D: Ez (V/m) NO PLANO XY (T = '+str(tempomax)+' passos)',
            fontsize=fs1)
    plt.xlabel(u'Lx (metros)',fontsize=fs1)
    plt.ylabel(u'Ly (metros)',fontsize=fs1)
```

```
    #plt.show()

    # plano xz
    plt.figure()
    im2 = plt.imshow(ez_pxz2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Lz], shape=None,
                vmax= nivel*abs(ez_pxz2).max(), vmin= -nivel*abs(ez_pxz2).max()  )

    plt.colorbar(im2) # funcionou
    plt.title(u'HEX 3D: Ez (V/m) NO PLANO XZ (T = '+str(tempomax)+' passos)',
            fontsize=fs1)
    plt.xlabel(u'Lx (metros)',fontsize=fs1)
    plt.ylabel(u'Lz (metros)',fontsize=fs1)
    #plt.show()

#*********************************************
if MOSTRAR_SOMENTE_GRAFICOS_1D == 1:
    # GRÁFICOS 1D
    # no plano xy
    plt.figure()
    plt.tick_params(labelsize = fse)
    plt.plot(x,ez_x1,'r',linewidth=lw)
    plt.axis([0.,130.,-.0004, .0003])
    plt.grid()
    plt.xlabel(u'Eixos x/y/d (metros)',fontsize=fs2)
    plt.ylabel(u'Amplitude ( V/m )',fontsize=fs2)
    plt.title(u'HEX 3D: CURVAS 1D DE CAMPOS Ez NO PLANO XY (TEMPO =
'+str(tempomax)+' passos)',
            fontsize=fs2)
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
    plt.plot(y,ez_y1,'b-',linewidth=lw)
    plt.plot(s,ez_d1,'k',linewidth=lw)
    texto1 = u'ez_x1 ( 0°)'
    texto2 = u'ez_y1 (90°)'
    texto3 = u'ez_d1 (30°)'
    plt.legend((texto1,texto2,texto3),loc = 'upper right')
    plt.hold(False)

    #plt.show()

    # no plano xz
    plt.figure()
    plt.tick_params(labelsize = fse)
    plt.plot(x,ez_x2,'r',linewidth=lw)
    plt.axis([0.,130.,-.0004, .0003])
    plt.grid()
    plt.xlabel(u'Eixos x/z/d (metros)',fontsize=fs2)
    plt.ylabel(u'Amplitude ( V/m )',fontsize=fs2)
    plt.title(u'HEX 3D: CURVAS 1D DE CAMPOS Ez NO PLANO XZ (TEMPO =
'+str(tempomax)+' passos)',
            fontsize=fs2)
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
```

```
    plt.plot(z,ez_z2,'b-',linewidth=lw)
    plt.plot(s2,ez_d2,'k',linewidth=lw)
    texto1 = u'ez_x2 ( 0°)'
    texto2 = u'ez_z2 (90°)'
    texto3 = u'ez_d2 (49°)'
    plt.legend((texto1,texto2,texto3),loc = 'upper right')
    plt.hold(False)
    #plt.show()

if MOSTRAR_SOMENTE_GRAFICOS_1D == 1:
    # fonte em função do tempo
    plt.figure()
    plt.tick_params(labelsize = fse)
    plt.plot(tempo_teste,ez_teste,'k',linewidth=lw)
    plt.grid()
    #plt.xlabel(u'Eixo Tempo, '+ textof,fontsize=fs2)
    plt.xlabel(u'Tempo ( número de passos ) ',fontsize=fs2)
    plt.ylabel(u'Amplitude ( V )',fontsize=fs2)
    plt.title('PULSO RICKER PARA CAMPO Ez ( TEMPO = '+str(tempomax)+\
            ' passos )',fontsize=fs2 )
    #plt.show()

plt.show()

# FINAL DO PROGRAMA
#########################################################################

ARQUIVO 2: PROGRAMA PYTHON PARA GRADE YEE
COM VISUALIZAÇÃO DE PULSO RICKER

# -*- coding: cp1252 -*-
#            cp1252 ou utf-8
# programa programa_fdtd_3d_1f_visualizacao_yee_1a.py em python 2.7.3
#
# AUTOR: MARINOEL JOAQUIM

# PROGRAMA VISUALIZAÇÃO GRÁFICOS FDTD YEE 3D
# COM ESCRITA DE ARQUIVOS DE ENTRADA E PLANOS XY E XZ
# (3° VERSÃO, COM GRÁFICOS 1D SIMPLIFICADO E 2D)
**********************************************************
# BIBLIOTECAS NECESSÁRIAS

import numpy as np
import time

#**********************************************************
tempo1 = time.time()

# INICIO DAS FUNÇÕES
#*********************************
# FUNÇÃO PARA FONTE DO TIPO PULSO RICKER OU SENOIDE
def fonte_1(tempo2,posicao,tipo_fonte,fator_at):
    if tipo_fonte == 0: # pulso ricker
        arg = np.pi*( (cdtds*tempo2 - posicao)/ppw - 1.0 )
        arg2 = arg*arg
```

```
        funcao =(1.0 - 2.0*arg2)*np.exp(-arg2)

    else:  # senóide com amortecimento inicial
        periodo = ppw/cdtds # periodo da senóide
                        # ou tempo para o pico do pulso ricker
        cte_tempo = fator_at*periodo
        funcao = ( 1.0 - np.exp(-1.0*tempo2/cte_tempo) )*\
                np.sin(2*np.pi*tempo2*cdtds/ppw)
    return funcao
#**********************************
#**********************************************
# Função para desenhar dipolo
def dipoloz_grafico(ic,jc,kc,comprimento,ativo_passivo,vminimo):
    Nk = int(comprimento/2) + 1
    if ativo_passivo == 0:
        kinicial = 0
    else:
        kinicial = 1
    for k in range(kinicial,Nk):
        ez_pxz[ic,kc+k] = vminimo
        ez_pxz[ic,kc-k] = vminimo

    return

# FINAL DAS FUNÇÕES
#**********************************************************
##############################################################################
# INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS
M = 22 # numero de pontos na direção x
N = 21  # numero de pontos na direção y
P = 21  # numero de pontos na direção z

ppw = 10 # numero de pontos por comprimento de onda
tempomax = 4

tempomin_estimado = 4.7e-6 # em segundos
                    # depende do desempenho da CPU
tipo_fonte = 1  # 0=pulso ricker ou 1=senoide
HARD_SOFT  = 1   # fonte: 0=hard ou 1=soft
nivel = 1.0

ATIVANDO_DIPOLO_1 = 0 # 1=ativa 0=desativa
ATIVANDO_DIPOLO_2 = 0 # 1=ativa 0=desativa
fator_comp = 1.4 # para dipolo_1

#****************************************

ativo = 1  # tipo dipolo_1
passivo = 0
comprimento_dipolo = int(fator_comp*ppw)

fator_cdtds = 3.0 # nova variavel usado para compatibilidade com hex
cdtds = 1.0/np.sqrt(fator_cdtds) # número de Courant

# novas variaveis
```

```
fator_at = 0.5 # usada na fonte para o sinal senoidal

fator_geometria = 10 # usado para compatibilidade com hex
rdsdz = 1.0 # usado para compatibilidade com hex
delta_passo = 1 # usado para compatibilidade com hex

arqsaida1 = "ez_pxy_yee1.txt"
arqsaida2 = "ez_pxz_yee2.txt"

# FINAL DAS PRINCIPAIS VARIÁVEIS
#################################################################
# LENDO ARQUIVO DADOS DE ENTRADA
# VARIÁVEIS DE CONTROLE DOS GRÁFICOS

USAR_ESTES_ARQUIVOS = 0 # usar 1 ou 2, ou 0

MOSTRAR_SOMENTE_GRAFICOS_1D = 0 # 1= graficos 1D, 0 = mostra tudo

ATIVA_POLAR = 0 # 0 = desativa 1 = ativa diagrama polar
nivel = 1.0       # controla nível máximo graficos 2D

# ALTERANDO ESPESSURA DA LINHA E TAMANHO DA FONTE DE LETRAS
lw = 1.6
fs1 = 15
fs2 = 17
fs3 = 17

MAXPOLAR_AUTOMATICO = 0 # automatico=0 ou usar Max_polarx
Max_polarx = 20 # posição do máximo no eixo x (0 a Max_polarx)
porcentagemdepx = 0.7 # para obter o maxpolar automático
#*******************************************************


# LEITURA DE ARQUIVO DE ENTRADA
if USAR_ESTES_ARQUIVOS == 0:
    arquivo3 = open("entrada2.txt","r")

if USAR_ESTES_ARQUIVOS == 1:
    arquivo3 = open("entrada_ta1.txt","r")

if USAR_ESTES_ARQUIVOS == 2:
    arquivo3 = open("entrada_ta2.txt","r")

Nx = int(arquivo3.readline()) #funcionou
Ny =  int(arquivo3.readline())
Nz =  int(arquivo3.readline())
ppw =  int(arquivo3.readline())
tempomax =  int(arquivo3.readline())
tipo_fonte =  int(arquivo3.readline())
HARD_SOFT =  int(arquivo3.readline())
ATIVANDO_DIPOLO_1 =  int(arquivo3.readline())
ATIVANDO_DIPOLO_2 =  int(arquivo3.readline())
rdsdz =  float(arquivo3.readline())
fator_comp =  float(arquivo3.readline())
arqsaida1 = str(arquivo3.readline())
```

```
arqsaida2 = str(arquivo3.readline())
delta_passo = int(arquivo3.readline())
fator_cdtds = float(arquivo3.readline())
fator_at = float(arquivo3.readline())
fator_geometria = int(arquivo3.readline())
arquivo3.close()

# RETIRANDO "\n" DAS STRINGS
txx1 = list(arqsaida1)
txx1.pop()
arqsaida1 = "".join(txx1)

txx2 = list(arqsaida2)
txx2.pop()
arqsaida2 = "".join(txx2)

if USAR_ESTES_ARQUIVOS == 0:
    arqsaida1 = "ez_pxy_yee1.txt"
    arqsaida2 = "ez_pxz_yee2.txt"

if USAR_ESTES_ARQUIVOS == 1:
    arqsaida1 = "ez_pxy_1_ta1.txt"
    arqsaida2 = "ez_pxz_2_ta1.txt"

if USAR_ESTES_ARQUIVOS == 2:
    arqsaida1 = "ez_pxy_1_ta2.txt"
    arqsaida2 = "ez_pxz_2_ta2.txt"

cdtds = 1.0/np.sqrt(fator_cdtds) # número de Courant

# IMPRIME VALORES OBTIDOS DO ARQUIVO DE ENTRADA
print 'Nx =',Nx
print 'Ny =',Ny
print 'Nz =',Nz
print 'ppw =',ppw
print 'tempomax =',tempomax
print 'tipo_fonte =',tipo_fonte
print 'HARD_SOFT =',HARD_SOFT
print 'ATIVANDO_DIPOLO_1 =',ATIVANDO_DIPOLO_1
print 'ATIVANDO_DIPOLO_2 =',ATIVANDO_DIPOLO_2
print 'rdsdz =', rdsdz
print 'cdtds =', cdtds
print 'fator comprimento dipolo  =',fator_comp
print 'fator atenuação (senóide) =',fator_at
print 'fator geometria (1, 2, 3) =',fator_geometria
print 'arqsaida1 =',arqsaida1
print 'arqsaida2 =',arqsaida2
# FINAL DA LEITURA

CFL = int(cdtds*100)/100.0 # número de Courant para gráfico
compdipolo = int(fator_comp*100)/100.0 # para gráfico

#****************************************

M = Nx # código adicional invertido para funcionar
```

```
N = Ny
P = Nz

px = M/2 + 1 # posição da fonte ricker
py = N/2 + 1
pz = P/2 + 1
NLOSS = 20 # numero de pontos (a mais)
        # na grade 1D usada para o campo total
Mx = M + NLOSS # numero de pontos para a grade 1D auxilar

# matrizes para os gráficos de Ez nos planos xy e xz
ez_pxy = np.zeros((Nx,Ny))
ez_pxz = np.zeros((Nx,Nz-1))

ez_pxy2 = np.zeros((Ny,Nx)) # transposta de ez_pxy
ez_pxz2 = np.zeros((Nz-1,Nx)) # transposta de ez_pxz

#************************************

# ENTRADA DE DADOS

#************************************************
tipo_1d = 0 # grade 1D auxiliar
tipo_3d = 3 # grade 3D
imp0 = 377.0 # impedância característica no vácuo

max_loss = 0.35 # fator de perda para grade 1D

firstx = 5 # define região de campo total
firsty = 5
lastx = 95
lasty = 75

dx = 1.0 # tamanho da célula na direção x
dy = dx  # tamanho da célula na direção y
dz = dx
Lx = (M-1)*dx # comprimento na direção x
Ly = (N-1)*dy # comprimento na direção y
Lz = (P-1)*dz

# textos para gráficos
texto2 =  'dim='+str(Nx)+'x'+str(Ny)+'x'+str(Nz-1)+\
          '  ppw='+str(ppw)+'  CFL='+str(CFL) +\
          '  dx=dy=dz='+str(dx)

if tipo_fonte == 0:
   texto1 = u'p_ricker '
else:
   texto1 = u'senóide '
if HARD_SOFT == 0:
   texto1 = texto1 + 'hard '
else:
   texto1 = texto1 + 'soft '
if ATIVANDO_DIPOLO_1 == 1 :
   texto3 = '  dipolo '+str(compdipolo)
```

```
else:
   texto3 = ''
textof = texto1 + texto2 + texto3

# testando a fonte sozinha no tempo
ez_teste = np.zeros((tempomax))
tempo_teste = np.zeros((tempomax))

#******************************
#******************************

# INICIO DOS CALCULOS PARA GRADE 3D

print u'\nINICIO FDTD YEE 3D'
print
tempoestimado = tempomin_estimado*(6.0*tempomax*Nx*Ny*Nz)
delta_min2 = int(tempoestimado)/60
delta_seg2 = int(tempoestimado)%60
print'Tempo estimado =', delta_min2,'min e',delta_seg2,'seg'
print


#inicializando_grade_3d()
#inicializando_grade_1d()

if ATIVANDO_DIPOLO_1 == 1:
   dipoloz(px,py,pz,comprimento_dipolo,ativo)

if ATIVANDO_DIPOLO_2 == 1:
   dipoloz(px+ppw/4,py,pz,ppw/2+2,0)


# inicio loop do tempo

for tempo in range(0,tempomax):
   #atualiza_h3d(tipo_3d)
   #atualiza_tfsf(tempo)
   #atualiza_e3d(tipo_3d)
   #ez[px,py,pz] = HARD_SOFT*ez[px,py,pz] + fonte_1(tempo,0.0,tipo_fonte,fator_at)
   tempo_teste[tempo] = tempo
   ez_teste[tempo] = fonte_1(tempo,0.0,tipo_fonte,fator_at)
   #abc3d_ordem1()
   #print '\n',tempo + 1,'/',tempomax

# final loop do tempo

#************************************
# LENDO ARQUIVOS DE ENTRADA

# LEITURA DAS MATRIZES PARA PLANOS XY E XZ
arquivo4 = open(arqsaida1,"r")
#linha = None
for i in range(0,Nx):
    for j in range(0,Ny):
       linha = arquivo4.readline() #funcionou
```

```
        ez_pxy[i,j] = float(linha)
arquivo4.close()

arquivo5 = open(arqsaida2,"r")
#linha = None
for i in range(0,Nx):
    for k in range(0,Nz-1):
        linha = arquivo5.readline() #funcionou
        ez_pxz[i,k] = float(linha)
arquivo5.close()

#*****************************************
#*************************************************

# gerando gráficos 1D a partir dos gráficos 2D
Nx2 = Nx - px -2
Ny2 = Ny - py -2
Nz2 = Nz - pz -3
ez_x1 = np.zeros((Nx2+1))
ez_y1 = np.zeros((Ny2+1))
ez_x2 = np.zeros((Nx2+1))
ez_z2 = np.zeros((Nz2+1))
x = np.zeros((Nx2+1))
y = np.zeros((Ny2+1))
z = np.zeros((Nz2+1))

for i in range(0,Nx2+1):
    x[i] = i*dx
    ez_x1[i] = ez_pxy[px+i,py] # plano xy
    ez_x2[i] = ez_pxz[px+i,pz] # plano xz
for j in range(0,Ny2+1):
    y[j] = j*dy
    ez_y1[j] = ez_pxy[px,py+j] # plano xy
for k in range(0,Nz2+1):
    z[k] = k*dz
    ez_z2[k] = ez_pxz[px,pz+k] # plano xz

if Nx < Ny: # diagonal do plano xy
    Nd = Nx2
else:
    Nd = Ny2
ez_d1 = np.zeros((Nd+1))
s = np.zeros((Nd+1))
ds = np.sqrt(dx**2 +dy**2)
for k in range(0,Nd+1):
    s[k] = k*ds
    ez_d1[k] = ez_pxy[px+k,py+k]

if Nx < Nz: # diagonal do plano xz
    Nd2 = Nx2
else:
    Nd2 = Nz2
ez_d2 = np.zeros((Nd2+1))
s2 = np.zeros((Nd2+1))
ds2 = np.sqrt(dx**2 + dz**2)
```

```
for k in range(0,Nd2+1):
    s2[k] = k*ds2
    ez_d2[k] = ez_pxz[px+k,pz+k]
#****************************************
# GRAVANDO GRAFICOS 1D EM 3 SAIDAS
dados = [Nx2,Nz2,Nd2,dx]

arquivow = open("saida_dados_y.txt","w")
for i in range(0,4):
    arquivow.write("%f\n" % dados[i])  # dados plano xz
arquivow.close()

arquivox = open("saida_ez_x2y.txt","w")
for i in range(0,Nx2+1):
    arquivox.write("%12.11f\n" % ez_x2[i])  # plano xz
arquivox.close()

arquivoz = open("saida_ez_z2y.txt","w")
for k in range(0,Nz2+1):
    arquivoz.write("%12.11f\n" % ez_z2[k])  # plano xz
arquivoz.close()

arquivod = open("saida_ez_d2y.txt","w")
for k in range(0,Nd2+1):
    arquivod.write("%12.11f\n" % ez_d2[k])  # plano xz
arquivod.close()


#****************************************
# desenhando dipolo no plano xz
vminimo = -abs(ez_pxz).max()
if ATIVANDO_DIPOLO_1 == 1:
    dipoloz_grafico(px,py,pz,comprimento_dipolo,ativo,vminimo)

if ATIVANDO_DIPOLO_2 == 1:
    dipoloz_grafico(px+ppw/4,py,pz,ppw/2+2,0,vminimo)

ez_pxy2 = ez_pxy.T # transposta para corrigir imagem
ez_pxz2 = ez_pxz.T # transposta para corrigir imagem
#****************************************

# medindo o tempo da simulação
tempo2 = time.time()
delta_t1 = tempo2 - tempo1
delta_min = int(delta_t1)/60
delta_seg = int(delta_t1)%60
print'Tempo =', delta_t1,'segundos'
print'Tempo =', delta_min,'min e',delta_seg,'seg'
tempomin = delta_t1/(8.0*tempomax*Nx*Ny*Nz)
print 'Tempo_min =',tempomin,'seg'
print
print'Tempo estimado =', delta_min2,'min e',delta_seg2,'seg'

if ATIVA_POLAR == 1:
    print 'px = ',px
```

```
    print 'nxp = ',nxp
    print 'n = ',n
    print 'raio = ',raio
    print'\n\n'
print u'FINAL'
print
print

#****************************************

# PLOTANDO CAMPOS Ez PARA O MODO 3D
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
fs1 = 16
fs2 = 17
fs3 = 17
fse = 13
lw = 1.6

if MOSTRAR_SOMENTE_GRAFICOS_1D == 0:
    # nearest ou bilinear - funcionou
    # cm.jet ou cm.RdYlGn ou cm.winter ou cm.hot
    # ficou imagem correta com shape=None e com extent=[0,Lx,0,Ly]
    # mas shape inverteu eixos x e y
    # no entanto usando a transposta de ez a imagem ficou com eixos corretos

    plt.figure()
    im = plt.imshow(ez_pxy2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Ly], shape=None,
                vmax= nivel*abs(ez_pxy2).max(), vmin= -nivel*abs(ez_pxy2).max()  )

    plt.colorbar(im) # funcionou
    plt.title(u'YEE 3D: Ez (V/m) NO PLANO XY (T = '+str(tempomax)+' passos)',
            fontsize=fs1)
    plt.xlabel(u'Lx (metros)',fontsize=fs1)
    plt.ylabel(u'Ly (metros)',fontsize=fs1)
    #plt.show()

    plt.figure()
    im2 = plt.imshow(ez_pxz2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Lz], shape=None,
                vmax= nivel*abs(ez_pxz2).max(), vmin= -nivel*abs(ez_pxz2).max()  )

    plt.colorbar(im2) # funcionou
    plt.title(u'YEE 3D: Ez (V/m) NO PLANO XZ (T = '+str(tempomax)+' passos)',
            fontsize=fs1)
    plt.xlabel(u'Lx (metros)',fontsize=fs1)
    plt.ylabel(u'Lz (metros)',fontsize=fs1)
    #plt.show()

#*****************************************
# GRÁFICOS 1D
# no plano xy
plt.figure()
```

```
plt.tick_params(labelsize = fse)
plt.plot(x,ez_x1,'r',linewidth=lw)
plt.axis([0.,130.,-.00045, .00035])
plt.grid()
plt.xlabel(u'Eixos x/y/d (metros)',fontsize=fs2)
plt.ylabel(u'Amplitude ( V/m )',fontsize=fs2)
plt.title(u'YEE 3D: CURVAS 1D DE CAMPOS Ez NO PLANO XY (TEMPO = '+str(tempomax)+'
passos)',
        fontsize=fs2)
plt.hold(True) # permite sobrepor vários
            # gráficos com dimensões diferentes
plt.plot(y,ez_y1,'b',linewidth=lw)
plt.plot(s,ez_d1,'k',linewidth=lw)
texto1 = u'ez_x1 ( 0°)'
texto2 = u'ez_y1 (90°)'
texto3 = u'ez_d1 (45°)'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)

#plt.show()

# no plano xz
plt.figure()
plt.tick_params(labelsize = fse)
plt.plot(x,ez_x2,'r',linewidth=lw)
plt.axis([0.,130.,-.00045, .00035])
plt.grid()
plt.xlabel(u'Eixos x/z/d (metros)',fontsize=fs2)
plt.ylabel(u'Amplitude ( V/m )',fontsize=fs2)
plt.title(u'YEE 3D: CURVAS 1D DE CAMPOS Ez NO PLANO XZ (TEMPO = '+str(tempomax)+'
passos)',
        fontsize=fs2)
plt.hold(True) # permite sobrepor vários
            # gráficos com dimensões diferentes

plt.plot(z,ez_z2,'b',linewidth=lw)
plt.plot(s2,ez_d2,'k',linewidth=lw)
texto1 = u'ez_x2 ( 0°)'
texto2 = u'ez_z2 (90°)'
texto3 = u'ez_d2 (45°)'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)
#plt.show()

# fonte em função do tempo
plt.figure()
plt.plot(tempo_teste,ez_teste,'k',linewidth=lw)
plt.grid()
plt.xlabel(u'Eixo Tempo, '+ textof,fontsize=fs2)
plt.ylabel(u'Amplitude',fontsize=fs2)
plt.title('Fonte para campo Ez (TEMPO = '+str(tempomax)+' passos)',
        fontsize=fs2)
#plt.show()

plt.show()# mostra todos os gráficos
```

```
# FINAL DO PROGRAMA
#####################################################################
```

ARQUIVO 3: PROGRAMA PARA VISUALIZAÇÃO CONJUNTA DE PULSOS RICKER DE AMBOS OS MÉTODOS FDTD

```
# -*- coding: cp1252 -*-
#            cp1252 ou utf-8
# programa programa_fdtd_3d_h_2c_visualizacao_1c.py em python 2.7.3
#
# AUTOR: MARINOEL JOAQUIM

#*********************************************************
# BIBLIOTECAS NECESSÁRIAS

import numpy as np
import time

#*********************************************************
tempo1 = time.time()

# INICIO DAS FUNÇÕES
#*********************************************************
#*********************************************************

# INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS
Nx = 81 # numero de pontos na direção x
Ny = 141  # numero de pontos na direção y
Nz = 71 # numero de pontos na direção z
ppw = 15 # numero de pontos por comprimento de onda
tempomax = 40

tipo_fonte = 1  # 0=pulso ricker ou 1=senoide
HARD_SOFT  = 1   # fonte: 0=hard ou 1=soft
ATIVANDO_DIPOLO_1 = 0 # 1=ativa 0=desativa
ATIVANDO_DIPOLO_2 = 0 # 1=ativa 0=desativa

rdsdz = 1.0 #np.sqrt(1.0) # razão ds/dz
fator_comp = 1.4 # em comprimentos de onda para dipolo_1

arqsaida1 = "ez_pxy_1.txt"
arqsaida2 = "ez_pxz_2.txt"
fator_at = 0.5

# FINAL DAS PRINCIPAIS VARIÁVEIS
#####################################################################
#####################################################################
#*********************************************

# VARIÁVEIS DE CONTROLE DOS GRÁFICOS

USAR_ESTES_ARQUIVOS = 2 # usar 1 ou 2, ou 0

MOSTRAR_SOMENTE_GRAFICOS_1D = 1 # 1= graficos 1D, 0 = não mostra
```

```
MOSTRAR_SOMENTE_GRAFICOS_2D = 0 # 1= graficos 2D, 0= não mostra

# ALTERANDO ESPESSURA DA LINHA E TAMANHO DA FONTE DE LETRAS
fs1 = 16
fs2 = 17
fs3 = 17
fse = 13
lw = 1.6

#*****************************************************
#######################################################################
## PARA METODO FDTD HEX
# LEITURA DE ARQUIVO DE ENTRADA (1° FORMA)
arquivo3 = open("saida_dados_h.txt","r")
Nx2_txt = list(arquivo3.readline()) #funcionou
Nx2_txt2 = Nx2_txt.pop(-1)
Nx2 = "".join(Nx2_txt)
Nx2 = int(float(Nx2))

Nz2_txt = list(arquivo3.readline()) #funcionou
Nz2_txt2 = Nz2_txt.pop(-1)
Nz2 = "".join(Nz2_txt)
Nz2 = int(float(Nz2))

Nd2_txt = list(arquivo3.readline()) #funcionou
Nd2_txt2 = Nd2_txt.pop(-1)
Nd2 = "".join(Nd2_txt)
Nd2 = int(float(Nd2))

ds_txt = list(arquivo3.readline()) #funcionou
ds_txt2 = ds_txt.pop(-1)
ds = "".join(ds_txt)
ds = float(ds)

rds_txt = list(arquivo3.readline()) #funcionou
rds_txt2 = rds_txt.pop(-1)
rdsdz = "".join(rds_txt)
rdsdz = float(rdsdz)
arquivo3.close()
#*********************************************************

# LEITURA DE ARQUIVO DE ENTRADA (2° FORMA) - MAIS SIMPLES
dados_h = np.zeros((5))
arquivo3 = open("saida_dados_h.txt","r")
for i in range(0,5):
   linha = arquivo3.readline()
   dados_h[i] = float(linha)
arquivo3.close()
print dados_h

#*************************************************
# GRÁFICOS 1D PARA HEX
ez_x2 = np.zeros((Nx2+1))
ez_z2 = np.zeros((Nz2+1))
ez_d2 = np.zeros((Nd2+1))
```

```
x = np.zeros((Nx2+1))
z = np.zeros((Nz2+1))
s2 = np.zeros((Nd2+1))

dx = ds*np.sqrt(3.0)/2.0
dz = ds
ds2 = np.sqrt(dx**2 + dz**2)

arquivox = open("saida_ez_x2h.txt","r")
for i in range(0,Nx2+1):
    linha = arquivox.readline()
    x[i] = i*dx
    ez_x2[i] = float(linha)
arquivox.close()

arquivoz = open("saida_ez_z2h.txt","r")
for i in range(0,Nz2+1):
    linha = arquivoz.readline()
    z[i] = i*dz
    ez_z2[i] = float(linha)
arquivoz.close()

arquivod = open("saida_ez_d2h.txt","r")
for i in range(0,Nd2+1):
    linha = arquivod.readline()
    s2[i] = i*ds2
    ez_d2[i] = float(linha)
arquivod.close()
#************************************************
## PARA MÉTODO FDTD YEE
dados_y = np.zeros((4))

arquivow = open("saida_dados_y.txt","r")
for i in range(0,4):
    linha = arquivow.readline()
    dados_y[i] = float(linha)
arquivow.close()
Nx2y = int(dados_y[0])
Nz2y = int(dados_y[1])
Nd2y = int(dados_y[2])
dxy = dados_y[3]
print dados_y

# GRÁFICOS 1D PARA YEE
ez_x2y = np.zeros((Nx2y+1))
ez_z2y = np.zeros((Nz2y+1))
ez_d2y = np.zeros((Nd2y+1))

xy = np.zeros((Nx2y+1))
zy = np.zeros((Nz2y+1))
s2y = np.zeros((Nd2y+1))

dzy = dxy
ds2y = np.sqrt(dxy**2 + dzy**2)
```

```
# LENDO ARQUIVOS 1D PARA YEE
arquivox = open("saida_ez_x2y.txt","r")
for i in range(0,Nx2y+1):
    linha = arquivox.readline()
    xy[i] = i*dxy
    ez_x2y[i] = float(linha)
arquivox.close()

arquivoz = open("saida_ez_z2y.txt","r")
for i in range(0,Nz2y+1):
    linha = arquivoz.readline()
    zy[i] = i*dzy
    ez_z2y[i] = float(linha)
arquivoz.close()

arquivod = open("saida_ez_d2y.txt","r")
for i in range(0,Nd2y+1):
    linha = arquivod.readline()
    s2y[i] = i*ds2y
    ez_d2y[i] = float(linha)
arquivod.close()

#************************************************
##############################################################################

# PLOTANDO CAMPOS Ez PARA O MODO 3D
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt


#*******************************************
if MOSTRAR_SOMENTE_GRAFICOS_1D == 1:
    # GRÁFICOS 1D HEX
    # no plano xz
    plt.figure()
    plt.plot(x,ez_x2,'r')
    plt.grid()
    plt.xlabel(u'Eixos x/z/d')
    plt.ylabel(u'Nível')
    plt.title('Campos HEX 1D: Ez plano xz (TEMPO = 200 passos)')
    plt.hold(True) # permite sobrepor vários
                   # gráficos com dimensões diferentes

    plt.plot(z,ez_z2,'b-')
    plt.plot(s2,ez_d2,'k')
    texto1 = u'ez_x2'
    texto2 = u'ez_z2'
    texto3 = u'ez_d2'
    plt.legend((texto1,texto2,texto3),loc = 'upper right')
    plt.hold(False)
    #plt.show()

    # GRÁFICOS 1D YEE
```

```
    # no plano xz
    plt.figure()
    plt.plot(xy,ez_x2y,'r')
    plt.grid()
    plt.xlabel(u'Eixos x/z/d')
    plt.ylabel(u'Nível')
    plt.title(u'Campos YEE 1D: Ez plano xz (TEMPO = 200 passos)')
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes

    plt.plot(zy,ez_z2y,'b-')
    plt.plot(s2y,ez_d2y,'k')
    texto1 = u'ez_x2'
    texto2 = u'ez_z2'
    texto3 = u'ez_d2'
    plt.legend((texto1,texto2,texto3),loc = 'upper right')
    plt.hold(False)
    #plt.show()
##########################################################################
    # COMPARANDO GRÁFICOS 1D HEX E YEE
    plt.figure()
    plt.tick_params(labelsize = fse)
    plt.plot(x,ez_x2,'k-',linewidth=lw)
    plt.axis([0.,130.,-.00042, .00036])
    plt.grid()
    plt.xlabel(u'Eixos x ( metros )',fontsize=fs2)
    plt.ylabel(u'Amplitude ( V/m )',fontsize=fs2)
    plt.title(u'COMPARANDO CAMPOS Ez: HEX E YEE NA DIREÇÃO X',fontsize=fs2)
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
    plt.plot(xy,ez_x2y,'r-',linewidth=lw)
    texto1 = u'eixo_x_(0°)_HEX'
    texto2 = u'eixo_x_(0°)_YEE'
    plt.legend((texto1,texto2),loc = 'upper right')
    plt.hold(False)
    #plt.show()

    plt.figure()
    plt.tick_params(labelsize = fse)
    plt.plot(z,ez_z2,'k',linewidth=lw)
    plt.axis([0.,130.,-.000016, .000022])
    plt.grid()
    plt.xlabel(u'Eixos z ( metros )',fontsize=fs2)
    plt.ylabel(u'Amplitude ( V/m )',fontsize=fs2)
    plt.title(u'COMPARANDO CAMPOS Ez: HEX E YEE NA DIREÇÃO Z',fontsize=fs2)
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
    plt.plot(zy,ez_z2y,'r-',linewidth=lw)
    texto1 = u'eixo_z_(90°)_HEX'
    texto2 = u'eixo_z_(90°)_YEE'
    plt.legend((texto1,texto2),loc = 'upper right')
    plt.hold(False)
    #plt.show()

    plt.figure()
```

```
    plt.tick_params(labelsize = fse)
    plt.plot(s2,ez_d2,'k',linewidth=lw)
    plt.axis([0.,130.,-.00022, .00017])
    plt.grid()
    plt.xlabel(u'Eixos d ( metros )',fontsize=fs2)
    plt.ylabel(u'Amplitude ( V/m )',fontsize=fs2)
    plt.title(u'COMPARANDO CAMPOS Ez: HEX E YEE NA DIREÇÃO D',fontsize=fs2)
    plt.hold(True) # permite sobrepor vários
                   # gráficos com dimensões diferentes
    plt.plot(s2y,ez_d2y,'r-',linewidth=lw)
    texto1 = u'eixo_d_(49°)_HEX'
    texto2 = u'eixo_d_(45°)_YEE'
    plt.legend((texto1,texto2),loc = 'upper right')
    plt.hold(False)
    plt.show()
# FINAL DO PROGRAMA
#############################################################
```

ARQUIVO 4: PROGRAMA PYTHON PARA GRADE DE PRIMAS HEXAGONAIS COM VISUALIZAÇÃO DE ONDA SENOIDAL

```
# -*- coding: cp1252 -*-
#            cp1252 ou utf-8
# programa programa_fdtd_3d_h_2c_visualizacao_1c.py em python 2.7.3
#
# AUTOR: MARINOEL JOAQUIM

# PROGRAMA FDTD 3D HEXAGONAL (VISUALIZAÇÃO)
# COM GRÁFICOS 1D PARA PLANOS XY E XZ, E FONTE NO TEMPO
#***********************************************************
# BIBLIOTECAS NECESSÁRIAS

import numpy as np
import time

#***********************************************************
tempo1 = time.time()

# INICIO DAS FUNÇÕES

# Função para hz e h3
def p2a(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx-1)
        p2 = (p1 + 1)/2
        if (p2 > PH3):
            p2 = 0
    return p2

# Função para ez e e3
def p2b(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
```

```
    else:
        p1 = i + (j-1)*(Nx)
        p2 = (p1 + 1)/2
        if (p2 > PEZ):
            p2 = 0
    return p2

# Função para h2 e e1
def p2c(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx)
        p2 = (p1)/2
        if (p2 > PH2):
            p2 = 0
    return p2

# Função para h1 e e2
def p2d(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx-1)
        p2 = (p1 + 1)/2
        if (p2 > PH1):
            p2 = 0
    return p2
#**************************************

# FUNÇÃO PARA FONTE DO TIPO PULSO RICKER OU SENOIDE
def fonte_1(tempo2,posicao,tipo_fonte,fator_at):
    if tipo_fonte == 0: # pulso ricker
        arg = np.pi*( (cdtds*tempo2 - posicao)/ppw - 1.0 )
        arg2 = arg*arg
        funcao =(1.0 - 2.0*arg2)*np.exp(-arg2)

    else:  # senóide com amortecimento inicial
        periodo = ppw/cdtds # periodo da senóide
                        # ou tempo para o pico do pulso ricker
        cte_tempo = fator_at*periodo
        funcao = ( 1.0 - np.exp(-1.0*tempo2/cte_tempo) )*\
                np.sin(2*np.pi*tempo2*cdtds/ppw)
    return funcao
#**********************************

# função para obter i,j a partir de p para campo ez
def ez_ij(p2):
    p1 = 2*p2 - 1
    j = p1/Nx + 1
    if (p1%Nx==0 and p1>=Nx):
        j = j - 1
    i = p1 - (j-1)*Nx
    return i,j
#******************************************************
```

```
# Função de interpolação para gráfico de ez no plano xy
def pteste(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        i = 0
        j = 0
    return i, j

# Função de interpolação para gráfico de ez no plano xz
def pteste2(i,k):
    if ((i<=0) or (i>=Nx+1) or (k<=0) or (k>=Nz+1)):
        i = 0
        k = 0
    return i, k
#*********************************************************

# Função para dipolo
def dipoloz(pc,kc,comprimento,ativo_passivo):
    Nk = int(comprimento/2) + 1
    if ativo_passivo == 0:
        kinicial = 0
    else:
        kinicial = 1
    for k in range(kinicial,Nk):
        cez[pc,kc+k] = 0.0
        cezh[pc,kc+k] = 0.0
        cez[pc,kc-k] = 0.0
        cezh[pc,kc-k] = 0.0
    return

# Função para desenhar dipolo
def dipoloz_grafico(pc,kc,comprimento,ativo_passivo,vminimo):
    Nk = int(comprimento/2) + 1
    if ativo_passivo == 0:
        kinicial = 0
    else:
        kinicial = 1
    ic,jc = ez_ij(pc)
    for k in range(kinicial,Nk):
        ez_pxz[ic,kc+k] = vminimo
        ez_pxz[ic,kc-k] = vminimo

    return

#*********************************************

# FINAL DAS FUNÇÕES
#*********************************************************

# INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS
Nx = 81 # numero de pontos na direção x
Ny = 141  # numero de pontos na direção y
Nz = 71 # numero de pontos na direção z
ppw = 15 # numero de pontos por comprimento de onda
tempomax = 40
```

```
tipo_fonte = 1  # 0=pulso ricker ou 1=senoide
HARD_SOFT  = 1   # fonte: 0=hard ou 1=soft
ATIVANDO_DIPOLO_1 = 0 # 1=ativa 0=desativa
ATIVANDO_DIPOLO_2 = 0 # 1=ativa 0=desativa

rdsdz = 1.0 #np.sqrt(1.0) # razão ds/dz
fator_comp = 1.4 # em comprimentos de onda para dipolo_1

arqsaida1 = "ez_pxy_1.txt"
arqsaida2 = "ez_pxz_2.txt"
fator_at = 0.5

# FINAL DAS PRINCIPAIS VARIÁVEIS
#########################################################################

# VARIÁVEIS DE CONTROLE DOS GRÁFICOS

USAR_ESTES_ARQUIVOS = 0 # usar 1 ou 2, ou 0

MOSTRAR_SOMENTE_GRAFICOS_1D = 1 # 1= graficos 1D, 0 = não mostra
MOSTRAR_SOMENTE_GRAFICOS_2D = 1 # 1= graficos 2D, 0= não mostra

# PARA EIXOS: 1= (x1 y1 d1 x2); 1B= (d1);  2= (z2);  3= (d2)
NP                =  2 #10.
DELTA_COMP1       =  1 #6.
NUMERO_COMPRIMENTOS1  = NP
NUMERO_COMPRIMENTOS1B = NP
NUMERO_COMPRIMENTOS2  = NP - .5
NUMERO_COMPRIMENTOS3  = NP

LIMIAR1 = 2e-5
LIMIAR2 = 5e-6
LIMIAR3 = 5e-6

CORRECAO = 0 #45 # correção provisória para eixo y

ATIVA_POLAR = 0 # 0 = desativa 1 = ativa diagrama polar
nivel = 1.0e-4      # controla nível máximo graficos 2D
# ALTERANDO ESPESSURA DA LINHA E TAMANHO DA FONTE DE LETRAS
lw = 1.6
fs1 = 15
fs2 = 17
fs3 = 17

MAXPOLAR_AUTOMATICO = 0 # automatico=0 ou usar Max_polarx
Max_polarx = 20 # posição do máximo no eixo x (0 a Max_polarx)
porcentagemdepx = 0.5 # para obter o maxpolar automático
#*****************************************************
#########################################################################

# LEITURA DE ARQUIVO DE ENTRADA
if USAR_ESTES_ARQUIVOS == 0:
    arquivo3 = open("entrada.txt","r")
```

```
if USAR_ESTES_ARQUIVOS == 1:
    arquivo3 = open("entrada_h2ta1.txt","r")

if USAR_ESTES_ARQUIVOS == 2:
    arquivo3 = open("entrada_h2ta2.txt","r")

Nx = int(arquivo3.readline()) #funcionou
Ny =  int(arquivo3.readline())
Nz =  int(arquivo3.readline())
ppw =  int(arquivo3.readline())
tempomax =  int(arquivo3.readline())
tipo_fonte =  int(arquivo3.readline())
HARD_SOFT =  int(arquivo3.readline())
ATIVANDO_DIPOLO_1 =  int(arquivo3.readline())
ATIVANDO_DIPOLO_2 =  int(arquivo3.readline())
rdsdz =  float(arquivo3.readline())
fator_comp =  float(arquivo3.readline())
arqsaida1 = str(arquivo3.readline())
arqsaida2 = str(arquivo3.readline())
delta_passo = int(arquivo3.readline())
fator_cdtds = float(arquivo3.readline())
fator_at = float(arquivo3.readline())
fator_geometria = int(arquivo3.readline())
arquivo3.close()

# RETIRANDO "\n" DAS STRINGS
txx1 = list(arqsaida1)
txx1.pop()
arqsaida1 = "".join(txx1)

txx2 = list(arqsaida2)
txx2.pop()
arqsaida2 = "".join(txx2)

if USAR_ESTES_ARQUIVOS == 0:
    arqsaida1 = "ez_pxy_1.txt"
    arqsaida2 = "ez_pxz_2.txt"

if USAR_ESTES_ARQUIVOS == 1:
    arqsaida1 = "ez_pxy_h2ta1.txt"
    arqsaida2 = "ez_pxz_h2ta1.txt"

if USAR_ESTES_ARQUIVOS == 2:
    arqsaida1 = "ez_pxy_h2ta2.txt"
    arqsaida2 = "ez_pxz_h2ta2.txt"

cdtds = 1.0/np.sqrt(fator_cdtds) # número de Courant

# IMPRIME VALORES OBTIDOS DO ARQUIVO DE ENTRADA
print 'Nx =',Nx
print 'Ny =',Ny
print 'Nz =',Nz
print 'ppw =',ppw
print 'tempomax =',tempomax
print 'tipo_fonte =',tipo_fonte
```

```
print 'HARD_SOFT =',HARD_SOFT
print 'ATIVANDO_DIPOLO_1 =',ATIVANDO_DIPOLO_1
print 'ATIVANDO_DIPOLO_2 =',ATIVANDO_DIPOLO_2
print 'rdsdz =', rdsdz
print 'cdtds =', cdtds
print 'fator comprimento dipolo  =',fator_comp
print 'fator atenuação (senóide) =',fator_at
print 'fator geometria (1, 2, 3) =',fator_geometria
print 'arqsaida1 =',arqsaida1
print 'arqsaida2 =',arqsaida2
# FINAL DA LEITURA


#************************************

razaodsdz = round(rdsdz,2) # para gráfico
CFL = int(cdtds*100)/100.0 # número de Courant para gráfico


ativo = 1  # tipo dipolo_1
passivo = 0
comprimento_dipolo = int(fator_comp*rdsdz*ppw)
compdipolo = int(fator_comp*100)/100.0 # para gráfico
imp0 = 120*np.pi

#************************************************


if (fator_geometria <= 2):
   ds = 1.0  # dimensão dos lados do hexagono 2D
   dx = ds*np.sqrt(3.0)/2.0 # tamanho da célula na direção x
   dy = ds/2.0   # tamanho da célula na direção y
   dz = ds/rdsdz
   Lx = 1.0*(Nx)*dx  # comprimento na direção x
   Ly = 1.0*(Ny)*dy  # comprimento na direção y
   Lz = 1.0*(Nz)*dz  # comprimento na direção z

#*****************************************************

tipo_1d = 0 # grade 1D auxiliar
tipo_3d = 3 # grade 3D

# calculando dimensões das matrizes para os campos E e H
ix = Nx/2
rx = Nx%2
jy = Ny/2
ry = Ny%2
jy2 = (Ny-1)/2
ry2 = (Ny-1)%2

PEZ = (ix + rx)*(jy + ry) + (ix)*(jy)
PH2 = (ix)*(jy + ry) + (ix + rx)*(jy)
PH1 = (ix)*(jy2 + ry2) + (ix)*(jy2)
PH3 = PH1
#print 'PEZ= ',PEZ
```

```
# campos

# matrizes para os gráficos de Ez nos planos xy e xz
ez_pxy = np.zeros((Nx+1,Ny+1))
ez_pxz = np.zeros((Nx+1,Nz+1))

ez_pxy2 = np.zeros((Ny+1,Nx+1)) # transposta de ez_pxy
ez_pxz2 = np.zeros((Nz+1,Nx+1)) # transposta de ez_pxz

#************************************
px = Nx/2 + 1# posição da fonte ricker
py = Ny/2 + 1
pz = Nz/2 + 1
pcentro = p2b(px,py)
#*****************************************

# textos para gráficos
texto2 =  'dim='+str(Nx)+'x'+str(Ny)+'x'+str(Nz)+\
          '  ppw='+str(ppw)+'  CFL='+str(CFL)+\
          '  ds/dz='+str(razaodsdz)
if tipo_fonte == 0:
   texto1 = u'p_ricker '
else:
   texto1 = u'senóide '
if HARD_SOFT == 0:
   texto1 = texto1 + 'hard '
else:
   texto1 = texto1 + 'soft '
if ATIVANDO_DIPOLO_1 == 1 :
   texto3 = '  dipolo '+str(compdipolo)
else:
   texto3 = ''
textof = texto1 + texto2 + texto3
#************************************

# testando a fonte sozinha no tempo
ez_teste = np.zeros((tempomax))
tempo_teste = np.zeros((tempomax))

#******************************

# INICIO DOS CÁLCULOS PARA GRADE HEXAGONAL 3D

print u'\nINICIO FDTD'
print

# inicio loop do tempo

for tempo in range(0,tempomax):

   tempo_teste[tempo] = tempo
   ez_teste[tempo] = fonte_1(tempo,0.0,tipo_fonte,fator_at)

   #print '\n',tempo+1,'/',tempomax
```

```
# final loop do tempo
#***********************************************

# LEITURA DAS MATRIZES PARA PLANOS XY E XZ
arquivo4 = open(arqsaida1,"r")
#linha = None
for i in range(0,Nx+1):
    for j in range(0,Ny+1):
       linha = arquivo4.readline() #funcionou
       ez_pxy[i,j] = float(linha)
arquivo4.close()

arquivo5 = open(arqsaida2,"r")
#linha = None
for i in range(0,Nx+1):
    for j in range(0,Nz+1):
       linha = arquivo5.readline() #funcionou
       ez_pxz[i,j] = float(linha)
arquivo5.close()

#*******************************************
# eliminando valores zero das matrizes para os gráficos de ez
for i in range(1,Nx+1): # plano xy
   for j in range(1,Ny+1):
      if (j%2==0 and i%2==1 and i<=Nx) or\
         (j%2==1 and i%2==0 and i<=Nx-1):
            inx, jnx = pteste(i,j+1)
            isx, jsx = pteste(i,j-1)
            ie, je   = pteste(i+1,j)
            iw, jw   = pteste(i-1,j)
            ez_pxy[i,j] = 0.25*(ez_pxy[inx,jnx] + ez_pxy[isx,jsx] +\
                        ez_pxy[ie,je] + ez_pxy[iw,jw])
j = py
for i in range(1,Nx+1):  # plano xz
   for k in range(1,Nz+1):
      if (j%2==0 and i%2==1 and i<=Nx) or\
         (j%2==1 and i%2==0 and i<=Nx-1):
            ie, ke   = pteste2(i+1,k)
            iw, kw   = pteste2(i-1,k)
            ez_pxz[i,k] = 0.5*(ez_pxz[ie,ke] + ez_pxz[iw,kw])

#***********************************************

# gerando gráficos 1D a partir dos gráficos 2D
Nx2 = Nx - px
Ny2 = Ny - py
Nz2 = Nz - pz
ez_x1 = np.zeros((Nx2+1))
ez_y1 = np.zeros((Ny2+1))
ez_x2 = np.zeros((Nx2+1))
ez_z2 = np.zeros((Nz2+1))
x = np.zeros((Nx2+1))
y = np.zeros((Ny2+1))
z = np.zeros((Nz2+1))
```

```
for i in range(0,Nx2+1):
    x[i] = i*dx
    ez_x1[i] = ez_pxy[px+i,py] # plano xy
    ez_x2[i] = ez_pxz[px+i,pz] # plano xz

for j in range(0,Ny2+1):
    y[j] = j*dy
    ez_y1[j] = ez_pxy[px,py+j] # plano xy
    if (j > (Ny2+1 - CORRECAO)): # correção provisória
        ez_y1[j] = 0.0


for k in range(0,Nz2+1):
    z[k] = k*dz
    ez_z2[k] = ez_pxz[px,pz+k] # plano xz


if Nx < Ny: # diagonal do plano xy
    Nd = Nx2
else:
    Nd = Ny2
ez_d1 = np.zeros((Nd+1))
s = np.zeros((Nd+1))

for k in range(0,Nd+1):
    s[k] = k*ds
    ez_d1[k] = ez_pxy[px+k,py+k]

if Nx < Nz: # diagonal do plano xz
    Nd2 = Nx2
else:
    Nd2 = Nz2
ez_d2 = np.zeros((Nd2+1))
s2 = np.zeros((Nd2+1))

ds2 = np.sqrt(dx**2 + dz**2)
for k in range(0,Nd2+1):
    s2[k] = k*ds2
    ez_d2[k] = ez_pxz[px+k,pz+k]

#*****************************************
# GRAVANDO GRAFICOS 1D EM 3 SAIDAS
dados = [Nx2,Nz2,Nd2,ds,rdsdz]

arquivow = open("saida_dados_h.txt","w")
for i in range(0,5):
    arquivow.write("%f\n" % dados[i])  # dados plano xz
arquivow.close()

arquivox = open("saida_ez_x2h.txt","w")
for i in range(0,Nx2+1):
    arquivox.write("%12.11f\n" % ez_x2[i])  # plano xz
arquivox.close()
```

```
arquivoz = open("saida_ez_z2h.txt","w")
for k in range(0,Nz2+1):
   arquivoz.write("%12.11f\n" % ez_z2[k])  # plano xz
arquivoz.close()

arquivod = open("saida_ez_d2h.txt","w")
for k in range(0,Nd2+1):
   arquivod.write("%12.11f\n" % ez_d2[k])  # plano xz
arquivod.close()

#########################################################################

# desenhando dipolo no plano xz
vminimo = -abs(ez_pxz).max()
if ATIVANDO_DIPOLO_1 == 1:
   dipoloz_grafico(pcentro,pz,comprimento_dipolo,ativo,vminimo)

if ATIVANDO_DIPOLO_2 == 1:
   dipoloz_grafico(pcentro+ppw/4,pz,ppw/2+2,0,vminimo)
#************************************************

ez_pxy2 = ez_pxy.T # transposta para corrigir imagem
ez_pxz2 = ez_pxz.T # transposta para corrigir imagem

#********************************************************

# medindo o tempo da simulação
tempo2 = time.time()
delta_t1 = tempo2 - tempo1
delta_min = int(delta_t1)/60
delta_seg = int(delta_t1)%60
#print'Tempo =', delta_t1,'segundos'
#print'Tempo =', delta_min,'min e',delta_seg,'seg'
tempomin = delta_t1/(8.0*tempomax*Nx*Ny*Nz)
#print 'Tempo_min =',tempomin,'seg'
print
#print'Tempo estimado =', delta_min2,'min e',delta_seg2,'seg'

if ATIVA_POLAR == 1:
   print 'px = ',px
   print 'nxp = ',nxp
   print 'n = ',n
   print 'raio = ',raio
#print'\n'
print u'FINAL'
#print ez_y1

#***************************************

# PLOTANDO CAMPOS Ez PARA O MODO 3D
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
fs1 = 16
fs2 = 17
```

```
fs3 = 17
fse = 13
lw = 1.6

# nearest ou bilinear - funcionou
# cm.jet ou cm.RdYlGn ou cm.winter ou cm.hot
# ficou imagem correta com shape=None e com extent=[0,Lx,0,Ly]
# mas shape inverteu eixos x e y
# no entanto usando a transposta de ez a imagem ficou com eixos corretos

# GRÁFICOS 2D
if MOSTRAR_SOMENTE_GRAFICOS_2D == 1:
   # plano xy
   plt.figure()
   im = plt.imshow(ez_pxy2, interpolation='nearest', cmap=cm.jet,
               origin='lower', extent=[0,Lx,0,Ly], shape=None,
               vmax= nivel*abs(ez_pxy2).max(), vmin= -nivel*abs(ez_pxy2).max()  )

   plt.colorbar(im) # funcionou
   plt.title(u'HEX 3D: Ez (V/m) NO PLANO XY (T = '+str(tempomax)+' passos)')
   plt.xlabel(u'Lx (metros)')
   plt.ylabel(u'Ly (metros)')
   #plt.show()

   # plano xz
   plt.figure()
   im2 = plt.imshow(ez_pxz2, interpolation='nearest', cmap=cm.jet,
               origin='lower', extent=[0,Lx,0,Lz], shape=None,
               vmax= nivel*abs(ez_pxz2).max(), vmin= -nivel*abs(ez_pxz2).max()  )

   plt.colorbar(im2) # funcionou
   plt.title(u'HEX 3D: Ez (V/m) NO PLANO XZ (T = '+str(tempomax)+' passos)')
   plt.xlabel(u'Lx (metros)')
   plt.ylabel(u'Lz (metros)')
   #plt.show()

#*********************************************
if MOSTRAR_SOMENTE_GRAFICOS_1D == 1:
   # GRÁFICOS 1D
   # no plano xy
   plt.figure()
   plt.tick_params(labelsize = fse)
   plt.plot(x,ez_x1,'r',linewidth=lw)
   plt.axis([0.,130.,-.0008, .0012])
   plt.grid()
   plt.xlabel(u'Eixos x/y/d (metros)',fontsize=fs2)
   plt.ylabel(u'Amplitude ( V/m )',fontsize=fs2)
   plt.title(u'HEX 3D: CURVAS 1D DE CAMPOS Ez NO PLANO XY (TEMPO =
'+str(tempomax)+' passos)',
            fontsize=fs2)
   plt.hold(True) # permite sobrepor vários
               # gráficos com dimensões diferentes
   plt.plot(y,ez_y1,'b-',linewidth=lw)
   plt.plot(s,ez_d1,'k',linewidth=lw)
   texto1 = u'ez_x1_( 0°)'
```

```
    texto2 = u'ez_y1_(90°)'
    texto3 = u'ez_d1_(30°)'
    plt.legend((texto1,texto2,texto3),loc = 'upper right')
    plt.hold(False)

    #plt.show()

    # no plano xz
    plt.figure()
    plt.plot(x,ez_x2,'r')
    plt.grid()
    plt.xlabel(u'Eixos x/z/d, '+ textof)
    plt.ylabel(u'Nível')
    plt.title(u'Campos 1D: Ez plano xz (TEMPO = '+str(tempomax)+' passos)')
    plt.hold(True) # permite sobrepor vários
                   # gráficos com dimensões diferentes

    plt.plot(z,ez_z2,'b-')
    plt.plot(s2,ez_d2,'k')
    texto1 = u'ez_x2'
    texto2 = u'ez_z2'
    texto3 = u'ez_d2'
    plt.legend((texto1,texto2,texto3),loc = 'upper right')
    plt.hold(False)
    #plt.show()

if MOSTRAR_SOMENTE_GRAFICOS_1D == 1:
    # fonte em função do tempo
    plt.figure()
    plt.tick_params(labelsize = fse)
    plt.plot(tempo_teste,ez_teste,'k',linewidth=lw)
    plt.grid()
    #plt.xlabel(u'Eixo Tempo, '+ textof,fontsize=fs2)
    plt.xlabel(u'Tempo ( número de passos )',fontsize=fs2)
    plt.ylabel(u'Amplitude ( V )',fontsize=fs2)
    plt.title(u'FONTE SENOIDAL PARA CAMPO Ez ( TEMPO = '+str(tempomax)+\
            u' passos, FATOR_ATENUAÇÃO = '+str(fator_at)+' )',fontsize=fs2 )
    #plt.show()

plt.show()

# FINAL DO PROGRAMA
#########################################################################
```

ARQUIVO 5: PROGRAMA PYTHON PARA GRADE YEE
COM VISUALIZAÇÃO DE ONDA SENOIDAL

```
# -*- coding: cp1252 -*-
#           cp1252 ou utf-8
# programa programa_fdtd_3d_1f_visualizacao_yee_1a.py em python 2.7.3
#
# AUTOR: MARINOEL JOAQUIM

# PROGRAMA VISUALIZAÇÃO GRÁFICOS FDTD YEE 3D
# COM ESCRITA DE ARQUIVOS DE ENTRADA E PLANOS XY E XZ
```

```
# (3° VERSÃO, COM GRÁFICOS 1D SIMPLIFICADO E 2D)
#*************************************************************
# BIBLIOTECAS NECESSÁRIAS

import numpy as np
import time

#*************************************************************
tempo1 = time.time()

# INICIO DAS FUNÇÕES
#*********************************
#*************************************************************

# FUNÇÃO PARA FONTE DO TIPO PULSO RICKER OU SENOIDE
def fonte_1(tempo2,posicao,tipo_fonte,fator_at):
    if tipo_fonte == 0: # pulso ricker
        arg = np.pi*( (cdtds*tempo2 - posicao)/ppw - 1.0 )
        arg2 = arg*arg
        funcao =(1.0 - 2.0*arg2)*np.exp(-arg2)

    else:  # senóide com amortecimento inicial
        periodo = ppw/cdtds # periodo da senóide
                        # ou tempo para o pico do pulso ricker
        cte_tempo = fator_at*periodo
        funcao = ( 1.0 - np.exp(-1.0*tempo2/cte_tempo) )*\
                np.sin(2*np.pi*tempo2*cdtds/ppw)
    return funcao
#*********************************
#*********************************************
# Função para desenhar dipolo
def dipoloz_grafico(ic,jc,kc,comprimento,ativo_passivo,vminimo):
    Nk = int(comprimento/2) + 1
    if ativo_passivo == 0:
        kinicial = 0
    else:
        kinicial = 1
    for k in range(kinicial,Nk):
        ez_pxz[ic,kc+k] = vminimo
        ez_pxz[ic,kc-k] = vminimo

    return

# FINAL DAS FUNÇÕES
#*************************************************************
###############################################################################
# INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS
M = 22 # numero de pontos na direção x
N = 21  # numero de pontos na direção y
P = 21  # numero de pontos na direção z

ppw = 10 # numero de pontos por comprimento de onda
tempomax = 4

tempomin_estimado = 4.7e-6 # em segundos
```

```
                    # depende do desempenho da CPU
tipo_fonte = 1  # 0=pulso ricker ou 1=senoide
HARD_SOFT  = 1   # fonte: 0=hard ou 1=soft
nivel = 1.0

ATIVANDO_DIPOLO_1 = 0 # 1=ativa 0=desativa
ATIVANDO_DIPOLO_2 = 0 # 1=ativa 0=desativa
fator_comp = 1.4 # para dipolo_1

#*************************************

ativo = 1  # tipo dipolo_1
passivo = 0
comprimento_dipolo = int(fator_comp*ppw)

fator_cdtds = 3.0 # nova variavel usado para compatibilidade com hex
cdtds = 1.0/np.sqrt(fator_cdtds) # número de Courant

# novas variaveis
fator_at = 0.5 # usada na fonte para o sinal senoidal

fator_geometria = 10 # usado para compatibilidade com hex
rdsdz = 1.0 # usado para compatibilidade com hex
delta_passo = 1 # usado para compatibilidade com hex

arqsaida1 = "ez_pxy_yee1.txt"
arqsaida2 = "ez_pxz_yee2.txt"

# FINAL DAS PRINCIPAIS VARIÁVEIS
#####################################################################
# LENDO ARQUIVO DADOS DE ENTRADA
# VARIÁVEIS DE CONTROLE DOS GRÁFICOS

USAR_ESTES_ARQUIVOS = 0 # usar 1 ou 2, ou 0

MOSTRAR_SOMENTE_GRAFICOS_1D = 0 # 1= graficos 1D, 0 = mostra tudo

ATIVA_POLAR = 0 # 0 = desativa 1 = ativa diagrama polar
nivel = 1.0e-4       # controla nível máximo graficos 2D

# ALTERANDO ESPESSURA DA LINHA E TAMANHO DA FONTE DE LETRAS
lw = 1.6
fs1 = 15
fs2 = 17
fs3 = 17

MAXPOLAR_AUTOMATICO = 0 # automatico=0 ou usar Max_polarx
Max_polarx = 20 # posição do máximo no eixo x (0 a Max_polarx)
porcentagemdepx = 0.7 # para obter o maxpolar automático
#*****************************************************



# LEITURA DE ARQUIVO DE ENTRADA
if USAR_ESTES_ARQUIVOS == 0:
   arquivo3 = open("entrada2.txt","r")
```

```
if USAR_ESTES_ARQUIVOS == 1:
    arquivo3 = open("entrada_ta1.txt","r")

if USAR_ESTES_ARQUIVOS == 2:
    arquivo3 = open("entrada_ta2.txt","r")

Nx = int(arquivo3.readline()) #funcionou
Ny =  int(arquivo3.readline())
Nz =  int(arquivo3.readline())
ppw =  int(arquivo3.readline())
tempomax =  int(arquivo3.readline())
tipo_fonte =  int(arquivo3.readline())
HARD_SOFT =  int(arquivo3.readline())
ATIVANDO_DIPOLO_1 =  int(arquivo3.readline())
ATIVANDO_DIPOLO_2 =  int(arquivo3.readline())
rdsdz =  float(arquivo3.readline())
fator_comp =  float(arquivo3.readline())
arqsaida1 = str(arquivo3.readline())
arqsaida2 = str(arquivo3.readline())
delta_passo = int(arquivo3.readline())
fator_cdtds = float(arquivo3.readline())
fator_at = float(arquivo3.readline())
fator_geometria = int(arquivo3.readline())
arquivo3.close()

# RETIRANDO "\n" DAS STRINGS
txx1 = list(arqsaida1)
txx1.pop()
arqsaida1 = "".join(txx1)

txx2 = list(arqsaida2)
txx2.pop()
arqsaida2 = "".join(txx2)

if USAR_ESTES_ARQUIVOS == 0:
    arqsaida1 = "ez_pxy_yee1.txt"
    arqsaida2 = "ez_pxz_yee2.txt"

if USAR_ESTES_ARQUIVOS == 1:
    arqsaida1 = "ez_pxy_1_ta1.txt"
    arqsaida2 = "ez_pxz_2_ta1.txt"

if USAR_ESTES_ARQUIVOS == 2:
    arqsaida1 = "ez_pxy_1_ta2.txt"
    arqsaida2 = "ez_pxz_2_ta2.txt"

cdtds = 1.0/np.sqrt(fator_cdtds) # número de Courant

# IMPRIME VALORES OBTIDOS DO ARQUIVO DE ENTRADA
print 'Nx =',Nx
print 'Ny =',Ny
print 'Nz =',Nz
print 'ppw =',ppw
print 'tempomax =',tempomax
```

```
print 'tipo_fonte =',tipo_fonte
print 'HARD_SOFT =',HARD_SOFT
print 'ATIVANDO_DIPOLO_1 =',ATIVANDO_DIPOLO_1
print 'ATIVANDO_DIPOLO_2 =',ATIVANDO_DIPOLO_2
print 'rdsdz =', rdsdz
print 'cdtds =', cdtds
print 'fator comprimento dipolo  =',fator_comp
print 'fator atenuação (senóide) =',fator_at
print 'fator geometria (1, 2, 3) =',fator_geometria
print 'arqsaida1 =',arqsaida1
print 'arqsaida2 =',arqsaida2
# FINAL DA LEITURA

CFL = int(cdtds*100)/100.0 # número de Courant para gráfico
compdipolo = int(fator_comp*100)/100.0 # para gráfico

#****************************************

M = Nx # código adicional invertido para funcionar
N = Ny
P = Nz

px = M/2 + 1 # posição da fonte ricker
py = N/2 + 1
pz = P/2 + 1
NLOSS = 20 # numero de pontos (a mais)
        # na grade 1D usada para o campo total
Mx = M + NLOSS # numero de pontos para a grade 1D auxilar

# matrizes para os gráficos de Ez nos planos xy e xz
ez_pxy = np.zeros((Nx,Ny))
ez_pxz = np.zeros((Nx,Nz-1))

ez_pxy2 = np.zeros((Ny,Nx)) # transposta de ez_pxy
ez_pxz2 = np.zeros((Nz-1,Nx)) # transposta de ez_pxz

#************************************

# ENTRADA DE DADOS

#************************************************
tipo_1d = 0 # grade 1D auxiliar
tipo_3d = 3 # grade 3D
imp0 = 377.0 # impedância característica no vácuo

max_loss = 0.35 # fator de perda para grade 1D

firstx = 5 # define região de campo total
firsty = 5
lastx = 95
lasty = 75

dx = 1.0 # tamanho da célula na direção x
dy = dx  # tamanho da célula na direção y
dz = dx
```

```
Lx = (M-1)*dx # comprimento na direção x
Ly = (N-1)*dy # comprimento na direção y
Lz = (P-1)*dz

# textos para gráficos
texto2 =  'dim='+str(Nx)+'x'+str(Ny)+'x'+str(Nz-1)+\
          '  ppw='+str(ppw)+'  CFL='+str(CFL) +\
          '  dx=dy=dz='+str(dx)

if tipo_fonte == 0:
   texto1 = u'p_ricker '
else:
   texto1 = u'senóide '
if HARD_SOFT == 0:
   texto1 = texto1 + 'hard '
else:
   texto1 = texto1 + 'soft '
if ATIVANDO_DIPOLO_1 == 1 :
   texto3 = '  dipolo '+str(compdipolo)
else:
   texto3 = ''
textof = texto1 + texto2 + texto3

# testando a fonte sozinha no tempo
ez_teste = np.zeros((tempomax))
tempo_teste = np.zeros((tempomax))

#******************************
#******************************

# INICIO DOS CALCULOS PARA GRADE 3D

print u'\nINICIO FDTD YEE 3D'
print
tempoestimado = tempomin_estimado*(6.0*tempomax*Nx*Ny*Nz)
delta_min2 = int(tempoestimado)/60
delta_seg2 = int(tempoestimado)%60
print'Tempo estimado =', delta_min2,'min e',delta_seg2,'seg'
print


#inicializando_grade_3d()
#inicializando_grade_1d()

if ATIVANDO_DIPOLO_1 == 1:
   dipoloz(px,py,pz,comprimento_dipolo,ativo)

if ATIVANDO_DIPOLO_2 == 1:
   dipoloz(px+ppw/4,py,pz,ppw/2+2,0)


# inicio loop do tempo

for tempo in range(0,tempomax):
   #atualiza_h3d(tipo_3d)
```

```
    #atualiza_tfsf(tempo)
    #atualiza_e3d(tipo_3d)
    #ez[px,py,pz] = HARD_SOFT*ez[px,py,pz] + fonte_1(tempo,0.0,tipo_fonte,fator_at)
    tempo_teste[tempo] = tempo
    ez_teste[tempo] = fonte_1(tempo,0.0,tipo_fonte,fator_at)
    #abc3d_ordem1()
    #print '\n',tempo + 1,'/',tempomax

# final loop do tempo

#*************************************
# LENDO ARQUIVOS DE ENTRADA

# LEITURA DAS MATRIZES PARA PLANOS XY E XZ
arquivo4 = open(arqsaida1,"r")
#linha = None
for i in range(0,Nx):
     for j in range(0,Ny):
        linha = arquivo4.readline() #funcionou
        ez_pxy[i,j] = float(linha)
arquivo4.close()

arquivo5 = open(arqsaida2,"r")
#linha = None
for i in range(0,Nx):
     for k in range(0,Nz-1):
        linha = arquivo5.readline() #funcionou
        ez_pxz[i,k] = float(linha)
arquivo5.close()

#*******************************************
#************************************************

# gerando gráficos 1D a partir dos gráficos 2D
Nx2 = Nx - px -2
Ny2 = Ny - py -2
Nz2 = Nz - pz -3
ez_x1 = np.zeros((Nx2+1))
ez_y1 = np.zeros((Ny2+1))
ez_x2 = np.zeros((Nx2+1))
ez_z2 = np.zeros((Nz2+1))
x = np.zeros((Nx2+1))
y = np.zeros((Ny2+1))
z = np.zeros((Nz2+1))

for i in range(0,Nx2+1):
    x[i] = i*dx
    ez_x1[i] = ez_pxy[px+i,py] # plano xy
    ez_x2[i] = ez_pxz[px+i,pz] # plano xz
for j in range(0,Ny2+1):
    y[j] = j*dy
    ez_y1[j] = ez_pxy[px,py+j] # plano xy
for k in range(0,Nz2+1):
    z[k] = k*dz
    ez_z2[k] = ez_pxz[px,pz+k] # plano xz
```

```
if Nx < Ny: # diagonal do plano xy
   Nd = Nx2
else:
   Nd = Ny2
ez_d1 = np.zeros((Nd+1))
s = np.zeros((Nd+1))
ds = np.sqrt(dx**2 +dy**2)
for k in range(0,Nd+1):
   s[k] = k*ds
   ez_d1[k] = ez_pxy[px+k,py+k]

if Nx < Nz: # diagonal do plano xz
   Nd2 = Nx2
else:
   Nd2 = Nz2
ez_d2 = np.zeros((Nd2+1))
s2 = np.zeros((Nd2+1))
ds2 = np.sqrt(dx**2 + dz**2)
for k in range(0,Nd2+1):
   s2[k] = k*ds2
   ez_d2[k] = ez_pxz[px+k,pz+k]
#*****************************************
# GRAVANDO GRAFICOS 1D EM 3 SAIDAS
dados = [Nx2,Nz2,Nd2,dx]

arquivow = open("saida_dados_y.txt","w")
for i in range(0,4):
   arquivow.write("%f\n" % dados[i])  # dados plano xz
arquivow.close()

arquivox = open("saida_ez_x2y.txt","w")
for i in range(0,Nx2+1):
   arquivox.write("%12.11f\n" % ez_x2[i])  # plano xz
arquivox.close()

arquivoz = open("saida_ez_z2y.txt","w")
for k in range(0,Nz2+1):
   arquivoz.write("%12.11f\n" % ez_z2[k])  # plano xz
arquivoz.close()

arquivod = open("saida_ez_d2y.txt","w")
for k in range(0,Nd2+1):
   arquivod.write("%12.11f\n" % ez_d2[k])  # plano xz
arquivod.close()

#*****************************************
# desenhando dipolo no plano xz
vminimo = -abs(ez_pxz).max()
if ATIVANDO_DIPOLO_1 == 1:
   dipoloz_grafico(px,py,pz,comprimento_dipolo,ativo,vminimo)

if ATIVANDO_DIPOLO_2 == 1:
   dipoloz_grafico(px+ppw/4,py,pz,ppw/2+2,0,vminimo)
```

```
ez_pxy2 = ez_pxy.T # transposta para corrigir imagem
ez_pxz2 = ez_pxz.T # transposta para corrigir imagem
#****************************************

# medindo o tempo da simulação
tempo2 = time.time()
delta_t1 = tempo2 - tempo1
delta_min = int(delta_t1)/60
delta_seg = int(delta_t1)%60
print'Tempo =', delta_t1,'segundos'
print'Tempo =', delta_min,'min e',delta_seg,'seg'
tempomin = delta_t1/(8.0*tempomax*Nx*Ny*Nz)
print 'Tempo_min =',tempomin,'seg'
print
print'Tempo estimado =', delta_min2,'min e',delta_seg2,'seg'

if ATIVA_POLAR == 1:
    print 'px = ',px
    print 'nxp = ',nxp
    print 'n = ',n
    print 'raio = ',raio
    print'\n\n'
print u'FINAL'
print
print

#****************************************

# PLOTANDO CAMPOS Ez PARA O MODO 3D
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
fs1 = 16
fs2 = 17
fs3 = 17
fse = 13
lw = 1.6

if MOSTRAR_SOMENTE_GRAFICOS_1D == 0:
    # nearest ou bilinear - funcionou
    # cm.jet ou cm.RdYlGn ou cm.winter ou cm.hot
    # ficou imagem correta com shape=None e com extent=[0,Lx,0,Ly]
    # mas shape inverteu eixos x e y
    # no entanto usando a transposta de ez a imagem ficou com eixos corretos

    plt.figure()
    im = plt.imshow(ez_pxy2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Ly], shape=None,
                vmax= nivel*abs(ez_pxy2).max(), vmin= -nivel*abs(ez_pxy2).max()  )

    plt.colorbar(im) # funcionou
    plt.title(u'CAMPO YEE 3D: Ez NO PLANO XY (TEMPO = '+str(tempomax)+' passos)')
    plt.xlabel(u'Eixo X '+ textof)
    plt.ylabel(u'Eixo Y')
    #plt.show()
```

```
    plt.figure()
    im2 = plt.imshow(ez_pxz2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Lz], shape=None,
                vmax= nivel*abs(ez_pxz2).max(), vmin= -nivel*abs(ez_pxz2).max()  )

    plt.colorbar(im2) # funcionou
    plt.title(u'CAMPO YEE 3D: Ez NO PLANO XZ (TEMPO = '+str(tempomax)+' passos)')
    plt.xlabel(u'Eixo X '+ textof)
    plt.ylabel(u'Eixo Z')
    #plt.show()

#*******************************************
# GRÁFICOS 1D
# no plano xy
plt.figure() # figura yee 3d usada no texto final
plt.tick_params(labelsize = fse)
plt.plot(x,ez_x1,'r',linewidth=lw)
plt.axis([0.,130.,-.0009, .0013])
plt.grid()
plt.xlabel(u'Eixos x/y/d (metros)',fontsize=fs2)
plt.ylabel(u'Amplitude ( V/m )',fontsize=fs2)
plt.title(u'YEE 3D: CURVAS 1D DE CAMPOS Ez NO PLANO XY (TEMPO = '+str(tempomax)+'
passos)'
          ,fontsize=fs2)
plt.hold(True) # permite sobrepor vários
             # gráficos com dimensões diferentes
plt.plot(y,ez_y1,'b',linewidth=lw)
plt.plot(s,ez_d1,'k',linewidth=lw)
texto1 = u'ez_x1_( 0°)'
texto2 = u'ez_y1_(90°)'
texto3 = u'ez_d1_(45°)'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)

#plt.show()

# no plano xz
plt.figure()
plt.plot(x,ez_x2,'r')
plt.grid()
plt.xlabel(u'Eixos x/z/d, '+ textof)
plt.ylabel(u'Nível')
plt.title('Campos Yee 1D: Ez plano xz (TEMPO = '+str(tempomax)+' passos)')
plt.hold(True) # permite sobrepor vários
             # gráficos com dimensões diferentes

plt.plot(z,ez_z2,'b')
plt.plot(s2,ez_d2,'k')
texto1 = u'ez_x2'
texto2 = u'ez_z2'
texto3 = u'ez_d2'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)
#plt.show()
```

```
# fonte em função do tempo
plt.figure()
plt.plot(tempo_teste,ez_teste,'k')
plt.grid()
plt.xlabel(u'Eixo Tempo, '+ textof)
plt.ylabel(u'Nível')
plt.title('Fonte para campo Ez (TEMPO = '+str(tempomax)+' passos)')
#plt.show()

plt.show()# mostra todos os gráficos

# FINAL DO PROGRAMA
#############################################################
```

ARQUIVO 6: VISUALIZAÇÃO CONJUNTA DE ONDAS SENOIDAS
DE AMBOS OS MÉTODO FDTD

```
# -*- coding: cp1252 -*-
#            cp1252 ou utf-8
# programa programa_fdtd_3d_h_2c_visualizacao_1c.py em python 2.7.3
#
# AUTOR: MARINOEL JOAQUIM

#*********************************************************
# BIBLIOTECAS NECESSÁRIAS

import numpy as np
import time

#*********************************************************
tempo1 = time.time()

# INICIO DAS FUNÇÕES

def func_senoide(A2,r,fa,fb,sc,Nc,Nt,d,alfa,teta):
    c = fa*3.0e8
    dt = sc*d/c
    t = Nt*dt*fb
    fc = 1.0 # fc = -c para derivda no tempo
    co = Nc*d
    f = c/ co
    ct = alfa/f # com fator de atenuação: alfa
    k = 2.0*np.pi/co
    #fs2 = (A2*k/r )*np.cos(2.0*pi*f*t - r*k)
    fs2 = ( -A2/(c*ct*r) )*np.exp((-t + r/c)/ct)*\
       np.sin(2.0*np.pi*f*t - r*k + teta) + ( -A2*k/r )*( 1.0 -\
       np.exp((-t + r/c)/ct) )*np.cos(2.0*np.pi*f*t - r*k + teta)
    return fs2

def func_senoide2(A2,r,fa,fb,sc,Nc,Nt,d,alfa,teta):
    c = fa*3.0e8
    dt = sc*d/c
    t = Nt*dt*fb
    alfa2 = alfa
```

```
    fc = 1.0 # fc = -c para derivda no tempo
    co = Nc*d
    f = c/ co
    k = 2.0*np.pi/co
    fs2 = (-A2*k/r )*np.cos(2.0*np.pi*f*t - r*k + teta)
    return fs2

#*********************************************************

# INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS
Nx = 81 # numero de pontos na direção x
Ny = 141  # numero de pontos na direção y
Nz = 71 # numero de pontos na direção z
ppw = 15 # numero de pontos por comprimento de onda
tempomax = 40

tipo_fonte = 1  # 0=pulso ricker ou 1=senoide
HARD_SOFT  = 1   # fonte: 0=hard ou 1=soft
ATIVANDO_DIPOLO_1 = 0 # 1=ativa 0=desativa
ATIVANDO_DIPOLO_2 = 0 # 1=ativa 0=desativa

rdsdz = 1.0 #np.sqrt(1.0) # razão ds/dz
fator_comp = 1.4 # em comprimentos de onda para dipolo_1

arqsaida1 = "ez_pxy_1.txt"
arqsaida2 = "ez_pxz_2.txt"
fator_at = 0.5

# FINAL DAS PRINCIPAIS VARIÁVEIS
#########################################################################
#########################################################################
#************************************************

# VARIÁVEIS DE CONTROLE DOS GRÁFICOS

USAR_ESTES_ARQUIVOS = 2 # usar 1 ou 2, ou 0

MOSTRAR_SOMENTE_GRAFICOS_1D = 1 # 1= graficos 1D, 0 = não mostra
MOSTRAR_SOMENTE_GRAFICOS_2D = 0 # 1= graficos 2D, 0= não mostra

# ALTERANDO ESPESSURA DA LINHA E TAMANHO DA FONTE DE LETRAS
lw = 1.6
fs1 = 15
fs2 = 17
fs3 = 17

#*****************************************************
#########################################################################
## PARA METODO FDTD HEX
# LEITURA DE ARQUIVO DE ENTRADA (1° FORMA)
arquivo3 = open("saida_dados_h.txt","r")
Nx2_txt = list(arquivo3.readline()) #funcionou
Nx2_txt2 = Nx2_txt.pop(-1)
Nx2 = "".join(Nx2_txt)
Nx2 = int(float(Nx2))
```

```
Nz2_txt = list(arquivo3.readline()) #funcionou
Nz2_txt2 = Nz2_txt.pop(-1)
Nz2 = "".join(Nz2_txt)
Nz2 = int(float(Nz2))

Nd2_txt = list(arquivo3.readline()) #funcionou
Nd2_txt2 = Nd2_txt.pop(-1)
Nd2 = "".join(Nd2_txt)
Nd2 = int(float(Nd2))

ds_txt = list(arquivo3.readline()) #funcionou
ds_txt2 = ds_txt.pop(-1)
ds = "".join(ds_txt)
ds = float(ds)

rds_txt = list(arquivo3.readline()) #funcionou
rds_txt2 = rds_txt.pop(-1)
rdsdz = "".join(rds_txt)
rdsdz = float(rdsdz)
arquivo3.close()
#*********************************************************

# LEITURA DE ARQUIVO DE ENTRADA (2° FORMA) - MAIS SIMPLES
dados_h = np.zeros((5))
arquivo3 = open("saida_dados_h.txt","r")
for i in range(0,5):
    linha = arquivo3.readline()
    dados_h[i] = float(linha)
arquivo3.close()
print dados_h

#************************************************
# GRÁFICOS 1D PARA HEX
ez_x2 = np.zeros((Nx2+1))
ez_z2 = np.zeros((Nz2+1))
ez_d2 = np.zeros((Nd2+1))

x = np.zeros((Nx2+1))
z = np.zeros((Nz2+1))
s2 = np.zeros((Nd2+1))

dx = ds*np.sqrt(3.0)/2.0
dz = ds
ds2 = np.sqrt(dx**2 + dz**2)

arquivox = open("saida_ez_x2h.txt","r")
for i in range(0,Nx2+1):
    linha = arquivox.readline()
    x[i] = i*dx
    ez_x2[i] = float(linha)
arquivox.close()

arquivoz = open("saida_ez_z2h.txt","r")
for i in range(0,Nz2+1):
```

```
    linha = arquivoz.readline()
    z[i] = i*dz
    ez_z2[i] = float(linha)
arquivoz.close()

arquivod = open("saida_ez_d2h.txt","r")
for i in range(0,Nd2+1):
    linha = arquivod.readline()
    s2[i] = i*ds2
    ez_d2[i] = float(linha)
arquivod.close()
#************************************************
## PARA MÉTODO FDTD YEE
dados_y = np.zeros((4))

arquivow = open("saida_dados_y.txt","r")
for i in range(0,4):
    linha = arquivow.readline()
    dados_y[i] = float(linha)
arquivow.close()
Nx2y = int(dados_y[0])
Nz2y = int(dados_y[1])
Nd2y = int(dados_y[2])
dxy = dados_y[3]
print dados_y

# GRÁFICOS 1D PARA YEE
ez_x2y = np.zeros((Nx2y+1))
ez_z2y = np.zeros((Nz2y+1))
ez_d2y = np.zeros((Nd2y+1))

xy = np.zeros((Nx2y+1))
zy = np.zeros((Nz2y+1))
s2y = np.zeros((Nd2y+1))

dzy = dxy
ds2y = np.sqrt(dxy**2 + dzy**2)

# LENDO ARQUIVOS 1D PARA YEE
arquivox = open("saida_ez_x2y.txt","r")
for i in range(0,Nx2y+1):
    linha = arquivox.readline()
    xy[i] = i*dxy
    ez_x2y[i] = float(linha)
arquivox.close()

arquivoz = open("saida_ez_z2y.txt","r")
for i in range(0,Nz2y+1):
    linha = arquivoz.readline()
    zy[i] = i*dzy
    ez_z2y[i] = float(linha)
arquivoz.close()

arquivod = open("saida_ez_d2y.txt","r")
for i in range(0,Nd2y+1):
```

```
    linha = arquivod.readline()
    s2y[i] = i*ds2y
    ez_d2y[i] = float(linha)
arquivod.close()

#*************************************************
# ENTRADA DE VARIÁVEIS PARA FUNÇAO ANALITICA - SENOIDE
N = 600 # para definir resolução espacial

fr = np.zeros((N))
fr2 = np.zeros((N))
fr3 = np.zeros((N))
fsa = np.zeros((N))
fsa2 = np.zeros((N))
r = np.zeros((N))
pi = np.pi

Ar = 0.023
Ar2 = 0.15
Ar3 = 0.012
Nt = 200    # passos no tempo
fa = 1.    # não altera forma de onda
fb = 1.0     # fb = 0.8 para pulso
          # fb = 1.0 para senoide
A = -0.075     # amplitude da senoide na direção x
teta1 = 80*np.pi/180 # fase inicial

A2 = -0.0196    # amplitude da senoide na direção z
fb2 = 1.0
teta2 = 160*np.pi/180 # fase inicial

alfa = 0.5    # fator de atenuação da cossenoide
L = 350.0
c = fa*3.0e8
sc = 1.0/np.sqrt(3.0) # numero de Courant
Nc = 20.0   # numero de ponto por comprimento de onda
d = 1.0     # tamanho do lado do hexagono grande
dt = sc*d/c
t = Nt*dt*fb
fc = 1.0        # fc = -c para derivda no tempo
co = Nc*d
f = c/ co
ro = t*c
cdr = 1.1 #/np.sqrt(3.0)
dr = 130./N #  1.0 para senoide ou >1.0 para pulso
# 118.0 é um valor empirico; maiores valores a senoide analitica
# aumenta a sua amplitude e é um pouco maior que ro=115.47
print 'ro =', ro

for i in range(0,N):
    r[i] = i*dr
    r2 = i*dr + 1e-25
    fsa[i] = func_senoide(A,r2,fa,fb,sc,Nc,Nt,d,alfa,teta1)
    fsa2[i] = func_senoide(A2,r2,fa,fb2,sc,Nc,Nt,d,alfa,teta2)
    fr[i] = Ar/(r2)
```

```
    fr2[i] = Ar2/(r2*r2)
    fr3[i] = Ar3/(r2)

#########################################################################

#########################################################################

# PLOTANDO CAMPOS Ez PARA O MODO 3D
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt


#*****************************************
fs1 = 16
fs2 = 17
fs3 = 17
fse = 13
lw = 1.6

if MOSTRAR_SOMENTE_GRAFICOS_1D == 1:
    # GRÁFICOS 1D HEX
    # no plano xz
    plt.figure()
    plt.plot(x,ez_x2,'r')
    plt.grid()
    plt.xlabel(u'Eixos x/z/d')
    plt.ylabel(u'Nível')
    plt.title('Campos HEX 1D: Ez plano xz (TEMPO = 200 passos)')
    plt.hold(True) # permite sobrepor vários
                   # gráficos com dimensões diferentes

    plt.plot(z,ez_z2,'b-')
    plt.plot(s2,ez_d2,'k')
    texto1 = u'ez_x2'
    texto2 = u'ez_z2'
    texto3 = u'ez_d2'
    plt.legend((texto1,texto2,texto3),loc = 'upper right')
    plt.hold(False)
    #plt.show()

    # GRÁFICOS 1D YEE
    # no plano xz
    plt.figure()
    plt.plot(xy,ez_x2y,'r')
    plt.grid()
    plt.xlabel(u'Eixos x/z/d')
    plt.ylabel(u'Nível')
    plt.title(u'Campos YEE 1D: Ez plano xz (TEMPO = 200 passos)')
    plt.hold(True) # permite sobrepor vários
                   # gráficos com dimensões diferentes

    plt.plot(zy,ez_z2y,'b-')
    plt.plot(s2y,ez_d2y,'k')
    texto1 = u'ez_x2'
```

```
    texto2 = u'ez_z2'
    texto3 = u'ez_d2'
    plt.legend((texto1,texto2,texto3),loc = 'upper right')
    plt.hold(False)
    #plt.show()
##################################################################
    # COMPARANDO GRÁFICOS 1D HEX E YEE
    plt.figure()
    plt.tick_params(labelsize = fse)
    plt.plot(x,ez_x2,'k',linewidth=lw)
    plt.axis([0.,130.,-.0010, .0014])
    plt.grid()
    plt.xlabel(u'Eixos x ( metros )',fontsize=fs2)
    plt.ylabel(u'Amplitude ( V/m )',fontsize=fs2)
    plt.title(u'COMPARANDO CAMPOS Ez: HEX E YEE NA DIREÇÃO X',fontsize=fs2)
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
    plt.plot(xy,ez_x2y,'r-',linewidth=lw)
    plt.plot(r,fr,'k--',linewidth=lw)
    texto1 = u'eixo_x_(0°)_HEX'
    texto2 = u'eixo_x_(0°)_YEE'
    texto3 = r'$1/r$'
    plt.legend((texto1,texto2, texto3),loc = 'upper right')
    plt.hold(False)
    #plt.show()

    plt.figure()
    plt.tick_params(labelsize = fse)
    plt.plot(z,ez_z2,'k',linewidth=lw)
    plt.axis([0.,130.,-.00093, .0003])
    plt.grid()
    plt.xlabel(u'Eixos z ( metros )',fontsize=fs2)
    plt.ylabel(u'Amplitude ( V/m )',fontsize=fs2)
    plt.title(u'COMPARANDO CAMPOS Ez: HEX E YEE NA DIREÇÃO Z',fontsize=fs2)
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
    plt.plot(zy,ez_z2y,'r-',linewidth=lw)
    plt.plot(r,fr2,'k--',linewidth=lw)
    texto1 = u'eixo_z_(90°)_HEX'
    texto2 = u'eixo_z_(90°)_YEE'
    texto3 = r'$1/r^2$'
    plt.legend((texto1,texto2,texto3),loc = 'lower right')
    plt.hold(False)
    #plt.show()

    plt.figure()
    plt.tick_params(labelsize = fse)
    plt.plot(s2,ez_d2,'k',linewidth=lw)
    plt.axis([0.,130.,-.0013, .0007])
    plt.grid()
    plt.xlabel(u'Eixos d ( metros )',fontsize=fs2)
    plt.ylabel(u'Amplitude ( V/m )',fontsize=fs2)
    plt.title(u'COMPARANDO CAMPOS Ez: HEX E YEE NA DIREÇÃO D',fontsize=fs2)
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
```

```
    plt.plot(s2y,ez_d2y,'r-',linewidth=lw)
    plt.plot(r,fr3,'k--',linewidth=lw)
    texto1 = u'eixo_d_(49°)_HEX'
    texto2 = u'eixo_d_(45°)_YEE'
    texto3 = r'$1/r$'
    plt.legend((texto1,texto2,texto3),loc = 'upper right')
    plt.hold(False)
    plt.show()

# FINAL DO PROGRAMA
### FINAL DO APÊNDICE 4
```

APÊNDICE 5. PROGRAMAS PARA GERAR GRÁFICOS DE VELOCIDADE NORMALIZADA, ANISOTROPIA E DISPERSÃO PARA AMBOS OS MÉTODOS FDTD

```
# -*- coding: cp1252 -*-

# programa: programa_fdtd_determinante_2a_com_grafico_1a.py
# AUTOR: MARINOEL JOAQUIM
# VERSÃO USADA PARA PLOTAR CURVAS DE ANISOTROPIA: HEX3D, YEE3D
# COM QUALQUER NUMERO DE ANGULOS
# 08/09/2014
# TROCADO SINAIS ERRADOS, MAS SEM ALTERAR RESULTADOS EM 16/09/2014
# ULTIMA ATUALIZAÇÃO EM 14/09/2015
#************************************************************
# BILIOTECAS NECESSÁRIAS AO PROGRAMA
import numpy as np
#************************************************************

FORMA_EXPOENTE = 0 # 1 = EXP OU 0 = COS + J*SIN
TIPO_COEF = 3 # 1 ou 2 ou 3 (correto)

TIPO = 0 # 0 = 3D HEX OU  1 = 2D HEX OU 2 = 3D YEE
TIPOG = TIPO  # PARA GRÁFICOS
# PARA TIPO = 0 (3D)
TIPO_K = 2  # 1 (qqr plano), 2 (plano xy), 3 (plano xz), 4 (plano yz)
        # 5 (plano perpendicular a alfa no plano xy)
MOSTRA_ANISOTROPIA = 0 # 0= não mostra ou 1= mostra
#************************************************************

# INÍCIO DOS CÁLCULOS
Nf = 360
alfag = np.zeros((Nf))
nivel = np.zeros((Nf))
nivel0 = np.zeros((Nf))
nivel1 = np.zeros((Nf))
nivel2 = np.zeros((Nf))
nively = np.zeros((Nf)) # para o método YEE 3D
cfly = 1.0/np.sqrt(3.0)
N = 10  # NUMERO DE PONTOS POR COMPRIMENTO DE ONDA

fbx = 0.8781 # Para tipo _coef = 1
fbx2 = 0.935 # Para tipo _coef = 2

# Para tipo _coef = 3 (correto)
fax3 = 1.0  # funcionou com 0.999 para N=15
fbx3 = 1.0  # funcionou com 1.001 para N=15


rdz  = 1.1547 # 1.06 anisotropia próxima ao yee
rdz_hex3d = rdz
rdz_yee3d = 1.0
fator_cfl_2d = 2.0
fator_cfl = fator_cfl_2d + rdz**2
```

```
cfl = 1.0/np.sqrt(fator_cfl)

ds = 1.0
zo = 120*np.pi
k = 2*np.pi/(N*ds)
dalfa = 2*np.pi/Nf
tipo_s = [0,2]

for nn in range(0,2):
    TIPO = tipo_s[nn]
    if TIPO == 0:
        #fator_cfl = 2.0 + rdz**2.0 #aproximado
        fator_cfl = (np.sqrt(9.*rdz**4 + 28.*rdz**2 + 36.)+ 3.*rdz**2 + 6.)/6.
        fator_cfl_hex3d = fator_cfl
    if TIPO == 2:
        rdz = 1.0
        fator_cfl = 2.0 + rdz**2.0
    cfl = 1.0/np.sqrt(fator_cfl)


    for i in range(0,Nf):
        alfag[i] = i*dalfa*180/np.pi
        alfa = i*dalfa
        if (TIPO == 0 or TIPO == 2):  # 3D: HEX (0) OU YEE (2)
            if TIPO_K == 1: # k no espaço 3D
                teta = (np.pi/180)*45
                #teta = np.arctan(1.0/np.cos(np.pi/6))
                kx = k*np.cos(alfa)*np.sin(teta)
                ky = k*np.sin(alfa)*np.sin(teta)
                kz = k*np.cos(teta)
                texto1 = u'PLANO COM ÂNGULO TETA = '+str(round((teta*180/np.pi),2))
            if TIPO_K == 2: # k no plano xy
                teta = np.pi/2 # não usado
                #alfa = 1*np.pi/3
                kx = k*np.cos(alfa)
                ky = k*np.sin(alfa)
                kz = 0.0
                texto1 = 'PLANO XY'
            if TIPO_K == 3: # k no plano xz
                teta = np.pi/2 # não usado
                #alfa = 1*np.pi/2
                #alfa = np.arctan(1.0/np.cos(np.pi/6))
                kx = k*np.cos(alfa)
                ky = 0.0
                kz = k*np.sin(alfa)
                texto1 = 'PLANO XZ'
            if TIPO_K == 4: # k no plano yz
                teta = np.pi/2 # não usado
                #alfa = 1*np.pi/3
                ky = k*np.cos(alfa)
                kz = k*np.sin(alfa)
                kx = 0.0
                texto1 = 'PLANO YZ'
            if TIPO_K == 5: # k no espaço 3D, para um certo alfa
                teta = (np.pi/180)*45 # trocando teta com alfa por simplicidade
```

```
        kx = k*np.cos(teta)*np.sin(alfa)
        ky = k*np.sin(teta)*np.sin(alfa)
        kz = k*np.cos(alfa)
        texto1 = u'PLANO COM ÂNGULO ALFA = '+\
          str(round((teta*180/np.pi),2))+ ' (PLANO XY)'


if TIPO == 1: # HEX 2D
    teta = np.pi/2 # não usado
    #alfa = 1*np.pi/3
    kx = k*np.cos(alfa)
    ky = k*np.sin(alfa)
    kz = 0.0
    texto1 = 'PLANO XY (2D)'
#*******************************************


if TIPO == 0: # HEX 3D
    if TIPO_COEF == 1:
        fa = 3.0*np.sqrt(3.0)/4
        fb = fbx*3.0/4
        fc = 2.0/3
    if TIPO_COEF == 2:
        fa = 2.0*np.sqrt(3.0)/3
        fb = fbx2*2.0/3
        fc = 2.0/3
    if TIPO_COEF == 3:
        fa = fax3*1.0
        fb = fbx3*1.0/np.sqrt(3.0)
        fc = 2.0/3
if TIPO == 1: # HEX 2D
    fa = 1.0
    fb = 1.0 #não usado
    fc = 2.0/3
if TIPO == 2: # YEE 3D
    fa = 1.0
    fb = 1.0 #não usado
    fc = 1.0



ha = fa*cfl/zo
ea = fa*cfl*zo
hb = fb*rdz*cfl/zo
eb = fb*rdz*cfl*zo
hz = fc*cfl/zo
ez = fc*cfl*zo


asc1 = (3.0**(1.0/2))*kx*ds/12.0 - ky*ds/4.0
asc2 = (3.0**(1.0/2))*kx*ds/12.0 + ky*ds/4.0
asc3 = (3.0**(1.0/2))*kx*ds/6.0
s1 = np.sin(ky*ds/2.0)
s2 = np.sin((3.0**(1.0/2))*kx*ds/4.0 - ky*ds/4.0)
s3 = np.sin((3.0**(1.0/2))*kx*ds/4.0 + ky*ds/4.0)
```

```
sz = np.sin(kz*ds/(2.0*rdz))

j2 = 1.0j
sc1 = (np.cos(asc1) - np.sin(asc1)*j2)
sd1 = (np.cos(asc1) + np.sin(asc1)*j2)
sc2 = (np.cos(asc2) - np.sin(asc2)*j2)
sd2 = (np.cos(asc2) + np.sin(asc2)*j2)
sc3 = (np.cos(asc3) + np.sin(asc3)*j2)
sd3 = (np.cos(asc3) - np.sin(asc3)*j2)

#sx2 = np.sin(kx*ds/2.0) # método Yee
#sy2 = np.sin(ky*ds/2.0)
#sz2 = np.sin(kz*ds/2.0)


if  FORMA_EXPOENTE == 1:
    sc1 = np.exp(-j2*asc1)
    sd1 = np.exp(j2*asc1)
    sc2 = np.exp(-j2*asc2)
    sd2 = np.exp(j2*asc2)
    sc3 = np.exp(j2*asc3)
    sd3 = np.exp(-j2*asc3)


if TIPO == 0: # HEX 3D
    a6 = -hb*sz*sc1
    a7 = -hb*sz*sc2
    a8 = ha*s1
    b5 = hb*sz*sc1
    b7 = -hb*sz*sc3
    b8 = -ha*s2
    c5 = hb*sz*sc2
    c6 = hb*sz*sc3
    c8 = -ha*s3
    d5 = -hz*s1
    d6 = hz*s2
    d7 = hz*s3
    e2 = eb*sz*sd1
    e3 = eb*sz*sd2
    e4 = -ea*s1
    f1 = -eb*sz*sd1
    f3 = eb*sz*sd3
    f4 = ea*s2
    g1 = -eb*sz*sd2
    g2 = -eb*sz*sd3
    g4 = ea*s3
    h1 = ez*s1
    h2 = -ez*s2
    h3 = -ez*s3
    a = 0.0
    x2 = np.matrix([[a,0,0,0,0,a6,a7,a8],
                    [0,a,0,0,b5,0,b7,b8],
                    [0,0,a,0,c5,c6,0,c8],
                    [0,0,0,a,d5,d6,d7,0],
                    [0,e2,e3,e4,a,0,0,0],
```

```
                [f1,0,f3,f4,0,a,0,0],
                [g1,g2,0,g4,0,0,a,0],
                [h1,h2,h3,0,0,0,0,a]])
    x3 = np.linalg.eigvals(x2)
    x4 = (N/(np.pi*cfl))*np.arcsin(x3)
    x5 = x4.max()
    nivel0[i] = np.real(x5)
    #nivel[i] = np.imag(x5)

if TIPO == 2: # método YEE 3D
    dx = ds
    dy = ds
    dz = ds
    sx2 = np.sin(kx*dx/2)
    sy2 = np.sin(ky*dy/2)
    sz2 = np.sin(kz*dz/2)
    ha = 1.0
    hb = cfl/zo
    ea = 1.0
    eb = cfl*zo
    a5 = -hb*sz2
    a6 = hb*sy2
    b4 = hb*sz2
    b6 = -hb*sx2
    c4 = -hb*sy2
    c5 = hb*sx2
    d2 = eb*sz2
    d3 = -eb*sy2
    e1 = -eb*sz2
    e3 = eb*sx2
    f1 = eb*sy2
    f2 = -eb*sx2
    a = 0.0
    x2 = np.matrix([[a,0,0,0,a5,a6],
                [0,a,0,b4,0,b6],
                [0,0,a,c4,c5,0],
                [0,d2,d3,a,0,0],
                [e1,0,e3,0,a,0],
                [f1,f2,0,0,0,a]])
    x3 = np.linalg.eigvals(x2)
    x4 = (N/(np.pi*cfl))*np.arcsin(x3)
    x5 = x4.max()
    nivel2[i] = np.real(x5)
    #nivel2[i] = np.imag(x5)

if TIPO == 1: # HEX 2D
    a8 = ha*s1
    b8 = -ha*s2
    c8 = -ha*s3
    h1 = ez*s1
    h2 = -ez*s2
    h3 = -ez*s3
    a = 0.0
    x2 = np.matrix([[a,0,0,a8],
                [0,a,0,b8],
```

```
                    [0,0,a,c8],
                    [h1,h2,h3,a]])
        x3 = np.linalg.eigvals(x2)
        x4 = (N/(np.pi*cfl))*np.arcsin(x3)
        x5 = x4.max()
        nivel1[i] = np.real(x5)
        #nivel1[i] = np.imag(x5)
    # HEX 3D
    anisotropia_max = (nivel0.max() - nivel0.min())*100/nivel0.min()
    anivelmax = nivel0.max()
    anivelmin = nivel0.min()
    dispersao = (nivel0.min() - 1.0)*100
    # YEE 3D
    anisotropia_max2 = (nivel2.max() - nivel2.min())*100/nivel2.min()
    anivelmax2 = nivel2.max()
    anivelmin2 = nivel2.min()
    dispersao2 = (nivel2.min() - 1.0)*100
    anteorica_yee = ( (np.pi/N)**2 )*100/12
    anteorica = ( (np.pi/N)**4 )*100/360
    # FINAL CALCULOS DOS AUTOVALORES
#*********************************************

# PLOTANDO CAMPOS Ez PARA O MODO 3D
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

# GRÁFICOS 1D
# no plano desejado
lw = 1.6
fs1 = 17
fs2 = 17
fse = 14
if TIPOG == 0: # HEX 3D
    plt.figure()
    plt.ticklabel_format(style = 'plain',useOffset= False)
    plt.tick_params(labelsize= fse)
    plt.plot(alfag,nivel0,'k-',linewidth=lw)
    plt.grid()
    plt.xlabel(u'Ângulo (graus)',fontsize=fs1)
    plt.ylabel(u'Velocidade normalizada',fontsize=fs1)
    plt.title(u'ANISOTROPIA (PRISMAS HEXAGONAIS) NO '+ texto1)
    #plt.show()
if TIPO == 1: # HEX 2D
    plt.figure()
    plt.plot(alfag,nivel1,'k-')
    plt.grid()
    plt.xlabel(u'Angle (degrees))')
    plt.ylabel(u'Normalized Phase Velocity')
    plt.title(u'ANISOTROPIA NO '+ texto1)
if (TIPOG == 0 or TIPO == 2): #HEX 3D E YEE 3D
    plt.figure()
    #plt.savefig('alldone',dpi=1000)
    plt.ticklabel_format(style = 'plain',useOffset= False)
    plt.tick_params(labelsize= fse)
```

```
    plt.plot(alfag,nivel0,'k-',linewidth=lw)
    plt.grid()
    plt.xlabel(u'Ângulo (graus)',fontsize=fs1)
    plt.ylabel(u'Velocidade normalizada',fontsize=fs1)
    plt.title(u'COMPARAÇÃO DE ANISOTROPIA NO '+ texto1)
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
    plt.plot(alfag,nivel2,'r-',linewidth=lw)
    #plt.plot(s,ez_d1,'k.-')
    texto1 = u'HEX 3D'
    texto2 = u'YEE 3D'
    #texto3 = u'ez_d1'
    plt.legend((texto1,texto2),loc = 'upper right')#,fontsize=fs2)
    plt.hold(False)
    plt.show()

#plt.show()
#**************************************
# IMPRESSAO DE RESULTADOS
print 'TIPO_SIMULAÇÃO      =',TIPO
print 'TIPO_PLANO (TIPO=0) =',TIPO_K
print 'TIPO_COEFICIENTE    =',TIPO_COEF
print 'FORMA_EXPOENTE =',FORMA_EXPOENTE
print 'fator_cfl =',fator_cfl
print 'cfl  =',cfl
print 'rdz  =',rdz
print 'N    =',N
print 'Zo   =',zo
print 'teta =',teta*180/np.pi
print 'alfa =',alfa*180/np.pi
print 'k    =',k
print 'kx   =',kx
print 'ky   =',ky
print 'kz   =',kz
print 'fbx  =',fbx
print 'fbx2 =',fbx2
print 'fa   =',fa
print 'fb   =',fb
print 'fc   =',fc
print 'cha  =',ha
print 'chb  =',hb
print 'chz  =',hz
print 'cea  =',ea
print 'ceb  =',eb
print 'cez  =',ez
print
print 'ANISOTROPIA NO '+ texto1 + ' PARA TIPO =',TIPO
print 'Numero de pontos por comp_onda =',N
print 'fax3 =',fax3
print 'fbx3 =',fbx3
print 'fator_cfl =',fator_cfl
print 'cfl  =',cfl
print 'FATOR_CFL_HEX3D =',fator_cfl_hex3d
print 'razão_hex3d ds/dz =',rdz_hex3d
print 'razão_yee3d dx/dz =',rdz_yee3d
```

```
print 'HEX 3D'
print 'Nivel maximo = ',anivelmax
print 'Nivel minimo = ',anivelmin
print 'Dispersão (%) = ',dispersao
print 'anisotropia_max (%) =',anisotropia_max
print 'anisotropia_hex (%) =',anteorica
print 'YEE 3D'
print 'Nivel maximo2 = ',anivelmax2
print 'Nivel minimo2 = ',anivelmin2
print 'Dispersão2 (%) = ',dispersao2
print 'anisotropia_max2 (%) =',anisotropia_max2
print 'anisotropia_yee (%) =',anteorica_yee
print

#************************************************************
#FINAL DO PROGRAMA
#### FINAL DO APÊNDICE 5
```

APÊNDICE 6. PROGRAMA PARA VISUALIZAÇÃO E MEDIDAS DE ANISOTROPIA NUMÉRICA

```
# -*- coding: cp1252 -*-
#            cp1252 ou utf-8
# programa programa_fdtd_3d_h_2c_visualizacao_1c.py em python 2.7.3
#
# AUTOR: MARINOEL JOAQUIM

# PROGRAMA FDTD 3D HEXAGONAL
#************************************************************
# BIBLIOTECAS NECESSÁRIAS

import numpy as np
import time

#************************************************************
tempo1 = time.time()

# INICIO DAS FUNÇÕES

# Função para hz e h3
def p2a(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx-1)
        p2 = (p1 + 1)/2
        if (p2 > PH3):
            p2 = 0
    return p2

# Função para ez e e3
def p2b(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx)
        p2 = (p1 + 1)/2
        if (p2 > PEZ):
            p2 = 0
    return p2

# Função para h2 e e1
def p2c(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx)
        p2 = (p1)/2
        if (p2 > PH2):
            p2 = 0
    return p2
```

```
# Função para h1 e e2
def p2d(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx-1)
        p2 = (p1 + 1)/2
        if (p2 > PH1):
            p2 = 0
    return p2
#*************************************

# FUNÇÃO PARA FONTE DO TIPO PULSO RICKER OU SENOIDE
def fonte_1(tempo2,posicao,tipo_fonte,fator_at):
    if tipo_fonte == 0: # pulso ricker
        arg = np.pi*( (cdtds*tempo2 - posicao)/ppw - 1.0 )
        arg2 = arg*arg
        funcao =(1.0 - 2.0*arg2)*np.exp(-arg2)

    else:  # senóide com amortecimento inicial
        periodo = ppw/cdtds # periodo da senóide
                        # ou tempo para o pico do pulso ricker
        cte_tempo = fator_at*periodo
        funcao = ( 1.0 - np.exp(-1.0*tempo2/cte_tempo) )*\
                np.sin(2*np.pi*tempo2*cdtds/ppw)
    return funcao
#***********************************

# função para obter i,j a partir de p para campo ez
def ez_ij(p2):
    p1 = 2*p2 - 1
    j = p1/Nx + 1
    if (p1%Nx==0 and p1>=Nx):
        j = j - 1
    i = p1 - (j-1)*Nx
    return i,j
#*********************************************************

# Função de interpolação para gráfico de ez no plano xy
def pteste(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        i = 0
        j = 0
    return i, j

# Função de interpolação para gráfico de ez no plano xz
def pteste2(i,k):
    if ((i<=0) or (i>=Nx+1) or (k<=0) or (k>=Nz+1)):
        i = 0
        k = 0
    return i, k
#*********************************************************
def medidav(ez_t2,Nt,limiar,dist,fator_ppw):
    ez_t = np.zeros((Nt+1))
    ez_t[:] = ez_t2[:]
```

```
    n_limiar = Nt # modificado
    for i in range(Nt,0,-1):
        if (ez_t[i] > limiar):
            n_limiar = i
            break
    n_limiar -= int(fator_ppw*ppw*ds/dist)
    Nlimiar = Nt - n_limiar
    eztemp = np.zeros((Nlimiar))
    for i in range(0,Nlimiar):
        eztemp[i] = ez_t[n_limiar + i]
    n_max = eztemp.argmax()
    n_max += n_limiar
    return n_max
#*********************************************

def medidav2(ez_t2, Nt, limiar, dist, fator_ppw, novo_alg):
    ez_t = np.zeros((Nt+1))
    ez_t[:] = ez_t2[:]
    n_limiar = Nt # modificado
    for i in range(Nt,0,-1):
        if (ez_t[i] > limiar):
            n_limiar = i
            break
    n_limiar -= int(fator_ppw*ppw*ds/dist)
    Nlimiar = Nt - n_limiar
    eztemp = np.zeros((Nlimiar))
    for i in range(0,Nlimiar):
        eztemp[i] = ez_t[n_limiar + i]
    n_max = eztemp.argmax()
    n_max += n_limiar
    ni = n_max # modificado
    for i in range(n_max,Nt):
        if ez_t[i] <= 0.0:
            ni = i
            break
    a = (ez_t[ni] - ez_t[ni-1])/dist
    b = ez_t[ni] - a*ni*dist
    x0 = -b/a
    # novo algoritmo em 18/06/2015
    if novo_alg == 1:
        d_ang = (2.0*np.pi/ppw)*dist/ds
        c_onda = ppw*ds
        f_alfa = ((ni-1.0)/ni)*(ez_t[ni-1]/ez_t[ni])
        B = f_alfa*np.cos(d_ang)
        C = f_alfa*np.sin(d_ang)
        teta_1 = np.arctan(C/(1.0-B))
        x0 = (c_onda/2.0)*(-teta_1)/np.pi + (ni-1.0)*dist
    return x0

# FINAL DAS FUNÇÕES
#*********************************************************
#*********************************************************

# INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS
Nx = 81 # numero de pontos na direção x
```

```
Ny = 141  # numero de pontos na direção y
Nz = 71 # numero de pontos na direção z
ppw = 15 # numero de pontos por comprimento de onda
tempomax = 40

tipo_fonte = 1  # 0=pulso ricker ou 1=senoide
HARD_SOFT  = 1   # fonte: 0=hard ou 1=soft

rdsdz = 1.0 #np.sqrt(1.0) # razão ds/dz
fator_comp = 1.4 # em comprimentos de onda para dipolo_1

arqsaida1 = "ez_pxy_1.txt"
arqsaida2 = "ez_pxz_2.txt"
fator_at = 0.5

# FINAL DAS PRINCIPAIS VARIÁVEIS
#**************************************************

# VARIÁVEIS DE CONTROLE DOS GRÁFICOS

USAR_ESTES_ARQUIVOS = 0 # usar 1 ou 2, ou 0

MOSTRAR_SOMENTE_GRAFICOS_1D = 1 # 1= graficos 1D, 0 = não mostra
MOSTRAR_SOMENTE_GRAFICOS_2D = 0 # 1= graficos 2D, 0= não mostra

# PARA EIXOS: 1= (x1 y1 d1 x2); 1B= (d1);  2= (z2);  3= (d2)
# VERSAO APERFEIÇOADA EM 22/09/2014
# NESTA VERSÃO NOVA APROXIMAÇÃO SENOIDAL NA FUNÇÃO medidav2 EM 18/06/2015
novo_alg            =   1           # 1 = USA NOVO ALGORITMO
NP                  =  17
DELTA_COMP1         =  10

NUMERO_COMPRIMENTOS1   = NP + 0.  # x1
NUMERO_COMPRIMENTOS1Y  = NP + 0. # y1
NUMERO_COMPRIMENTOS1B  = NP       # d1
NUMERO_COMPRIMENTOS_X2 = NP + 0.     # x2
NUMERO_COMPRIMENTOS2   = NP - 1.3 # z2 estava -1.3
NUMERO_COMPRIMENTOS3   = NP - 0.7 # d2

LIMIAR1 = 5e-7   # x1, d1
LIMIAR1Y = 5e-7  # y1
LIMIAR_X2 = 5e-7 # x2
LIMIAR2 = 5e-7   # z2
LIMIAR3 = 5e-7   # d2

# ALTERANDO ESPESSURA DA LINHA E TAMANHO DA FONTE DE LETRAS
lw = 1.6
fs1 = 15
fs2 = 17
fs3 = 17

CORRECAO = 0 #45 # correção provisória para eixo y

ATIVA_POLAR = 0 # 0 = desativa 1 = ativa diagrama polar
```

```
nivel = 1.0e-5      # controla nível máximo graficos 2D

MAXPOLAR_AUTOMATICO = 0 # automatico=0 ou usar Max_polarx
Max_polarx = 20 # posição do máximo no eixo x (0 a Max_polarx)
porcentagemdepx = 0.5 # para obter o maxpolar automático
#*****************************************************
#####################################################################

# LEITURA DE ARQUIVO DE ENTRADA
if USAR_ESTES_ARQUIVOS == 0:
   arquivo3 = open("entrada.txt","r")

if USAR_ESTES_ARQUIVOS == 1:
   arquivo3 = open("entrada.txt","r")

if USAR_ESTES_ARQUIVOS == 2:
   arquivo3 = open("entrada.txt","r")

Nx = int(arquivo3.readline()) #funcionou
Ny =  int(arquivo3.readline())
Nz =  int(arquivo3.readline())
ppw =  int(arquivo3.readline())
tempomax =  int(arquivo3.readline())
tipo_fonte =  int(arquivo3.readline())
HARD_SOFT =  int(arquivo3.readline())
ATIVANDO_DIPOLO_1 =  int(arquivo3.readline())
ATIVANDO_DIPOLO_2 =  int(arquivo3.readline())
rdsdz =  float(arquivo3.readline())
fator_comp =  float(arquivo3.readline())
arqsaida1 = str(arquivo3.readline())
arqsaida2 = str(arquivo3.readline())
delta_passo = int(arquivo3.readline())
fator_cdtds = float(arquivo3.readline())
fator_at = float(arquivo3.readline())
fator_geometria = int(arquivo3.readline())
arquivo3.close()

# RETIRANDO "\n" DAS STRINGS
txx1 = list(arqsaida1)
txx1.pop()
arqsaida1 = "".join(txx1)

txx2 = list(arqsaida2)
txx2.pop()
arqsaida2 = "".join(txx2)

if USAR_ESTES_ARQUIVOS == 0:
   arqsaida1 = "ez_pxy_1_384.txt"
   arqsaida2 = "ez_pxz_2_384.txt"

if USAR_ESTES_ARQUIVOS == 1:
   arqsaida1 = "ez_pxy_1_400.txt"
   arqsaida2 = "ez_pxz_2_400.txt"

if USAR_ESTES_ARQUIVOS == 2:
```

```
    arqsaida1 = "ez_pxy_1_394.txt"
    arqsaida2 = "ez_pxz_2_394.txt"

cdtds = 1.0/np.sqrt(fator_cdtds) # número de Courant

# IMPRIME VALORES OBTIDOS DO ARQUIVO DE ENTRADA
print 'Nx =',Nx
print 'Ny =',Ny
print 'Nz =',Nz
print 'ppw =',ppw
print 'tempomax =',tempomax
print 'tipo_fonte =',tipo_fonte
print 'HARD_SOFT =',HARD_SOFT
print 'ATIVANDO_DIPOLO_1 =',ATIVANDO_DIPOLO_1
print 'ATIVANDO_DIPOLO_2 =',ATIVANDO_DIPOLO_2
print 'rdsdz =', rdsdz
print 'cdtds =', cdtds
print 'fator comprimento dipolo  =',fator_comp
print 'fator atenuação (senóide) =',fator_at
print 'fator geometria (1, 2, 3) =',fator_geometria
print 'arqsaida1 =',arqsaida1
print 'arqsaida2 =',arqsaida2
# FINAL DA LEITURA


#**************************************

razaodsdz = round(rdsdz,2) # para gráfico
CFL = int(cdtds*100)/100.0 # número de Courant para gráfico


ativo = 1  # tipo dipolo_1
passivo = 0
comprimento_dipolo = int(fator_comp*rdsdz*ppw)
compdipolo = int(fator_comp*100)/100.0 # para gráfico
imp0 = 120*np.pi

#**********************************************


if (fator_geometria <= 2):
    ds = 1.0  # dimensão dos lados do hexagono 2D
    dx = ds*np.sqrt(3.0)/2.0 # tamanho da célula na direção x
    dy = ds/2.0   # tamanho da célula na direção y
    dz = ds/rdsdz
    Lx = 1.0*(Nx)*dx  # comprimento na direção x
    Ly = 1.0*(Ny)*dy  # comprimento na direção y
    Lz = 1.0*(Nz)*dz  # comprimento na direção z

#******************************************************

tipo_1d = 0 # grade 1D auxiliar
tipo_3d = 3 # grade 3D

# calculando dimensões das matrizes para os campos E e H
```

```
ix = Nx/2
rx = Nx%2
jy = Ny/2
ry = Ny%2
jy2 = (Ny-1)/2
ry2 = (Ny-1)%2

PEZ = (ix + rx)*(jy + ry) + (ix)*(jy)
PH2 = (ix)*(jy + ry) + (ix + rx)*(jy)
PH1 = (ix)*(jy2 + ry2) + (ix)*(jy2)
PH3 = PH1
#print 'PEZ= ',PEZ

# campos

# matrizes para os gráficos de Ez nos planos xy e xz
ez_pxy = np.zeros((Nx+1,Ny+1))
ez_pxz = np.zeros((Nx+1,Nz+1))

ez_pxy2 = np.zeros((Ny+1,Nx+1)) # transposta de ez_pxy
ez_pxz2 = np.zeros((Nz+1,Nx+1)) # transposta de ez_pxz

#************************************
px = Nx/2 + 1# posição da fonte ricker
py = Ny/2 + 1
pz = Nz/2 + 1
pcentro = p2b(px,py)
#*****************************************

# textos para gráficos
texto2 =  'dim='+str(Nx)+'x'+str(Ny)+'x'+str(Nz)+\
          '  ppw='+str(ppw)+'  CFL='+str(CFL)+\
          '  ds/dz='+str(razaodsdz)
if tipo_fonte == 0:
   texto1 = u'p_ricker '
else:
   texto1 = u'senóide '
if HARD_SOFT == 0:
   texto1 = texto1 + 'hard '
else:
   texto1 = texto1 + 'soft '
if ATIVANDO_DIPOLO_1 == 1 :
   texto3 = '  dipolo '+str(compdipolo)
else:
   texto3 = ''
#textof = texto1 + texto2 + texto3
textof = u'distância (metros)' # modificado para artigo momag
#************************************

# testando a fonte sozinha no tempo
ez_teste = np.zeros((tempomax))
tempo_teste = np.zeros((tempomax))

#******************************
```

```
# INICIO DOS CÁLCULOS PARA GRADE HEXAGONAL 3D

print u'\nINICIO FDTD'
print

# inicio loop do tempo

for tempo in range(0,tempomax):

    tempo_teste[tempo] = tempo
    ez_teste[tempo] = fonte_1(tempo,0.0,tipo_fonte,fator_at)

    #print '\n',tempo+1,'/',tempomax

# final loop do tempo
#************************************************

# LEITURA DAS MATRIZES PARA PLANOS XY E XZ
arquivo4 = open(arqsaida1,"r")
#linha = None
for i in range(0,Nx+1):
     for j in range(0,Ny+1):
        linha = arquivo4.readline() #funcionou
        ez_pxy[i,j] = float(linha)
arquivo4.close()

arquivo5 = open(arqsaida2,"r")
#linha = None
for i in range(0,Nx+1):
     for j in range(0,Nz+1):
        linha = arquivo5.readline() #funcionou
        ez_pxz[i,j] = float(linha)
arquivo5.close()

#*******************************************
# eliminando valores zero das matrizes para os gráficos de ez
for i in range(1,Nx+1): # plano xy
    for j in range(1,Ny+1):
        if (j%2==0 and i%2==1 and i<=Nx) or\
           (j%2==1 and i%2==0 and i<=Nx-1):
              inx, jnx = pteste(i,j+1)
              isx, jsx = pteste(i,j-1)
              ie, je   = pteste(i+1,j)
              iw, jw   = pteste(i-1,j)
              ez_pxy[i,j] = 0.25*(ez_pxy[inx,jnx] + ez_pxy[isx,jsx] +\
                          ez_pxy[ie,je] + ez_pxy[iw,jw])
j = py
for i in range(1,Nx+1):  # plano xz
    for k in range(1,Nz+1):
        if (j%2==0 and i%2==1 and i<=Nx) or\
           (j%2==1 and i%2==0 and i<=Nx-1):
              ie, ke   = pteste2(i+1,k)
              iw, kw   = pteste2(i-1,k)
              ez_pxz[i,k] = 0.5*(ez_pxz[ie,ke] + ez_pxz[iw,kw])
```

```
#************************************************

# gerando gráficos 1D a partir dos gráficos 2D
Nx2 = Nx - px
Ny2 = Ny - py
Nz2 = Nz - pz
ez_x1 = np.zeros((Nx2+1))
ez_y1 = np.zeros((Ny2+1))
ez_x2 = np.zeros((Nx2+1))
ez_z2 = np.zeros((Nz2+1))
x = np.zeros((Nx2+1))
y = np.zeros((Ny2+1))
z = np.zeros((Nz2+1))

for i in range(0,Nx2+1):
    x[i] = i*dx
    ez_x1[i] = ez_pxy[px+i,py] # plano xy
    ez_x2[i] = ez_pxz[px+i,pz] # plano xz

for j in range(0,Ny2+1):
    y[j] = j*dy
    ez_y1[j] = ez_pxy[px,py+j] # plano xy
    if (j > (Ny2+1 - CORRECAO)): # correção provisória
        ez_y1[j] = 0.0


for k in range(0,Nz2+1):
    z[k] = k*dz
    ez_z2[k] = ez_pxz[px,pz+k] # plano xz


if Nx < Ny: # diagonal do plano xy
    Nd = Nx2
else:
    Nd = Ny2
ez_d1 = np.zeros((Nd+1))
s = np.zeros((Nd+1))

for k in range(0,Nd+1):
    s[k] = k*ds
    ez_d1[k] = ez_pxy[px+k,py+k]

if Nx < Nz: # diagonal do plano xz
    Nd2 = Nx2
else:
    Nd2 = Nz2
ez_d2 = np.zeros((Nd2+1))
s2 = np.zeros((Nd2+1))

ds2 = np.sqrt(dx**2 + dz**2)
for k in range(0,Nd2+1):
    s2[k] = k*ds2

    ez_d2[k] = ez_pxz[px+k,pz+k]
```

```
#*****************************************
#*****************************************
# TESTANDO TAMANHOS DOS NÚMEROS REAIS
print ez_x1[11]
print "{0:>21.19e}".format(ez_x1[11])
print ez_x1[15]
print "{0:>21.19e}".format(ez_x1[15])
print

#*******************************************************

# NOVA FORMA DE MEDIR ANISOTROPIA
# NOVA FORMA DE MEDIDA DE ANISOTROPIA NO PLANO XY
# achando pontos de nulos para medir anisotropia (plano xy)
if tipo_fonte == 1:
    limiar = LIMIAR1
    limiary = LIMIAR1Y
    ncomp = 1.0*NUMERO_COMPRIMENTOS1 # numero de comprimentos de onda
    ncompy = 1.0*NUMERO_COMPRIMENTOS1Y
    ncompB = 1.0* NUMERO_COMPRIMENTOS1B

    x1_limiar = medidav2(ez_x1,Nx2,limiar,dx,ncomp,novo_alg)
    y1_limiar = medidav2(ez_y1,Ny2,limiary,dy,ncompy,novo_alg)
    d1_limiar = medidav2(ez_d1,Nd,limiar,ds,ncompB,novo_alg)

    dt_limiar = tempomax
    print u'# 1° PONTOS DE NULOS - PLANO XY'
    print u'# [x1_limiar, y1_limiar, d1_limiar, tempomax, ppw, HARD_SOFT]'
    print "{0:2}{1:>24.20f}{2:2}".format("[",x1_limiar,",")
    print "{0:>24.20f}{1:2}".format(y1_limiar,",")
    print "{0:>24.20f}{1:2}".format(d1_limiar,",")
    print "{0:>24.20f}{1:2}".format(dt_limiar,",")
    print "{0:>24.20f}{1:2}{2:2d}{3:2}".format(ppw,",",HARD_SOFT,"]")
    print
    limiar = LIMIAR1
    ncomp = 1.0*NUMERO_COMPRIMENTOS1 - DELTA_COMP1
    ncompy = 1.0*NUMERO_COMPRIMENTOS1Y - DELTA_COMP1
    ncompB = 1.0*NUMERO_COMPRIMENTOS1B - DELTA_COMP1

    x1_limiar2 = medidav2(ez_x1,Nx2,limiar,dx,ncomp,novo_alg)
    y1_limiar2 = medidav2(ez_y1,Ny2,limiary,dy,ncompy,novo_alg)
    d1_limiar2 = medidav2(ez_d1,Nd,limiar,ds,ncompB,novo_alg)

    dt_limiar = tempomax
    print u'# 2° PONTOS DE NULOS - PLANO XY'
    print u'# [x1_limiar, y1_limiar, d1_limiar, tempomax, ppw, HARD_SOFT]'
    print "{0:2}{1:>24.20f}{2:2}".format("[",x1_limiar2,",")
    print "{0:>24.20f}{1:2}".format(y1_limiar2,",")
    print "{0:>24.20f}{1:2}".format(d1_limiar2,",")
    print "{0:>24.20f}{1:2}".format(dt_limiar,",")
    print "{0:>24.20f}{1:2}{2:2d}{3:2}".format(ppw,",",HARD_SOFT,"]")
    print
    avy1 = 100.0*( y1_limiar2 - y1_limiar - (x1_limiar2 -\
              x1_limiar))/(x1_limiar2 - x1_limiar)
    avd1 = 100.0*( d1_limiar2 - d1_limiar - (x1_limiar2 -\
```

```
            x1_limiar))/(x1_limiar2 - x1_limiar)


#***************************************

# NOVA FORMA DE MEDIDA DE ANISOTROPIA NO PLANO XZ
# achando pontos de máximo para medir anisotropia (plano xz)
if tipo_fonte == 1:
    limiar = LIMIAR_X2
    ncomp = 1.0*NUMERO_COMPRIMENTOS_X2
    limiar2 = LIMIAR2
    ncomp2 = 1.0*NUMERO_COMPRIMENTOS2
    limiar3 = LIMIAR3
    ncomp3 = 1.0*NUMERO_COMPRIMENTOS3

    x2_limiar = medidav2(ez_x2,Nx2,limiar,dx,ncomp,novo_alg)
    z2_limiar = medidav2(ez_z2,Nz2,limiar2,dz,ncomp2,novo_alg)
    d2_limiar = medidav2(ez_d2,Nd2,limiar3,ds2,ncomp3,novo_alg)

    dt_limiar = tempomax
    print u'# 1° PONTOS DE NULOS - PLANO XZ'
    print u'# [x2_limiar, z2_limiar, d2_limiar, tempomax, ppw, HARD_SOFT]'
    print "{0:2}{1:>24.20f}{2:2}".format("[",x2_limiar,",")
    print "{0:>24.20f}{1:2}".format(z2_limiar,",")
    print "{0:>24.20f}{1:2}".format(d2_limiar,",")
    print "{0:>24.20f}{1:2}".format(dt_limiar,",")
    print "{0:>24.20f}{1:2}{2:2d}{3:2}".format(ppw,",",HARD_SOFT,"]")
    print
    limiar = LIMIAR_X2
    ncomp = 1.0*NUMERO_COMPRIMENTOS_X2 - DELTA_COMP1
    limiar2 = LIMIAR2
    ncomp2 = 1.0*NUMERO_COMPRIMENTOS2 - DELTA_COMP1
    limiar3 = LIMIAR3
    ncomp3 = 1.0*NUMERO_COMPRIMENTOS3 - DELTA_COMP1

    x2_limiar2 = medidav2(ez_x2,Nx2,limiar,dx,ncomp,novo_alg)
    z2_limiar2 = medidav2(ez_z2,Nz2,limiar2,dz,ncomp2,novo_alg)
    d2_limiar2 = medidav2(ez_d2,Nd2,limiar3,ds2,ncomp3,novo_alg)

    dt_limiar = tempomax
    print u'# 2° PONTOS DE NULOS - PLANO XZ'
    print u'# [x2_limiar, z2_limiar, d2_limiar, tempomax, ppw, HARD_SOFT]'
    print "{0:2}{1:>24.20f}{2:2}".format("[",x2_limiar2,",")
    print "{0:>24.20f}{1:2}".format(z2_limiar2,",")
    print "{0:>24.20f}{1:2}".format(d2_limiar2,",")
    print "{0:>24.20f}{1:2}".format(dt_limiar,",")
    print "{0:>24.20f}{1:2}{2:2d}{3:2}".format(ppw,",",HARD_SOFT,"]")
    #print
    avz2 = 100.0*( z2_limiar2 - z2_limiar - (x2_limiar2 -\
            x2_limiar))/(x2_limiar2 - x2_limiar)
    avd2 = 100.0*( d2_limiar2 - d2_limiar - (x2_limiar2 -\
            x2_limiar))/(x2_limiar2 - x2_limiar)
    ahex = ((np.pi/ppw)**4)*(100.0/360)
    ayee = ((np.pi/ppw)**2)*(100.0/12)
    print 'RESULTADOS DE ANISOTROPIA'
```

```
    print 'NUMERO_COMPRIMENTOS1 =',NUMERO_COMPRIMENTOS1
    print 'DELTA_COMPRIMENTOS1  =',DELTA_COMP1
    print 'PLANO XY'
    print 'avy1(%) =',avy1
    print 'avd1(%) =',avd1
    print 'ahex(%) =',ahex
    print 'PLANO XZ'
    print 'avz2(%) =',avz2
    print 'avd2(%) =',avd2
    print 'ayee(%) =',ayee
    print
# FINAL NOVA FORMA DE MEDIR ANISOTROPIA

#####################################################################

ez_pxy2 = ez_pxy.T # transposta para corrigir imagem
ez_pxz2 = ez_pxz.T # transposta para corrigir imagem

#*******************************************************

# PLOTANDO CAMPOS Ez PARA O MODO 3D
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
fs1 = 16
fs2 = 17
fs3 = 17
fse = 14
lw = 1.6

# nearest ou bilinear - funcionou
# cm.jet ou cm.RdYlGn ou cm.winter ou cm.hot
# ficou imagem correta com shape=None e com extent=[0,Lx,0,Ly]
# mas shape inverteu eixos x e y
# no entanto usando a transposta de ez a imagem ficou com eixos corretos

# GRÁFICOS 2D
if MOSTRAR_SOMENTE_GRAFICOS_2D == 1:
    # plano xy
    plt.figure()
    im = plt.imshow(ez_pxy2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Ly], shape=None,
                vmax= nivel*abs(ez_pxy2).max(), vmin= -nivel*abs(ez_pxy2).max()  )

    plt.colorbar(im) # funcionou
    plt.title(u'CAMPO 3D: Ez NO PLANO XY (TEMPO = '+str(tempomax)+' passos)')
    plt.xlabel(u'Eixo X '+ textof)
    plt.ylabel(u'Eixo Y')
    #plt.show()

    # plano xz
    plt.figure()
    im2 = plt.imshow(ez_pxz2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Lz], shape=None,
                vmax= nivel*abs(ez_pxz2).max(), vmin= -nivel*abs(ez_pxz2).max()  )
```

```
    plt.colorbar(im2) # funcionou
    plt.title(u'CAMPO 3D: Ez NO PLANO XZ (TEMPO = '+str(tempomax)+' passos)')
    plt.xlabel(u'Eixo X '+ textof)
    plt.ylabel(u'Eixo Z')
    #plt.show()

#*******************************************
if MOSTRAR_SOMENTE_GRAFICOS_1D == 1:
    # GRÁFICOS 1D
    # no plano xy
    fig = plt.figure()
    ax = fig.add_subplot(111)
    plt.tick_params(labelsize= fse)
    plt.plot(x,ez_x1,'r',linewidth=lw)
    plt.axis([0.,220.,-0.005,.008])
    plt.grid()
    plt.xlabel(u'Eixos x/y/d ( metros )',fontsize=fs2)
    plt.ylabel(u'Amplitude ( V/m )',fontsize=fs2)
    plt.title(u'HEX 3D: CAMPOS Ez NO PLANO XY (TEMPO = '+str(tempomax)+' passos)',
            fontsize=fs2)
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
    plt.plot(y,ez_y1,'b-',linewidth=lw)
    plt.plot(s,ez_d1,'k',linewidth=lw)
    texto1 = u'eixo_x1 (  0°)'
    texto2 = u'eixo_y1 (90°)'
    texto3 = u'eixo_d1 (30°)'
    plt.legend((texto1,texto2,texto3),loc = 'upper right')
    ax.annotate(u'PONTOS DE NULO (POSIÇÃO A)',fontsize=fs3, xy=(53.,0.00001),
          xytext=(59.,0.0033),arrowprops=dict(facecolor='black',width=4,
                           headwidth=12,frac= 0.13,shrink=0.05),)
    ax.annotate(u'PONTOS DE NULO (POSIÇÃO B)',fontsize=fs3, xy=(152.,0.00001),
           xytext=(155.,0.0016),arrowprops=dict(facecolor='black',width=4,
                          headwidth=12,frac= 0.3,shrink=0.05),)
    plt.hold(False)

    #plt.show()

    # no plano xz
    fig = plt.figure()
    ax = fig.add_subplot(111)
    plt.tick_params(labelsize= fse)
    plt.plot(x,ez_x2,'r',linewidth=lw)
    plt.axis([0.,220.,-0.005,.007])
    plt.grid()
    plt.xlabel(u'Eixos x/z/d ( metros )',fontsize=fs2)
    plt.ylabel(u'Amplitude ( V/m )',fontsize=fs2)
    plt.title(u'HEX 3D: CAMPOS Ez NO PLANO XZ (TEMPO = '+str(tempomax)+' passos)',
            fontsize=fs2)
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes

    plt.plot(z,ez_z2,'b-',linewidth=lw)
    plt.plot(s2,ez_d2,'k',linewidth=lw)
```

```
    texto1 = u'eixo_x2 (  0°)'
    texto2 = u'eixo_z2 (90°)'
    texto3 = u'eixo_d2 (45°)'
    plt.legend((texto1,texto2,texto3),loc = 'upper right')
    ax.annotate(u'PONTOS DE NULO (POSIÇÃO A)',fontsize=fs3, xy=(53.,0.00001),
          xytext=(59.,0.0033),arrowprops=dict(facecolor='black',width=4,
                              headwidth=12,frac= 0.13,shrink=0.05),)
    ax.annotate(u'PONTOS DE NULO (POSIÇÃO B)',fontsize=fs3, xy=(152.,0.00001),
           xytext=(155.,0.0016),arrowprops=dict(facecolor='black',width=4,
                             headwidth=12,frac= 0.3,shrink=0.05),)
    plt.hold(False)
    #plt.show()

if MOSTRAR_SOMENTE_GRAFICOS_1D == 1:
    # fonte em função do tempo
    plt.figure()
    plt.tick_params(labelsize= fse)
    plt.plot(tempo_teste,ez_teste,'k-',linewidth=lw)
    plt.grid()
    plt.xlabel(u'Tempo ( número de passos )',fontsize=fs2)
    plt.ylabel(u'Amplitude ( V )',fontsize=fs2)
    plt.title(u'FONTE SENOIDAL PARA CAMPO Ez ( TEMPO = '+str(tempomax)+\
            u' passos, FATOR_ATENUAÇÃO = '+str(fator_at)+' )',fontsize=fs2 )
    #plt.show()

plt.show()

# FINAL DO PROGRAMA
## FINAL DO APÊNDICE 6
```

APÊNDICE 7. PROGRAMAS PARA MEDIDA DE REFLEXÃO NAS PMLs PARA AMBOS OS MÉTODOS FDTD

ESTE APÊNDICE CONTÉM 2 ARQUIVOS

ARQUIVO 1: MEDIDAS DE REFLEXÃO PARA A GRADE DE PRISMAS HEXAGONAIS

```
# -*- coding: cp1252 -*-
#            cp1252 ou utf-8
# programa programa_fdtd_3d_h_2c_visualizacao_1d.py em python 2.7.3
# derivado do programa: programa_fdtd_3d_h_2c_visualizacao_1c.py
# AUTOR: MARINOEL JOAQUIM
#*********************************************************
# BIBLIOTECAS NECESSÁRIAS

import numpy as np
import time

#*********************************************************
tempo1 = time.time()

# INICIO DAS FUNÇÕES

# Função para hz e h3
def p2a(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx-1)
        p2 = (p1 + 1)/2
        if (p2 > PH3):
            p2 = 0
    return p2

# Função para ez e e3
def p2b(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx)
        p2 = (p1 + 1)/2
        if (p2 > PEZ):
            p2 = 0
    return p2

# Função para h2 e e1
def p2c(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx)
        p2 = (p1)/2
        if (p2 > PH2):
            p2 = 0
```

```
    return p2

# Função para h1 e e2
def p2d(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx-1)
        p2 = (p1 + 1)/2
        if (p2 > PH1):
            p2 = 0
    return p2
#*************************************

# FUNÇÃO PARA FONTE DO TIPO PULSO RICKER OU SENOIDE
def fonte_1(tempo2,posicao,tipo_fonte,fator_at):
    if tipo_fonte == 0: # pulso ricker
        arg = np.pi*( (cdtds*tempo2 - posicao)/ppw - 1.0 )
        arg2 = arg*arg
        funcao =(1.0 - 2.0*arg2)*np.exp(-arg2)

    else:  # senóide com amortecimento inicial
        periodo = ppw/cdtds # periodo da senóide
                        # ou tempo para o pico do pulso ricker
        cte_tempo = fator_at*periodo
        funcao = ( 1.0 - np.exp(-1.0*tempo2/cte_tempo) )*\
                np.sin(2*np.pi*tempo2*cdtds/ppw)
    return funcao
#***********************************

# função para obter i,j a partir de p para campo ez
def ez_ij(p2):
    p1 = 2*p2 - 1
    j = p1/Nx + 1
    if (p1%Nx==0 and p1>=Nx):
        j = j - 1
    i = p1 - (j-1)*Nx
    return i,j
#*********************************************************

# Função de interpolação para gráfico de ez no plano xy
def pteste(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        i = 0
        j = 0
    return i, j

# Função de interpolação para gráfico de ez no plano xz
def pteste2(i,k):
    if ((i<=0) or (i>=Nx+1) or (k<=0) or (k>=Nz+1)):
        i = 0
        k = 0
    return i, k
#*********************************************************
```

```
def label(x,y, text,tam_fonte):
    y = y - 0.0 # shift y-value for label so that it's below the artist
    plt.text(x, y, text, ha="center", family='sans-serif',
            size=tam_fonte,fontweight ='bold',color='k')

# FINAL DAS FUNÇÕES
#*******************************************************
#*******************************************************

# INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS
Nx = 81 # numero de pontos na direção x
Ny = 141  # numero de pontos na direção y
Nz = 71 # numero de pontos na direção z
ppw = 15 # numero de pontos por comprimento de onda
tempomax = 40

tipo_fonte = 1  # 0=pulso ricker ou 1=senoide
HARD_SOFT  = 1   # fonte: 0=hard ou 1=soft

rdsdz = 1.0 #np.sqrt(1.0) # razão ds/dz
fator_comp = 1.4 # em comprimentos de onda para dipolo_1

arqsaida1 = "ez_pxy_1.txt"
arqsaida2 = "ez_pxz_2.txt"
fator_at = 0.5

# FINAL DAS PRINCIPAIS VARIÁVEIS

#*************************************************

# VARIÁVEIS DE CONTROLE DOS GRÁFICOS

USAR_ESTES_ARQUIVOS = 0 # usar 1 ou 2, ou 0

MOSTRAR_SOMENTE_GRAFICOS_1D = 0 # 1= graficos 1D, 0 = mostra tudo

TESTE_REFLEXAO = 1 # 0 = desativa 1 = ativa
nivel = 0.3e-3  # controla nível máximo graficos 2D

#*******************************************************

# LEITURA DE ARQUIVO DE ENTRADA
if USAR_ESTES_ARQUIVOS == 0:
    arquivo3 = open("entrada3.txt","r")

if USAR_ESTES_ARQUIVOS == 1:
    arquivo3 = open("entrada_ta1.txt","r")

if USAR_ESTES_ARQUIVOS == 2:
    arquivo3 = open("entrada_ta2.txt","r")

Nx = int(arquivo3.readline())#funcionou
Ny =  int(arquivo3.readline())
Nz =  int(arquivo3.readline())
```

```
ppw =  int(arquivo3.readline())
tempomax =  int(arquivo3.readline())
tipo_fonte =  int(arquivo3.readline())
HARD_SOFT =  int(arquivo3.readline())
ds =  float(arquivo3.readline())
rdsdz =  float(arquivo3.readline())
fator_cdtds = float(arquivo3.readline())
fator_at =  float(arquivo3.readline())
delta_passo = int(arquivo3.readline())
tempomax2 =  int(arquivo3.readline())
tempomax3 =  int(arquivo3.readline())
CPML =  int(arquivo3.readline())
nxx =  int(arquivo3.readline())
Reflexao = float(arquivo3.readline())
Reflexao2 = float(arquivo3.readline())
me = float(arquivo3.readline())
me2 = float(arquivo3.readline())
numero_materiais = int(arquivo3.readline())
numero_objetos = int(arquivo3.readline())
arquivo3.close()

fator_geometria = 2
# RETIRANDO "\n" DAS STRINGS
txx1 = list(arqsaida1)
txx1.pop()
arqsaida1 = "".join(txx1)

txx2 = list(arqsaida2)
txx2.pop()
arqsaida2 = "".join(txx2)

if USAR_ESTES_ARQUIVOS == 0:
    arqsaida1 = "ez_pxy_1_257.txt"
    arqsaida2 = "ez_pxz_2_257.txt"
    if TESTE_REFLEXAO == 1:
        arqsaida3  = "ez_A_nxx_20_CPML_0_Temp_257.txt"
        arqsaida3b = "ez_A_nxx_20_CPML_1_Temp_257.txt"
        arqsaida4  = "ez_B_nxx_20_CPML_0_Temp_257.txt"
        arqsaida4b = "ez_B_nxx_20_CPML_1_Temp_257.txt"
        arqsaida5  = "ez_C_nxx_20_CPML_0_Temp_257.txt"
        arqsaida5b = "ez_C_nxx_20_CPML_1_Temp_257.txt"
        arqsaida6  = "ez_D_nxx_20_CPML_0_Temp_257.txt"
        arqsaida6b = "ez_D_nxx_20_CPML_1_Temp_257.txt"

if USAR_ESTES_ARQUIVOS == 1:
    arqsaida1 = "ez_pxy_1_ta1.txt"
    arqsaida2 = "ez_pxz_2_ta1.txt"

if USAR_ESTES_ARQUIVOS == 2:
    arqsaida1 = "ez_pxy_1_ta2.txt"
    arqsaida2 = "ez_pxz_2_ta2.txt"

cdtds = 1.0/np.sqrt(fator_cdtds) # número de Courant

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

```
eo = 1.0e-9/(36*np.pi)
uo = np.pi*4e-7
co = 1.0/np.sqrt(uo*eo)
#print 'co =',co

# IMPRIME VALORES OBTIDOS DO ARQUIVO DE ENTRADA
print 'Nx =',Nx
print 'Ny =',Ny
print 'Nz =',Nz
print 'ppw =',ppw
print 'tempomax =',tempomax
print 'tipo_fonte =',tipo_fonte
print 'HARD_SOFT =',HARD_SOFT
#print 'ATIVANDO_DIPOLO_1 =',ATIVANDO_DIPOLO_1
#print 'ATIVANDO_DIPOLO_2 =',ATIVANDO_DIPOLO_2
print 'rdsdz =', rdsdz
print 'cdtds =', cdtds
#print 'fator comprimento dipolo  =',fator_comp
print 'fator atenuação (senóide) =',fator_at
#print 'fator geometria (1, 2, 3) =',fator_geometria
#print 'arqsaida1 =',arqsaida1
#print 'arqsaida2 =',arqsaida2
# FINAL DA LEITURA


#**************************************

razaodsdz = round(rdsdz,2) # para gráfico
CFL = int(cdtds*100)/100.0 # número de Courant para gráfico


ativo = 1  # tipo dipolo_1
passivo = 0
comprimento_dipolo = int(fator_comp*rdsdz*ppw)
compdipolo = int(fator_comp*100)/100.0 # para gráfico
imp0 = 120.0*np.pi

#**********************************************

if (fator_geometria <= 2):
    ds = 1.0  # dimensão dos lados do hexagono maior 2D
    dx = ds*np.cos(np.pi/6) # tamanho da célula na direção x
    dy = ds/2.0   # tamanho da célula na direção y
    dz = ds/rdsdz
    Lx = (Nx)*dx  # comprimento na direção x
    Ly = (Ny)*dy  # comprimento na direção y
    Lz = (Nz)*dz  # comprimento na direção z

#****************************************************

tipo_1d = 0 # grade 1D auxiliar
tipo_3d = 3 # grade 3D

# calculando dimensões das matrizes para os campos E e H
ix = Nx/2
```

```
rx = Nx%2
jy = Ny/2
ry = Ny%2
jy2 = (Ny-1)/2
ry2 = (Ny-1)%2

PEZ = (ix + rx)*(jy + ry) + (ix)*(jy)
PH2 = (ix)*(jy + ry) + (ix + rx)*(jy)
PH1 = (ix)*(jy2 + ry2) + (ix)*(jy2)
PH3 = PH1
#print 'PEZ= ',PEZ

# matrizes para os gráficos de Ez nos planos xy e xz
ez_pxy = np.zeros((Nx+1,Ny+1))
ez_pxz = np.zeros((Nx+1,Nz+1))

ez_pxy2 = np.zeros((Ny+1,Nx+1)) # transposta de ez_pxy
ez_pxz2 = np.zeros((Nz+1,Nx+1)) # transposta de ez_pxz

#************************************
px = Nx/2 + 1 # posição da fonte ricker
py = Ny/2 + 1
pz = Nz/2 + 1
pcentro = p2b(px,py)
#*****************************************

# textos para gráficos
texto2 =  'dim='+str(Nx)+'x'+str(Ny)+'x'+str(Nz)+\
          '  ppw='+str(ppw)+'  CFL='+str(CFL)+\
          '  ds/dz='+str(razaodsdz) + ' CPML='+ str(nxx)
if tipo_fonte == 0:
   texto1 = u'p_ricker '
else:
   texto1 = u'senóide '
if HARD_SOFT == 0:
   texto1 = texto1 + 'hard '
else:
   texto1 = texto1 + 'soft '
if ATIVANDO_DIPOLO_1 == 1 :
   texto3 = '  dipolo '+str(compdipolo)
else:
   texto3 = ''
textof = texto1 + texto2 + texto3
#************************************

# testando a fonte sozinha no tempo
ez_teste = np.zeros((tempomax))
tempo_teste = np.zeros((tempomax))

#******************************

# INICIO DOS CÁLCULOS PARA GRADE HEXAGONAL 3D

print u'\nINICIO FDTD'
print
```

```
# inicio loop do tempo

for tempo in range(0,tempomax):

    tempo_teste[tempo] = tempo
    ez_teste[tempo] = fonte_1(tempo,0.0,tipo_fonte,fator_at)

    #print '\n',tempo+1,'/',tempomax

# final loop do tempo
#***********************************************


# LEITURA DAS MATRIZES PARA PLANOS XY E XZ
arquivo4 = open(arqsaida1,"r")
#linha = None
for i in range(0,Nx+1):
    for j in range(0,Ny+1):
        linha = arquivo4.readline() #funcionou
        ez_pxy[i,j] = float(linha)
arquivo4.close()

arquivo5 = open(arqsaida2,"r")
#linha = None
for i in range(0,Nx+1):
    for j in range(0,Nz+1):
        linha = arquivo5.readline() #funcionou
        ez_pxz[i,j] = float(linha)
arquivo5.close()

#***********************************************
# LEITURA DOS ARQUIVOS PARA MEDIR REFLEXAO NA CPML
eza0 = np.zeros((tempomax))
eza1 = np.zeros((tempomax))
erro_eza =  np.zeros((tempomax))
ezb0 = np.zeros((tempomax))
ezb1 = np.zeros((tempomax))
erro_ezb =  np.zeros((tempomax))
ezc0 = np.zeros((tempomax))
ezc1 = np.zeros((tempomax))
erro_ezc =  np.zeros((tempomax))
ezd0 = np.zeros((tempomax))
ezd1 = np.zeros((tempomax))
erro_ezd =  np.zeros((tempomax))

# LEITURA PONTO A
arquivo4 = open(arqsaida3,"r")
for i in range(0,tempomax):
    linha = arquivo4.readline() #funcionou
    eza0[i]= float(linha)
arquivo4.close()
arquivo4 = open(arqsaida3b,"r")
for i in range(0,tempomax):
    linha = arquivo4.readline() #funcionou
```

```
    eza1[i]= float(linha)
arquivo4.close()
# LEITURA PONTO B
arquivo4 = open(arqsaida4,"r")
for i in range(0,tempomax):
    linha = arquivo4.readline() #funcionou
    ezb0[i]= float(linha)
arquivo4.close()
arquivo4 = open(arqsaida4b,"r")
for i in range(0,tempomax):
    linha = arquivo4.readline() #funcionou
    ezb1[i]= float(linha)
arquivo4.close()
# LEITURA PONTO C
arquivo4 = open(arqsaida5,"r")
for i in range(0,tempomax):
    linha = arquivo4.readline() #funcionou
    ezc0[i]= float(linha)
arquivo4.close()
arquivo4 = open(arqsaida5b,"r")
for i in range(0,tempomax):
    linha = arquivo4.readline() #funcionou
    ezc1[i]= float(linha)
arquivo4.close()
# LEITURA PONTO D
arquivo4 = open(arqsaida6,"r")
for i in range(0,tempomax):
    linha = arquivo4.readline() #funcionou
    ezd0[i]= float(linha)
arquivo4.close()
arquivo4 = open(arqsaida6b,"r")
for i in range(0,tempomax):
    linha = arquivo4.readline() #funcionou
    ezd1[i]= float(linha)
arquivo4.close()

# CALCULO COEFICIENTE DE REFLEXÃO
erroref_a = 0.0
nerros_a = 0
erroref_b = 0.0
nerros_b = 0
erroref_c = 0.0
nerros_c = 0
erroref_d = 0.0
nerros_d = 0

for i in range(0,tempomax):
   erro_eza[i] = np.abs((eza1[i] - eza0[i])/(eza0[i] + 1e-20))
   if erro_eza[i] > 0.0:
      erroref_a = erroref_a + erro_eza[i]
      nerros_a = nerros_a + 1
   erro_ezb[i] = np.abs((ezb1[i] - ezb0[i])/(ezb0[i] + 1e-20))
   if erro_ezb[i] > 0.0:
      erroref_b = erroref_b + erro_ezb[i]
      nerros_b = nerros_b + 1
```

```
    erro_ezc[i] = np.abs((ezc1[i] - ezc0[i])/(ezc0[i] + 1e-20))
    if erro_ezc[i] > 0.0:
        erroref_c = erroref_c + erro_ezc[i]
        nerros_c = nerros_c + 1
    erro_ezd[i] = np.abs((ezd1[i] - ezd0[i])/(ezd0[i] + 1e-29))
    if erro_ezd[i] > 0.0:
        erroref_d = erroref_d + erro_ezd[i]
        nerros_d = nerros_d + 1
erroref_medio_a = erroref_a/nerros_a
erroref_medio_b = erroref_b/nerros_b
erroref_medio_c = erroref_c/nerros_c
erroref_medio_d = erroref_d/nerros_d
print 'MEDIDAS DE REFLEXÃO:'
print 'ESPESSURA_CPML_X = ',nxx
print 'REFLEXAO = ',Reflexao
print 'me = ',me
print 'NUMERO_PASSOS = ',tempomax
print 'NUMERO_ERROS_A = ',nerros_a
print 'COEF_REFLEXAO_MEDIO_A = ',erroref_medio_a
print 'NUMERO_ERROS_B = ',nerros_b
print 'COEF_REFLEXAO_MEDIO_B = ',erroref_medio_b
print 'NUMERO_ERROS_C = ',nerros_c
print 'COEF_REFLEXAO_MEDIO_C = ',erroref_medio_c
print 'NUMERO_ERROS_D = ',nerros_d
print 'COEF_REFLEXAO_MEDIO_D = ',erroref_medio_d
print
print


#*******************************************

# eliminando valores zero das matrizes para os gráficos de ez
for i in range(1,Nx+1): # plano xy
    for j in range(1,Ny+1):
        if (j%2==0 and i%2==1 and i<=Nx) or\
           (j%2==1 and i%2==0 and i<=Nx-1):
            inx, jnx = pteste(i,j+1)
            isx, jsx = pteste(i,j-1)
            ie, je   = pteste(i+1,j)
            iw, jw   = pteste(i-1,j)
            ez_pxy[i,j] = 0.25*(ez_pxy[inx,jnx] + ez_pxy[isx,jsx] +\
                        ez_pxy[ie,je] + ez_pxy[iw,jw])
j = py
for i in range(1,Nx+1):  # plano xz
    for k in range(1,Nz+1):
        if (j%2==0 and i%2==1 and i<=Nx) or\
           (j%2==1 and i%2==0 and i<=Nx-1):
            ie, ke   = pteste2(i+1,k)
            iw, kw   = pteste2(i-1,k)
            ez_pxz[i,k] = 0.5*(ez_pxz[ie,ke] + ez_pxz[iw,kw])

#*******************************************

# gerando gráficos 1D a partir dos gráficos 2D
Nx2 = Nx - px
```

```
Ny2 = Ny - py
Nz2 = Nz - pz
ez_x1 = np.zeros((Nx2+1))
ez_y1 = np.zeros((Ny2+1))
ez_x2 = np.zeros((Nx2+1))
ez_z2 = np.zeros((Nz2+1))
x = np.zeros((Nx2+1))
y = np.zeros((Ny2+1))
z = np.zeros((Nz2+1))

for i in range(0,Nx2+1):
    x[i] = i*dx
    ez_x1[i] = ez_pxy[px+i,py] # plano xy
    ez_x2[i] = ez_pxz[px+i,pz] # plano xz

for j in range(0,Ny2+1):
    y[j] = j*dy
    ez_y1[j] = ez_pxy[px,py+j] # plano xy

for k in range(0,Nz2+1):
    z[k] = k*dz
    ez_z2[k] = ez_pxz[px,pz+k] # plano xz


if Nx < Ny: # diagonal do plano xy
    Nd = Nx2
else:
    Nd = Ny2
ez_d1 = np.zeros((Nd+1))
s = np.zeros((Nd+1))

for k in range(0,Nd+1):
    s[k] = k*ds
    ez_d1[k] = ez_pxy[px+k,py+k]

if Nx < Nz: # diagonal do plano xz
    Nd2 = Nx2
else:
    Nd2 = Nz2
ez_d2 = np.zeros((Nd2+1))
s2 = np.zeros((Nd2+1))

ds2 = np.sqrt(dx**2 + dz**2)
for k in range(0,Nd2+1):
    s2[k] = k*ds2

    ez_d2[k] = ez_pxz[px+k,pz+k]

#****************************************

ez_pxy2 = ez_pxy.T # transposta para corrigir imagem
ez_pxz2 = ez_pxz.T # transposta para corrigir imagem

#******************************************************
```

```
# PLOTANDO CAMPOS Ez PARA O MODO 3D
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

# nearest ou bilinear - funcionou
# cm.jet ou cm.RdYlGn ou cm.winter ou cm.hot
# ficou imagem correta com shape=None e com extent=[0,Lx,0,Ly]
# mas shape inverteu eixos x e y
# no entanto usando a transposta de ez a imagem ficou com eixos corretos

# GRÁFICOS 2D
if MOSTRAR_SOMENTE_GRAFICOS_1D == 0:
    fs1 = 14   # xlabel e ylabel
    fs2 = 15   # title
    fse = 13   # numeros nos eixos x e y

    # plano xy
    plt.figure()
    im = plt.imshow(ez_pxy2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Ly], shape=None,
                vmax= nivel*abs(ez_pxy2).max(), vmin= -nivel*abs(ez_pxy2).max()  )

    plt.colorbar(im) # funcionou
    plt.title(u'HEX 3D: Ez (V/m) NO PLANO XY (T = '+str(tempomax)+' passos)')
    plt.xlabel(u'Lx (metros) ',fontsize = fs1)
    plt.ylabel(u'Ly (metros)',fontsize = fs1)
    #plt.show()
    # posições dos pontos de medida de reflexão
    label((px+71)*dx,(py-0)*dy,u'*',20)
    label((px+71+8)*dx,(py-0)*dy,u'A',18)
    label((px+0)*dx,(py + 121 -0)*dy,u'*',20)
    label((px+8)*dx,(py + 121 -0)*dy,u'B',18)
    label((px+71)*dx,(py + 121 -0)*dy,u'*',20)
    label((px+71+8)*dx,(py + 121 -0)*dy,u'C',18)

    # plano xz
    plt.figure()
    im2 = plt.imshow(ez_pxz2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Lz], shape=None,
                vmax= nivel*abs(ez_pxz2).max(), vmin= -nivel*abs(ez_pxz2).max()  )

    plt.colorbar(im2) # funcionou
    plt.title(u'HEX 3D: Ez (V/m) NO PLANO XZ (T = '+str(tempomax)+' passos)')
    plt.xlabel(u'Lx (metros) ',fontsize = fs1)
    plt.ylabel(u'Lz (metros)',fontsize = fs1)
    #plt.show()
    # posições dos pontos de medida de reflexão
    label(px*dx,(pz - 1 + 71)*dz,u'*',20)
    label((px+ 8)*dx,(pz - 1 + 71)*dz,u'D',18)

#*******************************************
# GRÁFICOS 1D
fs1 = 16   # xlabel e ylabel
fs2 = 17   # title
```

```
fse = 15   # numeros nos eixos x e y
# no plano xy
plt.figure()
plt.tick_params(labelsize = fse) # novo
plt.plot(x,ez_x1,'r')
plt.axis([0.,100.,-0.0013,0.0018])
plt.grid()
plt.xlabel(u'Eixos x/y/d (metros) ',fontsize = fs1) # novo
plt.ylabel(u'Amplitude (V/m)',fontsize = fs1)
plt.title('CAMPOS 1D: Ez (V/m) NO PLANO XY (TEMPO = ' +\
            str(tempomax)+' passos)',fontsize = fs2)
plt.hold(True) # permite sobrepor vários
            # gráficos com dimensões diferentes
plt.plot(y,ez_y1,'b')
plt.plot(s,ez_d1,'k-')
texto1 = u'ez_x1 ( 0°)'
texto2 = u'ez_y1 (90°)'
texto3 = u'ez_d1 (30°)'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)

#plt.show()

# no plano xz
plt.figure()
plt.tick_params(labelsize = fse) # novo
plt.plot(x,ez_x2,'r')
plt.axis([0.,120.,-0.0019,0.0018])
plt.grid()
plt.xlabel(u'Eixos x/z/d (metros) ',fontsize = fs1)
plt.ylabel(u'Amplitude (V/m)',fontsize = fs1)
plt.title('CAMPOS 1D: Ez (V/m) NO PLANO XZ (TEMPO = ' +\
                str(tempomax)+' passos)',fontsize = fs2)
plt.hold(True) # permite sobrepor vários
            # gráficos com dimensões diferentes

plt.plot(z,ez_z2,'b-')
plt.plot(s2,ez_d2,'k')
texto1 = u'ez_x2 ( 0°)'
texto2 = u'ez_z2 (90°)'
texto3 = u'ez_d2 (45°)'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)
#plt.show()

#**************************************************
if TESTE_REFLEXAO == 1:
    fs1 = 14   # xlabel e ylabel
    fs2 = 15   # title
    fse = 13   # numeros nos eixos x e y

    # PONTO A
    plt.figure()
    plt.plot(tempo_teste,eza0,'r',tempo_teste,eza1,'k')
    plt.grid()
```

```
plt.xlabel(u'Tempo (número de passos)',fontsize = fs1)
plt.ylabel(u'Amplitude (V/m)',fontsize = fs1)
plt.title(u'Ponto A: campos Ez ( T = '+str(tempomax)+\
        ' passos, FATOR_AT = '+str(fator_at)+' )',fontsize = fs2 )
texto1 = u'sem CPML'
texto2 = u'com CPML'
plt.legend((texto1,texto2),loc = 'upper left')

plt.figure()
plt.plot(tempo_teste,erro_eza,'k')
plt.grid()
plt.xlabel(u'Tempo (número de passos)',fontsize = fs1)
plt.ylabel(u'Amplitude (adimensional)',fontsize = fs1)
plt.title(u'Ponto A: Fator de reflexão ( T = '+str(tempomax)+\
        ' passos, FATOR_AT = '+str(fator_at)+' )',fontsize = fs2 )
# PONTO B
plt.figure()
plt.plot(tempo_teste,ezb0,'r',tempo_teste,ezb1,'k')
plt.grid()
plt.xlabel(u'Tempo (número de passos)',fontsize = fs1)
plt.ylabel(u'Amplitude (V/m)',fontsize = fs1)
plt.title(u'Ponto B: os campos Ez ( TEMPO = '+str(tempomax)+\
        ' passos, FATOR_AT = '+str(fator_at)+' )',fontsize = fs2 )
texto1 = u'sem CPML'
texto2 = u'com CPML'
plt.legend((texto1,texto2),loc = 'upper left')
plt.figure()
plt.plot(tempo_teste,erro_ezb,'k')
plt.grid()
plt.xlabel(u'Tempo (número de passos)',fontsize = fs1)
plt.ylabel(u'Amplitude (adimensional)',fontsize = fs1)
plt.title(u'Ponto B: Fator de reflexão ( T = '+str(tempomax)+\
        ' passos, FATOR_AT = '+str(fator_at)+' )',fontsize = fs2)
# PONTO C
plt.figure()
plt.plot(tempo_teste,ezc0,'r',tempo_teste,ezc1,'k')
plt.grid()
plt.xlabel(u'Tempo (número de passos)',fontsize = fs1)
plt.ylabel(u'Amplitude (V/m)',fontsize = fs1)
plt.title(u'Ponto C: os campos Ez ( TEMPO = '+str(tempomax)+\
        ' passos, FATOR_AT = '+str(fator_at)+' )',fontsize = fs2 )
texto1 = u'sem CPML'
texto2 = u'com CPML'
plt.legend((texto1,texto2),loc = 'upper left')

plt.figure()
plt.plot(tempo_teste,erro_ezc,'k')
plt.grid()
plt.xlabel(u'Tempo (número de passos)',fontsize = fs1)
plt.ylabel(u'Amplitude (adimensional)',fontsize = fs1)
plt.title(u'Ponto C: Fator de reflexão ( T = '+str(tempomax)+\
        ' passos, FATOR_AT = '+str(fator_at)+' )',fontsize = fs2 )
# PONTO D
plt.figure()
plt.plot(tempo_teste,ezd0,'r',tempo_teste,ezd1,'k')
```

```
    plt.grid()
    plt.xlabel(u'Tempo (número de passos)',fontsize = fs1)
    plt.ylabel(u'Amplitude (V/m)',fontsize = fs1)
    plt.title(u'Ponto D: campos Ez ( T = '+str(tempomax)+\
            ' passos, FATOR_AT = '+str(fator_at)+' )',fontsize = fs2 )
    texto1 = u'sem CPML'
    texto2 = u'com CPML'
    plt.legend((texto1,texto2),loc = 'upper left')

    plt.figure()
    plt.plot(tempo_teste,erro_ezd,'k')
    plt.grid()
    plt.xlabel(u'Tempo (número de passos)',fontsize = fs1)
    plt.ylabel(u'Amplitude (adimensional)',fontsize = fs1)
    plt.title(u'Ponto D: Fator de reflexão ( T = '+str(tempomax)+\
            ' passos, FATOR_AT = '+str(fator_at)+' )',fontsize = fs2 )
#*********************************************************

if MOSTRAR_SOMENTE_GRAFICOS_1D == 1:
    # fonte em função do tempo
    plt.figure()
    plt.plot(tempo_teste,ez_teste,'k')
    plt.grid()
    plt.xlabel(u'Eixo Tempo, '+ textof)
    plt.ylabel(u'Nível')
    plt.title('Fonte para campo Ez ( TEMPO = '+str(tempomax)+\
            ' passos, FATOR_AT = '+str(fator_at)+' )' )
    #plt.show()

plt.show()

# FINAL DO PROGRAMA
#################################################################
```

ARQUIVO 2: MEDIDAS DE REFLEXÃO PARA A GRADE YEE

```
# -*- coding: cp1252 -*-
#            cp1252 ou utf-8
# programa programa_fdtd_3d_h_2c_visualizacao_1d.py em python 2.7.3
# derivado do programa: programa_fdtd_3d_h_2c_visualizacao_1c.py
# AUTOR: MARINOEL JOAQUIM

#*********************************************************
# BIBLIOTECAS NECESSÁRIAS

import numpy as np
import time

#*********************************************************
tempo1 = time.time()

# INICIO DAS FUNÇÕES

# FUNÇÃO PARA FONTE DO TIPO PULSO RICKER OU SENOIDE
def fonte_1(tempo2,posicao,tipo_fonte,fator_at):
```

```
    if tipo_fonte == 0: # pulso ricker
        arg = np.pi*( (cdtds*tempo2 - posicao)/ppw - 1.0 )
        arg2 = arg*arg
        funcao =(1.0 - 2.0*arg2)*np.exp(-arg2)

    else:  # senóide com amortecimento inicial
        periodo = ppw/cdtds # periodo da senóide
                        # ou tempo para o pico do pulso ricker
        cte_tempo = fator_at*periodo
        funcao = ( 1.0 - np.exp(-1.0*tempo2/cte_tempo) )*\
                np.sin(2*np.pi*tempo2*cdtds/ppw)
    return funcao
#**********************************

# função para obter i,j a partir de p para campo ez
def ez_ij(p2):
    p1 = 2*p2 - 1
    j = p1/Nx + 1
    if (p1%Nx==0 and p1>=Nx):
        j = j - 1
    i = p1 - (j-1)*Nx
    return i,j
#**********************************************************

# Função de interpolação para gráfico de ez no plano xy
def pteste(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        i = 0
        j = 0
    return i, j

# Função de interpolação para gráfico de ez no plano xz
def pteste2(i,k):
    if ((i<=0) or (i>=Nx+1) or (k<=0) or (k>=Nz+1)):
        i = 0
        k = 0
    return i, k
#**********************************************************
def label(x,y, text,tam_fonte):
    y = y - 0.0 # shift y-value for label so that it's below the artist
    plt.text(x, y, text, ha="center", family='sans-serif',
            size=tam_fonte,fontweight ='bold',color='k')

# FINAL DAS FUNÇÕES
#**********************************************************
#**********************************************************

# INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS
Nx = 81 # numero de pontos na direção x
Ny = 141  # numero de pontos na direção y
Nz = 71 # numero de pontos na direção z
ppw = 15 # numero de pontos por comprimento de onda
tempomax = 40

tipo_fonte = 1  # 0=pulso ricker ou 1=senoide
```

```
HARD_SOFT  = 1   # fonte: 0=hard ou 1=soft
ATIVANDO_DIPOLO_1 = 0 # 1=ativa 0=desativa
ATIVANDO_DIPOLO_2 = 0 # 1=ativa 0=desativa

rdsdz = 1.0 #np.sqrt(1.0) # razão ds/dz
fator_comp = 1.4 # em comprimentos de onda para dipolo_1

arqsaida1 = "ez_pxy_1.txt"
arqsaida2 = "ez_pxz_2.txt"
fator_at = 0.5

# FINAL DAS PRINCIPAIS VARIÁVEIS

#************************************************

# VARIÁVEIS DE CONTROLE DOS GRÁFICOS

USAR_ESTES_ARQUIVOS = 0 # usar 1 ou 2, ou 0

MOSTRAR_SOMENTE_GRAFICOS_1D = 0 # 1= graficos 1D, 0 = mostra tudo

TESTE_REFLEXAO = 1 # 0 = desativa 1 = ativa
nivel = 3e-4  # controla nível máximo graficos 2D
#*****************************************************


# LEITURA DE ARQUIVO DE ENTRADA
if USAR_ESTES_ARQUIVOS == 0:
    arquivo3 = open("entrada4.txt","r")

if USAR_ESTES_ARQUIVOS == 1:
    arquivo3 = open("entrada_ta1.txt","r")

if USAR_ESTES_ARQUIVOS == 2:
    arquivo3 = open("entrada_ta2.txt","r")

Nx = int(arquivo3.readline())#funcionou
Ny =  int(arquivo3.readline())
Nz =  int(arquivo3.readline())
ppw =  int(arquivo3.readline())
tempomax =  int(arquivo3.readline())
tipo_fonte =  int(arquivo3.readline())
HARD_SOFT =  int(arquivo3.readline())
ds =  float(arquivo3.readline())
rdsdz =  float(arquivo3.readline())
fator_cdtds = float(arquivo3.readline())
fator_at =  float(arquivo3.readline())
delta_passo = int(arquivo3.readline())
tempomax2 =  int(arquivo3.readline())
tempomax3 =  int(arquivo3.readline())
CPML =  int(arquivo3.readline())
nxx =  int(arquivo3.readline())
Reflexao = float(arquivo3.readline())
Reflexao2 = float(arquivo3.readline())
me = float(arquivo3.readline())
```

```
me2 = float(arquivo3.readline())
numero_materiais = int(arquivo3.readline())
numero_objetos = int(arquivo3.readline())
arquivo3.close()

fator_geometria = 2
# RETIRANDO "\n" DAS STRINGS
txx1 = list(arqsaida1)
txx1.pop()
arqsaida1 = "".join(txx1)

txx2 = list(arqsaida2)
txx2.pop()
arqsaida2 = "".join(txx2)

if USAR_ESTES_ARQUIVOS == 0:
   arqsaida1 = "ez_pxy_yee1_243.txt"
   arqsaida2 = "ez_pxz_yee2_243.txt"
   if TESTE_REFLEXAO == 1:
      arqsaida3  = "ez_YA_nxx_18_CPML_0_Temp_243.txt"
      arqsaida3b = "ez_YA_nxx_18_CPML_1_Temp_243.txt"
      arqsaida4  = "ez_YB_nxx_18_CPML_0_Temp_243.txt"
      arqsaida4b = "ez_YB_nxx_18_CPML_1_Temp_243.txt"
      arqsaida5  = "ez_YC_nxx_18_CPML_0_Temp_243.txt"
      arqsaida5b = "ez_YC_nxx_18_CPML_1_Temp_243.txt"
      arqsaida6  = "ez_YD_nxx_18_CPML_0_Temp_243.txt"
      arqsaida6b = "ez_YD_nxx_18_CPML_1_Temp_243.txt"

if USAR_ESTES_ARQUIVOS == 1:
   arqsaida1 = "ez_pxy_1_ta1.txt"
   arqsaida2 = "ez_pxz_2_ta1.txt"

if USAR_ESTES_ARQUIVOS == 2:
   arqsaida1 = "ez_pxy_1_ta2.txt"
   arqsaida2 = "ez_pxz_2_ta2.txt"

cdtds = 1.0/np.sqrt(fator_cdtds) # número de Courant

#####################################################
eo = 1.0e-9/(36*np.pi)
uo = np.pi*4e-7
co = 1.0/np.sqrt(uo*eo)
print 'co =',co

# IMPRIME VALORES OBTIDOS DO ARQUIVO DE ENTRADA
print 'Nx =',Nx
print 'Ny =',Ny
print 'Nz =',Nz
print 'ppw =',ppw
print 'tempomax =',tempomax
print 'tipo_fonte =',tipo_fonte
print 'HARD_SOFT =',HARD_SOFT
print 'ATIVANDO_DIPOLO_1 =',ATIVANDO_DIPOLO_1
print 'ATIVANDO_DIPOLO_2 =',ATIVANDO_DIPOLO_2
print 'rdsdz =', rdsdz
```

```
print 'cdtds =', cdtds
print 'fator comprimento dipolo  =',fator_comp
print 'fator atenuação (senóide) =',fator_at
#print 'fator geometria (1, 2, 3) =',fator_geometria
#print 'arqsaida1 =',arqsaida1
#print 'arqsaida2 =',arqsaida2
# FINAL DA LEITURA


#************************************

razaodsdz = round(rdsdz,2) # para gráfico
CFL = int(cdtds*100)/100.0 # número de Courant para gráfico


ativo = 1  # tipo dipolo_1
passivo = 0
comprimento_dipolo = int(fator_comp*rdsdz*ppw)
compdipolo = int(fator_comp*100)/100.0 # para gráfico
imp0 = 120.0*np.pi

#************************************************

if (fator_geometria <= 2):
    ds = 1.0  # dimensão dos lados do hexagono maior 2D
    dx = ds # tamanho da célula na direção x
    dy = ds   # tamanho da célula na direção y
    dz = ds
    Lx = (Nx)*dx  # comprimento na direção x
    Ly = (Ny)*dy  # comprimento na direção y
    Lz = (Nz-1)*dz  # comprimento na direção z

#*********************************************
tipo_1d = 0 # grade 1D auxiliar
tipo_3d = 3 # grade 3D

# calculando dimensões das matrizes para os campos E e H
ix = Nx/2
rx = Nx%2
jy = Ny/2
ry = Ny%2
jy2 = (Ny-1)/2
ry2 = (Ny-1)%2

PEZ = (ix + rx)*(jy + ry) + (ix)*(jy)
PH2 = (ix)*(jy + ry) + (ix + rx)*(jy)
PH1 = (ix)*(jy2 + ry2) + (ix)*(jy2)
PH3 = PH1
#print 'PEZ= ',PEZ

# matrizes para os gráficos de Ez nos planos xy e xz
ez_pxy = np.zeros((Nx,Ny)) # modificado para YEE 3D
ez_pxz = np.zeros((Nx,Nz-1))

ez_pxy2 = np.zeros((Ny,Nx)) # transposta de ez_pxy
```

```
ez_pxz2 = np.zeros((Nz-1,Nx)) # transposta de ez_pxz

#************************************
px = Nx/2 + 1 # posição da fonte ricker
py = Ny/2 + 1
pz = Nz/2 + 1
pcentro = p2b(px,py)
#*****************************************

# textos para gráficos
texto2 =  'dim='+str(Nx)+'x'+str(Ny)+'x'+str(Nz)+\
          '  ppw='+str(ppw)+'  CFL='+str(CFL)+\
          ' dx=dy=dz='+str(ds) + ' CPML='+ str(nxx)
if tipo_fonte == 0:
   texto1 = u'p_ricker '
else:
   texto1 = u'senóide '
if HARD_SOFT == 0:
   texto1 = texto1 + 'hard '
else:
   texto1 = texto1 + 'soft '
if ATIVANDO_DIPOLO_1 == 1 :
   texto3 = '  dipolo '+str(compdipolo)
else:
   texto3 = ''
textof = texto1 + texto2 + texto3
#************************************

# testando a fonte sozinha no tempo
ez_teste = np.zeros((tempomax))
tempo_teste = np.zeros((tempomax))

#******************************

# INICIO DOS CÁLCULOS PARA GRADE HEXAGONAL 3D

print u'\nINICIO FDTD'
print

# inicio loop do tempo

for tempo in range(0,tempomax):

   tempo_teste[tempo] = tempo
   ez_teste[tempo] = fonte_1(tempo,0.0,tipo_fonte,fator_at)

   #print '\n',tempo+1,'/',tempomax

# final loop do tempo
#************************************************

# LEITURA DAS MATRIZES PARA PLANOS XY E XZ
arquivo4 = open(arqsaida1,"r")
#linha = None
for i in range(0,Nx): # modificado para YEE 3D
```

```
    for j in range(0,Ny):
       linha = arquivo4.readline() #funcionou
       ez_pxy[i,j] = float(linha)
arquivo4.close()

arquivo5 = open(arqsaida2,"r")
#linha = None
for i in range(0,Nx): # modificado para YEE 3D
    for j in range(0,Nz-1):
       linha = arquivo5.readline() #funcionou
       ez_pxz[i,j] = float(linha)
arquivo5.close()

#*************************************************
# LEITURA DOS ARQUIVOS PARA MEDIR REFLEXAO NA CPML (MÉTODO YEE 3D)
eza0 = np.zeros((tempomax))
eza1 = np.zeros((tempomax))
erro_eza =  np.zeros((tempomax))
ezb0 = np.zeros((tempomax))
ezb1 = np.zeros((tempomax))
erro_ezb =  np.zeros((tempomax))
ezc0 = np.zeros((tempomax))
ezc1 = np.zeros((tempomax))
erro_ezc =  np.zeros((tempomax))
ezd0 = np.zeros((tempomax))
ezd1 = np.zeros((tempomax))
erro_ezd =  np.zeros((tempomax))

# LEITURA PONTO A
arquivo4 = open(arqsaida3,"r")
for i in range(0,tempomax):
    linha = arquivo4.readline() #funcionou
    eza0[i]= float(linha)
arquivo4.close()
arquivo4 = open(arqsaida3b,"r")
for i in range(0,tempomax):
    linha = arquivo4.readline() #funcionou
    eza1[i]= float(linha)
arquivo4.close()
# LEITURA PONTO B
arquivo4 = open(arqsaida4,"r")
for i in range(0,tempomax):
    linha = arquivo4.readline() #funcionou
    ezb0[i]= float(linha)
arquivo4.close()
arquivo4 = open(arqsaida4b,"r")
for i in range(0,tempomax):
    linha = arquivo4.readline() #funcionou
    ezb1[i]= float(linha)
arquivo4.close()
# LEITURA PONTO C
arquivo4 = open(arqsaida5,"r")
for i in range(0,tempomax):
    linha = arquivo4.readline() #funcionou
    ezc0[i]= float(linha)
```

```
arquivo4.close()
arquivo4 = open(arqsaida5b,"r")
for i in range(0,tempomax):
    linha = arquivo4.readline() #funcionou
    ezc1[i]= float(linha)
arquivo4.close()
# LEITURA PONTO D
arquivo4 = open(arqsaida6,"r")
for i in range(0,tempomax):
    linha = arquivo4.readline() #funcionou
    ezd0[i]= float(linha)
arquivo4.close()
arquivo4 = open(arqsaida6b,"r")
for i in range(0,tempomax):
    linha = arquivo4.readline() #funcionou
    ezd1[i]= float(linha)
arquivo4.close()

# CALCULO COEFICIENTE DE REFLEXÃO
erroref_a = 0.0
nerros_a = 0
erroref_b = 0.0
nerros_b = 0
erroref_c = 0.0
nerros_c = 0
erroref_d = 0.0
nerros_d = 0

for i in range(0,tempomax):
   erro_eza[i] = np.abs((eza1[i] - eza0[i])/(eza0[i] + 1e-20))
   if erro_eza[i] > 0.0:
      erroref_a = erroref_a + erro_eza[i]
      nerros_a = nerros_a + 1
   erro_ezb[i] = np.abs((ezb1[i] - ezb0[i])/(ezb0[i] + 1e-20))
   if erro_ezb[i] > 0.0:
      erroref_b = erroref_b + erro_ezb[i]
      nerros_b = nerros_b + 1
   erro_ezc[i] = np.abs((ezc1[i] - ezc0[i])/(ezc0[i] + 1e-20))
   if erro_ezc[i] > 0.0:
      erroref_c = erroref_c + erro_ezc[i]
      nerros_c = nerros_c + 1
   erro_ezd[i] = np.abs((ezd1[i] - ezd0[i])/(ezd0[i] + 1e-29))
   if erro_ezd[i] > 0.0:
      erroref_d = erroref_d + erro_ezd[i]
      nerros_d = nerros_d + 1
erroref_medio_a = erroref_a/nerros_a
erroref_medio_b = erroref_b/nerros_b
erroref_medio_c = erroref_c/nerros_c
erroref_medio_d = erroref_d/nerros_d
print 'MEDIDAS DE REFLEXÃO (MÉTODO YEE 3D):'
print 'ESPESSURA_CPML_X = ',nxx
print 'REFLEXAO = ',Reflexao
print 'me = ',me
print 'NUMERO_PASSOS = ',tempomax
print 'NUMERO_ERROS_A = ',nerros_a
```

```
print 'COEF_REFLEXAO_MEDIO_A = ',erroref_medio_a
print 'NUMERO_ERROS_B = ',nerros_b
print 'COEF_REFLEXAO_MEDIO_B = ',erroref_medio_b
print 'NUMERO_ERROS_C = ',nerros_c
print 'COEF_REFLEXAO_MEDIO_C = ',erroref_medio_c
print 'NUMERO_ERROS_D = ',nerros_d
print 'COEF_REFLEXAO_MEDIO_D = ',erroref_medio_d
print
print

#*****************************************

# gerando gráficos 1D a partir dos gráficos 2D
Nx2 = Nx - px
Ny2 = Ny - py
Nz2 = Nz - 1 - pz
ez_x1 = np.zeros((Nx2+1))
ez_y1 = np.zeros((Ny2+1))
ez_x2 = np.zeros((Nx2+1))
ez_z2 = np.zeros((Nz2+1))
x = np.zeros((Nx2+1))
y = np.zeros((Ny2+1))
z = np.zeros((Nz2+1))

for i in range(0,Nx2):
    x[i] = i*dx
    ez_x1[i] = ez_pxy[px+i,py] # plano xy
    ez_x2[i] = ez_pxz[px+i,pz] # plano xz

for j in range(0,Ny2):
    y[j] = j*dy
    ez_y1[j] = ez_pxy[px,py+j] # plano xy

for k in range(0,Nz2):
    z[k] = k*dz
    ez_z2[k] = ez_pxz[px,pz+k] # plano xz


if Nx < Ny: # diagonal do plano xy
    Nd = Nx2
else:
    Nd = Ny2
ez_d1 = np.zeros((Nd+1))
s = np.zeros((Nd+1))
ds3 = np.sqrt(dx**2 + dy**2) # alterando para Yee 3D
for k in range(0,Nd):
    s[k] = k*ds3
    ez_d1[k] = ez_pxy[px+k,py+k]

if Nx < Nz: # diagonal do plano xz
    Nd2 = Nx2
else:
    Nd2 = Nz2
ez_d2 = np.zeros((Nd2+1))
s2 = np.zeros((Nd2+1))
```

```
ds2 = np.sqrt(dx**2 + dz**2)
for k in range(0,Nd2):
    s2[k] = k*ds2

    ez_d2[k] = ez_pxz[px+k,pz+k]

#*****************************************
#************************************************

ez_pxy2 = ez_pxy.T # transposta para corrigir imagem
ez_pxz2 = ez_pxz.T # transposta para corrigir imagem

#*******************************************************

# PLOTANDO CAMPOS Ez PARA O MODO 3D
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

# nearest ou bilinear - funcionou
# cm.jet ou cm.RdYlGn ou cm.winter ou cm.hot
# ficou imagem correta com shape=None e com extent=[0,Lx,0,Ly]
# mas shape inverteu eixos x e y
# no entanto usando a transposta de ez a imagem ficou com eixos corretos

# GRÁFICOS 2D
if MOSTRAR_SOMENTE_GRAFICOS_1D == 0:
    fs1 = 14   # xlabel e ylabel
    fs2 = 15   # title
    fse = 13   # numeros nos eixos x e y

    # plano xy
    plt.figure()
    im = plt.imshow(ez_pxy2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Ly], shape=None,
                vmax= nivel*abs(ez_pxy2).max(), vmin= -nivel*abs(ez_pxy2).max()  )

    plt.colorbar(im) # funcionou
    plt.title(u'YEE 3D: Ez (V/m) NO PLANO XY (T = '+str(tempomax)+' passos)')
    plt.xlabel(u'Lx (metros) ',fontsize = fs1)
    plt.ylabel(u'Ly (metros) ',fontsize = fs1)
    #plt.show()
    # posições dos pontos de medida de reflexão
    label((px+60)*dx,(py-0)*dy,u'*',20)
    label((px+60+8)*dx,(py-0)*dy,u'A',18)
    label((px+0)*dx,(py + 60 -0)*dy,u'*',20)
    label((px+8)*dx,(py + 60 -0)*dy,u'B',18)
    label((px+60)*dx,(py + 60 -0)*dy,u'*',20)
    label((px+60+8)*dx,(py + 60 -0)*dy,u'C',18)

    # plano xz
    plt.figure()
    im2 = plt.imshow(ez_pxz2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Lz], shape=None,
```

```
                vmax= nivel*abs(ez_pxz2).max(), vmin= -nivel*abs(ez_pxz2).max()  )

    plt.colorbar(im2) # funcionou
    plt.title(u'YEE 3D: Ez (V/m) NO PLANO XZ (T = '+str(tempomax)+' passos)')
    plt.xlabel(u'Lx (metros) ',fontsize = fs1)
    plt.ylabel(u'Lz (metros) ',fontsize = fs1)
    #plt.show()
    # posições dos pontos de medida de reflexão
    label(px*dx,(pz  + 60)*dz,u'*',20)
    label((px+ 8)*dx,(pz  + 60)*dz,u'D',18)


#*********************************************
# GRÁFICOS 1D
# no plano xy
plt.figure()
plt.plot(x,ez_x1,'r')
plt.grid()
plt.xlabel(u'Eixos x/y/d, '+ textof)
plt.ylabel(u'Nível')
plt.title('Campos 1D: Ez plano xy (TEMPO = '+str(tempomax)+' passos)')
plt.hold(True) # permite sobrepor vários
               # gráficos com dimensões diferentes
plt.plot(y,ez_y1,'b')
plt.plot(s,ez_d1,'k-')
texto1 = u'ez_x1 ( 0°)'
texto2 = u'ez_y1 (90°)'
texto3 = u'ez_d1 (45°)'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)

#plt.show()

# no plano xz
plt.figure()
plt.plot(x,ez_x2,'r')
plt.grid()
plt.xlabel(u'Eixos x/z/d, '+ textof)
plt.ylabel(u'Nível')
plt.title('Campos 1D: Ez plano xz (TEMPO = '+str(tempomax)+' passos)')
plt.hold(True) # permite sobrepor vários
               # gráficos com dimensões diferentes

plt.plot(z,ez_z2,'b-')
plt.plot(s2,ez_d2,'k')
texto1 = u'ez_x2 ( 0°)'
texto2 = u'ez_z2 (90°)'
texto3 = u'ez_d2 (45°)'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)
#plt.show()

#*************************************************
if TESTE_REFLEXAO == 1: # PARA MÉTODO YEE 3D
    # PONTO A
```

```
plt.figure()
plt.plot(tempo_teste,eza0,'r',tempo_teste,eza1,'k')
plt.grid()
plt.xlabel(u'Eixo Tempo, '+ textof)
plt.ylabel(u'Nível')
plt.title(u'Ponto A: campos Ez ( T = '+str(tempomax)+\
        ' passos, FATOR_AT = '+str(fator_at)+' )' )
texto1 = u'sem CPML'
texto2 = u'com CPML'
plt.legend((texto1,texto2),loc = 'upper left')

plt.figure()
plt.plot(tempo_teste,erro_eza,'k')
plt.grid()
plt.xlabel(u'Eixo Tempo, '+ textof)
plt.ylabel(u'Nível')
plt.title(u'Ponto A: Fator de reflexão ( T = '+str(tempomax)+\
        ' passos, FATOR_AT = '+str(fator_at)+' )' )
# PONTO B
plt.figure()
plt.plot(tempo_teste,ezb0,'r',tempo_teste,ezb1,'k')
plt.grid()
plt.xlabel(u'Eixo Tempo, '+ textof)
plt.ylabel(u'Nível')
plt.title(u'Ponto B: os campos Ez ( TEMPO = '+str(tempomax)+\
        ' passos, FATOR_AT = '+str(fator_at)+' )' )
texto1 = u'sem CPML'
texto2 = u'com CPML'
plt.legend((texto1,texto2),loc = 'upper left')
plt.figure()
plt.plot(tempo_teste,erro_ezb,'k')
plt.grid()
plt.xlabel(u'Eixo Tempo, '+ textof)
plt.ylabel(u'Nível')
plt.title(u'Ponto B: Fator de reflexão ( T = '+str(tempomax)+\
        ' passos, FATOR_AT = '+str(fator_at)+' )' )
# PONTO C
plt.figure()
plt.plot(tempo_teste,ezc0,'r',tempo_teste,ezc1,'k')
plt.grid()
plt.xlabel(u'Eixo Tempo, '+ textof)
plt.ylabel(u'Nível')
plt.title(u'Ponto C: os campos Ez ( TEMPO = '+str(tempomax)+\
        ' passos, FATOR_AT = '+str(fator_at)+' )' )
texto1 = u'sem CPML'
texto2 = u'com CPML'
plt.legend((texto1,texto2),loc = 'upper left')

plt.figure()
plt.plot(tempo_teste,erro_ezc,'k')
plt.grid()
plt.xlabel(u'Eixo Tempo, '+ textof)
plt.ylabel(u'Nível')
plt.title(u'Ponto C: Fator de reflexão ( T = '+str(tempomax)+\
        ' passos, FATOR_AT = '+str(fator_at)+' )' )
```

```
    # PONTO D
    plt.figure()
    plt.plot(tempo_teste,ezd0,'r',tempo_teste,ezd1,'k')
    plt.grid()
    plt.xlabel(u'Eixo Tempo, '+ textof)
    plt.ylabel(u'Nível')
    plt.title(u'Ponto D: campos Ez ( T = '+str(tempomax)+\
            ' passos, FATOR_AT = '+str(fator_at)+' )' )
    texto1 = u'sem CPML'
    texto2 = u'com CPML'
    plt.legend((texto1,texto2),loc = 'upper left')

    plt.figure()
    plt.plot(tempo_teste,erro_ezd,'k')
    plt.grid()
    plt.xlabel(u'Eixo Tempo, '+ textof)
    plt.ylabel(u'Nível')
    plt.title(u'Ponto D: Fator de reflexão ( T = '+str(tempomax)+\
            ' passos, FATOR_AT = '+str(fator_at)+' )' )
#**********************************************************


if MOSTRAR_SOMENTE_GRAFICOS_1D == 1:
    # fonte em função do tempo
    plt.figure()
    plt.plot(tempo_teste,ez_teste,'k')
    plt.grid()
    plt.xlabel(u'Eixo Tempo, '+ textof)
    plt.ylabel(u'Nível')
    plt.title('Fonte para campo Ez ( TEMPO = '+str(tempomax)+\
            ' passos, FATOR_AT = '+str(fator_at)+' )' )
    #plt.show()


plt.show()

# FINAL DO PROGRAMA
## FINAL DO APÊNDICE 7
################################################################
```

APÊNDICE 8. PROGRAMAS PARA MEDIR O ESPALHAMENTO DE CAMPO DE PLACA RETANGULAR METÁLICA PARA AMBOS OS MÉTODOS FDTD

ESTE APÊNDICE CONTÉM 2 ARQUIVOS

ARQUIVO 1: MEDIDAS DE CAMPO ESPALHADO PARA PLACA RETANGULAR USANDO O MÉTODO FDTD YEE

```
# -*- coding: cp1252 -*-
#            cp1252 ou utf-8
# programa programa_fdtd_3d_h_2c_visualizacao_1d.py em python 2.7.3
# derivado do programa: programa_fdtd_3d_h_2c_visualizacao_1c.py
# AUTOR: MARINOEL JOAQUIM

#*********************************************************
# BIBLIOTECAS NECESSÁRIAS

import numpy as np
import time
import scipy.special as sp
#*********************************************************
tempo1 = time.time()

# INICIO DAS FUNÇÕES


# FUNÇÃO PARA FONTE DO TIPO PULSO RICKER OU SENOIDE
def fonte_1(tempo2,posicao,tipo_fonte,fator_at):
    if tipo_fonte == 0: # pulso ricker
        arg = np.pi*( (cdtds*tempo2 - posicao)/ppw - 1.0 )
        arg2 = arg*arg
        funcao =(1.0 - 2.0*arg2)*np.exp(-arg2)

    else:  # senóide com amortecimento inicial
        periodo = ppw/cdtds # periodo da senóide
                        # ou tempo para o pico do pulso ricker
        cte_tempo = fator_at*periodo
        funcao = ( 1.0 - np.exp(-1.0*tempo2/cte_tempo) )*\
                 np.sin(2*np.pi*tempo2*cdtds/ppw)
    return funcao
#*********************************************************

# Função para criar grafico na forma polar
def gera_polar1(Max_polarx,porcentagemdepx, Einc, dw, dw2, pw, pw2,
                ez_puv,fator_raio):
    tamanhox = int(porcentagemdepx*pw2) # parte inicial não usada
    ez_x1polar = np.zeros((tamanhox))   # 0 a tamanhox
    ez_x1polar[:] = ez_pxy[:tamanhox,py]
    nxp = ez_x1polar.argmax()  # valor reduzido da dimensão central no eixo x
    if nxp <=10: nxp = 10
    if  MAXPOLAR_AUTOMATICO == 0:
        n = int(round((pw2 - nxp - 1)*dw2/dw)) # raio da medidas = n*dw
```

```
    else:
        n = int((pw2 - Max_polarx)*dw2/dw)

    dw = 1.0*dw # referencia principal dz ou dy: 0 < k < (n+1)
    dw2 = 1.0*dw2 # referencia dependente dx: 0 < m[k] < (n+1)*dw/dw2
    raio = n*dw
    #raio2 = int(raio)
    teta = np.zeros((4*n+1))
    ezp = np.zeros((4*n+1))
    a = np.zeros((n+1))
    b = np.zeros((n+1))
    m = np.zeros((n+1))
    tetar = np.zeros((2*n+1))
    ezpr = np.zeros((2*n+1))
    for k in range(0,n+1):
        a[k] = k*dw
        b[k] = np.sqrt(raio**2 - a[k]**2)
        m[k] = int(round(b[k]/dw2)) # modificado
        #teta[k] = np.arctan2(a[k],b[k])
        teta[k] = np.arctan2(a[k],m[k]*dw2)
        ezp[k] = abs(ez_puv[pw2 + m[k],pw + k])
    for k in range(0,n+1):
        teta[k+n] = np.pi - teta[n-k]
        ezp[k+n] = abs(ez_puv[pw2 - m[n-k],pw + n - k])
        teta[k+2*n] = np.pi + teta[k]
        ezp[k+2*n] = abs(ez_puv[pw2 - m[k],pw - k])
        teta[k+3*n] = 2*np.pi - teta[n-k]
        ezp[k+3*n] = abs(ez_puv[pw2 + m[n-k],pw - n + k])
    for k in range(0,2*n+1): # novas variaveis para medir espalhamento (RCS)
        tetar[k] = (teta[k+n] - np.pi/2.0)*180.0/np.pi
        ezpr[k] = ezp[k+n]
        comp = 1.0*dx*ppw
        raioesf = comp
        einc2 = Einc*(raio - raioesf )/(raio - raioesf +\
                        raioesf*(1.0 + np.cos(teta[k])))
        #einc2 = Einc*(np.cos(teta[k])*np.cos(fix) - np.sin(fix))
        einc2 = Einc
        raio3 = raio
        ezpr[k] = ezpr[k]*ezpr[k]*4.0*np.pi*raio3*raio3/((comp*einc2)**2)
        ezpr[k] = 10*np.log10(ezpr[k])
    return raio, teta, ezp, tetar, ezpr
    # Final função para gráfico polar

#*******************************************
# FINAL DAS FUNÇÕES
#*******************************************************
#*******************************************************

# INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS
Nx = 81 # numero de pontos na direção x
Ny = 141  # numero de pontos na direção y
Nz = 71 # numero de pontos na direção z
ppw = 15 # numero de pontos por comprimento de onda
tempomax = 40
```

```
tipo_fonte = 1  # 0=pulso ricker ou 1=senoide
HARD_SOFT  = 1   # fonte: 0=hard ou 1=soft

rdsdz = 1.0 #np.sqrt(1.0) # razão ds/dz
fator_comp = 1.4 # em comprimentos de onda para dipolo_1

arqsaida1 = "ez_pxy_1.txt"
arqsaida2 = "ez_pxz_2.txt"
fator_at = 0.5

# FINAL DAS PRINCIPAIS VARIÁVEIS

#************************************************

# VARIÁVEIS DE CONTROLE DOS GRÁFICOS

USAR_ESTES_ARQUIVOS = 0 # usar 1 ou 2, ou 0

MOSTRAR_SOMENTE_GRAFICOS_1D = 0 # 1= graficos 1D, 0 = mostra tudo

ATIVA_POLAR = 1 # 0 = desativa 1 = ativa diagrama polar
nivel = 1e-1  # controla nível máximo graficos 2D

SAIDA_ARQUIVOS_PLACA_YEE = 1  # 0 = desativa 1 = ativa
MAXPOLAR_AUTOMATICO = 1 # automatico=0 ou usar Max_polarx
Max_polarx = 32 # posição do máximo no eixo x (0 a Max_polarx)
Max_polarx2 = 32
fator_raio = .5
eincidente = .86  #.48
eincidente2 = .86  #.48
largura_a = 1.0
largura_b = 2.0
porcentagemdepx = 0.3# para obter o maxpolar automático
#*****************************************************


# LEITURA DE ARQUIVO DE ENTRADA
if USAR_ESTES_ARQUIVOS == 0:
    arquivo3 = open("entrada4.txt","r")

if USAR_ESTES_ARQUIVOS == 1:
    arquivo3 = open("entrada_ta1.txt","r")

if USAR_ESTES_ARQUIVOS == 2:
    arquivo3 = open("entrada_ta2.txt","r")

Nx = int(arquivo3.readline())#funcionou
Ny =  int(arquivo3.readline())
Nz =  int(arquivo3.readline())
ppw =  int(arquivo3.readline())
tempomax =  int(arquivo3.readline())
tipo_fonte =  int(arquivo3.readline())
HARD_SOFT =  int(arquivo3.readline())
ds =  float(arquivo3.readline())
rdsdz =  float(arquivo3.readline())
```

```
fator_cdtds = float(arquivo3.readline())
fator_at =  float(arquivo3.readline())
delta_passo = int(arquivo3.readline())
tempomax2 =  int(arquivo3.readline())
tempomax3 =  int(arquivo3.readline())
CPML =  int(arquivo3.readline())
nxx =  int(arquivo3.readline())
Reflexao = float(arquivo3.readline())
Reflexao2 = float(arquivo3.readline())
me = float(arquivo3.readline())
me2 = float(arquivo3.readline())
numero_materiais = int(arquivo3.readline())
numero_objetos = int(arquivo3.readline())
arquivo3.close()

fator_geometria = 2
# RETIRANDO "\n" DAS STRINGS
txx1 = list(arqsaida1)
txx1.pop()
arqsaida1 = "".join(txx1)

txx2 = list(arqsaida2)
txx2.pop()
arqsaida2 = "".join(txx2)

if USAR_ESTES_ARQUIVOS == 0:
   arqsaida1 = "ez_pxy_yee1_420.txt"
   arqsaida2 = "ez_pxz_yee2_420.txt"

if USAR_ESTES_ARQUIVOS == 1:
   arqsaida1 = "ez_pxy_1_ta1.txt"
   arqsaida2 = "ez_pxz_2_ta1.txt"

if USAR_ESTES_ARQUIVOS == 2:
   arqsaida1 = "ez_pxy_1_ta2.txt"
   arqsaida2 = "ez_pxz_2_ta2.txt"

cdtds = 1.0/np.sqrt(fator_cdtds) # número de Courant

####################################################
eo = 1.0e-9/(36*np.pi)
uo = np.pi*4e-7
co = 1.0/np.sqrt(uo*eo)
print 'co =',co

# IMPRIME VALORES OBTIDOS DO ARQUIVO DE ENTRADA
print 'Nx =',Nx
print 'Ny =',Ny
print 'Nz =',Nz
print 'ppw =',ppw
print 'tempomax =',tempomax
print 'tipo_fonte =',tipo_fonte
print 'HARD_SOFT =',HARD_SOFT
print 'ATIVANDO_DIPOLO_1 =',ATIVANDO_DIPOLO_1
print 'ATIVANDO_DIPOLO_2 =',ATIVANDO_DIPOLO_2
```

```
print 'rdsdz =', rdsdz
print 'cdtds =', cdtds
print 'fator comprimento dipolo  =',fator_comp
print 'fator atenuação (senóide) =',fator_at
#print 'fator geometria (1, 2, 3) =',fator_geometria
#print 'arqsaida1 =',arqsaida1
#print 'arqsaida2 =',arqsaida2
# FINAL DA LEITURA

#************************************

razaodsdz = round(rdsdz,2) # para gráfico
CFL = int(cdtds*100)/100.0 # número de Courant para gráfico

ativo = 1  # tipo dipolo_1
passivo = 0
comprimento_dipolo = int(fator_comp*rdsdz*ppw)
compdipolo = int(fator_comp*100)/100.0 # para gráfico
imp0 = 120.0*np.pi

#************************************************

if (fator_geometria <= 2):
   ds = 1.0  # dimensão dos lados do hexagono maior 2D
   dx = ds # tamanho da célula na direção x
   dy = ds   # tamanho da célula na direção y
   dz = ds
   Lx = (Nx)*dx  # comprimento na direção x
   Ly = (Ny)*dy  # comprimento na direção y
   Lz = (Nz-1)*dz  # comprimento na direção z
#*******************************************************

tipo_1d = 0 # grade 1D auxiliar
tipo_3d = 3 # grade 3D

# calculando dimensões das matrizes para os campos E e H
ix = Nx/2
rx = Nx%2
jy = Ny/2
ry = Ny%2
jy2 = (Ny-1)/2
ry2 = (Ny-1)%2

PEZ = (ix + rx)*(jy + ry) + (ix)*(jy)
PH2 = (ix)*(jy + ry) + (ix + rx)*(jy)
PH1 = (ix)*(jy2 + ry2) + (ix)*(jy2)
PH3 = PH1
#print 'PEZ= ',PEZ

# matrizes para os gráficos de Ez nos planos xy e xz
ez_pxy = np.zeros((Nx,Ny)) # modificado para YEE 3D
ez_pxz = np.zeros((Nx,Nz-1))

ez_pxy2 = np.zeros((Ny,Nx)) # transposta de ez_pxy
ez_pxz2 = np.zeros((Nz-1,Nx)) # transposta de ez_pxz
```

```
#************************************
px = Nx/2 + 1 # posição da fonte ricker
py = Ny/2 + 1
pz = Nz/2 + 1

#*****************************************

# textos para gráficos
texto2 =  'dim='+str(Nx)+'x'+str(Ny)+'x'+str(Nz)+\
          '  ppw='+str(ppw)+'  CFL='+str(CFL)+\
          ' dx=dy=dz='+str(ds) + ' PML='+ str(int(nxx*dx)) + ' m'
if tipo_fonte == 0:
   texto1 = u'pulso '
else:
   texto1 = u'seno '
if HARD_SOFT == 0:
   texto1 = texto1 + 'hard '
else:
   texto1 = texto1 + 'soft '
if ATIVANDO_DIPOLO_1 == 1 :
   texto3 = '  dipolo '+str(compdipolo)
else:
   texto3 = ''
textof = texto1 + texto2 + texto3
#************************************

# testando a fonte sozinha no tempo
ez_teste = np.zeros((tempomax))
tempo_teste = np.zeros((tempomax))

#******************************

# INICIO DOS CÁLCULOS PARA GRADE HEXAGONAL 3D

print u'\nINICIO FDTD'
print

# inicio loop do tempo

for tempo in range(0,tempomax):

   tempo_teste[tempo] = tempo
   ez_teste[tempo] = fonte_1(tempo,0.0,tipo_fonte,fator_at)

   #print '\n',tempo+1,'/',tempomax

# final loop do tempo
#*********************************************

#*****************************************

# LEITURA DAS MATRIZES PARA PLANOS XY E XZ
arquivo4 = open(arqsaida1,"r")
#linha = None
```

```
for i in range(0,Nx): # modificado para YEE 3D
    for j in range(0,Ny):
       linha = arquivo4.readline() #funcionou
       ez_pxy[i,j] = float(linha)
arquivo4.close()

arquivo5 = open(arqsaida2,"r")
#linha = None
for i in range(0,Nx): # modificado para YEE 3D
    for j in range(0,Nz-1):
       linha = arquivo5.readline() #funcionou
       ez_pxz[i,j] = float(linha)
arquivo5.close()

#************************************************
#******************************************
fx = 1 # funcionou com 0 e 1
# gerando gráficos 1D a partir dos gráficos 2D
Nx2 = Nx - fx*px
Ny2 = Ny - fx*py
Nz2 = Nz - 1 - fx*pz
ez_x1 = np.zeros((Nx2+1))
ez_y1 = np.zeros((Ny2+1))
ez_x2 = np.zeros((Nx2+1))
ez_z2 = np.zeros((Nz2+1))
x = np.zeros((Nx2+1))
y = np.zeros((Ny2+1))
z = np.zeros((Nz2+1))

for i in range(0,Nx2):
   x[i] = i*dx
   ez_x1[i] = ez_pxy[fx*px+i,py] # plano xy
   ez_x2[i] = ez_pxz[fx*px+i,pz] # plano xz

for j in range(0,Ny2):
   y[j] = j*dy
   ez_y1[j] = ez_pxy[px,fx*py+j] # plano xy

for k in range(0,Nz2):
   z[k] = k*dz
   ez_z2[k] = ez_pxz[px,fx*pz+k] # plano xz


if Nx < Ny: # diagonal do plano xy
   Nd = Nx2
else:
   Nd = Ny2
ez_d1 = np.zeros((Nd+1))
s = np.zeros((Nd+1))
ds3 = np.sqrt(dx**2 + dy**2) # alterando para Yee 3D
for k in range(0,Nd):
   s[k] = k*ds3
   ez_d1[k] = ez_pxy[fx*px+k,fx*py+k]

if Nx < Nz: # diagonal do plano xz
```

```
    Nd2 = Nx2
else:
    Nd2 = Nz2
ez_d2 = np.zeros((Nd2+1))
s2 = np.zeros((Nd2+1))

ds2 = np.sqrt(dx**2 + dz**2)
for k in range(0,Nd2):
    s2[k] = k*ds2

    ez_d2[k] = ez_pxz[fx*px+k,fx*pz+k]

#****************************************

# diagrama na forma polar no plano xz do campo ez
# modificações em 05/01/2016
pw = pz - 0
pw2 = px -0
dw = dz
dw2 = dx
Eincx = eincidente
Einc = Eincx #*4*np.pi
if ATIVA_POLAR == 1:
    # PLANO E (plano xz)
    raio1, teta, ezp, tetar, ezpr = gera_polar1(Max_polarx,porcentagemdepx,
                                Einc, dw, dw2, pw, pw2,ez_pxz,fator_raio)
    vmaxpolar = abs(ezp).max()
    ezp = ezp/vmaxpolar
    # PLANO H (plano xy)
    pw = py - 0
    pw2 = px - 0
    dw = dy
    dw2 = dx
    Eincx = eincidente2
    Einc = Eincx #*4*np.pi
    raio2, tetah, ezph, tetarh, ezprh = gera_polar1(Max_polarx2,porcentagemdepx,
                                Einc, dw, dw2, pw, pw2, ez_pxy,fator_raio)
    vmaxpolarh = abs(ezph).max()
    ezph = ezph/vmaxpolarh

    # diagrama teorico de dipolos
    N2 = 100
    teta2 = np.zeros((N2+1))
    ed2 = np.zeros((N2+1))
    dteta = 2*np.pi/N2
    for i in range(0,N2+1):
        if ATIVANDO_DIPOLO_1 == 1:
            teta2[i] = i*dteta
            rot = np.pi/2
            ed2[i] = abs((np.cos(np.pi*fator_comp*np.cos(teta2[i]+rot)) -\
                np.cos(np.pi*fator_comp))/(np.sin(teta2[i]+rot) + 1e-8))
            textop = texto3
        else:
            teta2[i] = i*dteta
            ed2[i] = abs(np.cos(teta2[i])**1)
```

```
        textop = u'dipolo curto'
textop1 = textop + u' fdtd' # para gráfico na forma polar
textop2 = textop + u' teórico'
#****************************************
# ESPALHAMENTO 3D DE PLACA RETANGULAR MODO TMz
Nteta2 = 500
fator_compa2 = largura_a    # gráfico rsc_h2x
fator_compb2 = largura_b    # gráfico rsc_e2x
ajuste_final = .98
fis = 1.*np.pi/2
atetai = 0.*np.pi/6
comp2 = 1.0 # usar 1.0: normaliza em relação ao comprimento de onda
atetas = np.zeros((Nteta2 + 1))
atetas2 = np.zeros((Nteta2 + 1))
rsc_e2 = np.zeros((Nteta2 + 1))
rsc_h2 = np.zeros((Nteta2 + 1))
atetas3 = np.zeros((2*Nteta2 + 1))
rsc_e3 = np.zeros((2*Nteta2 + 1))
rsc_h3 = np.zeros((2*Nteta2 + 1))
rsc_e2x = np.zeros((Nteta2 + 1))
rsc_h2x = np.zeros((Nteta2 + 1))

a2 = fator_compa2*comp2
b2 = fator_compb2*comp2
b = 2.0*np.pi/comp2
dteta2 = ajuste_final*np.pi/(2.0*Nteta2)
# PLANO E (plano xz)
for i in range(0,Nteta2+1):
    atetas[i] = i*dteta2
    atetas2[i] = i*dteta2*180/np.pi
    stetas2 = np.sin(atetas[i])*np.sin(atetas[i])
    ctetas2 = np.cos(atetas[i])*np.cos(atetas[i])
    sfis2 = np.sin(fis)*np.sin(fis)
    cfis2 = np.cos(fis)*np.cos(fis)
    ctetai2 = np.cos(atetai)*np.cos(atetai)
    Xp = (b*a2/2.0)*np.sin(atetas[i])*np.cos(fis)
    Yp = (b*b2/2.0)*(np.sin(atetas[i])*np.sin(fis) - np.sin(atetai))
    sinX2 = (np.sin(Xp)/(Xp + .0e-29))**2
    sinY2 = (np.sin(Yp)/(Yp + .0e-29))**2
    ab2 = (a2*b2/comp2)**2
    rsc_e2x[i] = 4.0*np.pi*ab2*(ctetai2*(ctetas2*cfis2 + sfis2))*sinX2*sinY2
    rsc_e2x[i] = 10.0*np.log10(np.abs(rsc_e2x[i]))
# PLANO H (plano xy)
for i in range(0,Nteta2+1):
    atetas[i] = i*dteta2
    atetas2[i] = i*dteta2*180/np.pi
    stetas2 = np.sin(atetas[i])*np.sin(atetas[i])
    ctetas2 = np.cos(atetas[i])*np.cos(atetas[i])
    sfis2 = np.sin(fis)*np.sin(fis)
    cfis2 = np.cos(fis)*np.cos(fis)
    ctetai2 = np.cos(atetai)*np.cos(atetai)
    Xp = (b*b2/2.0)*np.sin(atetas[i])*np.cos(fis)
    Yp = (b*a2/2.0)*(np.sin(atetas[i])*np.sin(fis) - np.sin(atetai))
    sinX2 = (np.sin(Xp)/(Xp + .0e-29))**2
    sinY2 = (np.sin(Yp)/(Yp + .0e-29))**2
```

```
        ab2 = (a2*b2/comp2)**2
        rsc_h2x[i] = 4.0*np.pi*ab2*(ctetai2*(ctetas2*cfis2 + sfis2))*sinX2*sinY2
        rsc_h2x[i] = 10.0*np.log10(np.abs(rsc_h2x[i]))

    #*******************************************************
    # PLACA RETANGULAR COM FORMULAS SIMPLIFICADAS ??
    for i in range(0,Nteta2+1):
        atetas[i] = i*dteta2
        atetas2[i] = i*dteta2*180/np.pi
        stetas2 = np.sin(atetas[i])*np.sin(atetas[i])
        ctetas2 = np.cos(atetas[i])*np.cos(atetas[i])
        esintetab = np.sin(b*b2*np.sin(atetas[i]))/(b*b2*np.sin(atetas[i]))
        ab2 = (a2*b2/comp2)**2
        rsc_e2[i] = 4.0*np.pi*ab2*ctetas2*esintetab*esintetab
        rsc_e2[i] = 10.0*np.log10(np.abs(rsc_e2[i]))
        esintetaa = np.sin(b*a2*np.sin(atetas[i]))/(b*a2*np.sin(atetas[i]))
        ab2 = (a2*b2/comp2)**2
        rsc_h2[i] = 4.0*np.pi*ab2*ctetas2*esintetaa*esintetaa
        rsc_h2[i] = 10.0*np.log10(np.abs(rsc_h2[i]))
    for i in range(0,Nteta2+1):
        atetas3[i] = 90.0 - atetas2[Nteta2-i]
        rsc_e3[i] = rsc_e2x[Nteta2-i] # corrigido para equações exatas
        rsc_h3[i] = rsc_h2x[Nteta2-i]
        atetas3[Nteta2+i] = 90.0 + atetas2[i]
        rsc_e3[Nteta2+i] = rsc_e2x[i]
        rsc_h3[Nteta2+i] = rsc_h2x[i]
# final da placa retangular
#*************************************************************

# SAIDA DE ARQUIVOS PARA A PLACA YEE 3D QUE SERÃO USADOS NA
# COMPARAÇÃO COM A PLACA HEX 3D E PLACA TEÓRICA

if (SAIDA_ARQUIVOS_PLACA_YEE == 1):
    tempomax_yee = tempomax
    a_yee = largura_a
    b_yee = largura_b
    compE = len(ezpr)
    compH = len(ezprh)
    #print 'compE =' , compE
    dados_placa = [tempomax_yee,a_yee,b_yee, compE, compH]
    saida_dados = 'pyee_dados_placa_T' + str(tempomax) + '.txt'
    arquivox1 = open(saida_dados, "w")
    for i in range(0,len(dados_placa)):
        arquivox1.write("%f\n" % dados_placa[i])
    arquivox1.close()

    # PLANO E
    esf_planoE = 'pyee_planoE_T' + str(tempomax) + '.txt'
    arquivox1 = open(esf_planoE, 'w')
    for i in range(0,compE):
        arquivox1.write("%18.17f\n" % ezpr[i])
    arquivox1.close()

    esf_anguloE = 'pyee_anguloE_T' + str(tempomax) + '.txt'
    arquivox1 = open(esf_anguloE, 'w')
```

```
    for i in range(0,compE):
        arquivox1.write("%18.17f\n" % tetar[i])
    arquivox1.close()

    # PLANO H
    esf_planoH = 'pyee_planoH_T' + str(tempomax) + '.txt'
    arquivox1 = open(esf_planoH, 'w')
    for i in range(0,compH):
        arquivox1.write("%18.17f\n" % ezprh[i])
    arquivox1.close()

    esf_anguloH = 'pyee_anguloH_T' + str(tempomax) + '.txt'
    arquivox1 = open(esf_anguloH, 'w')
    for i in range(0,compH):
        arquivox1.write("%18.17f\n" % tetarh[i])
    arquivox1.close()
# FINAL SAIDA DOS ARQUIVOS DA ESFERA YEE 3D

#*****************************************************************

ez_pxy2 = ez_pxy.T # transposta para corrigir imagem
ez_pxz2 = ez_pxz.T # transposta para corrigir imagem

#*******************************************************


if ATIVA_POLAR == 1:
    print 'px = ',px
    print 'py = ',py
    print 'pz = ',pz
    print 'raio1_plano_E = ',raio1
    print 'raio2_plano_H = ',raio2
#print'\n'
print u'FINAL'

#****************************************

# PLOTANDO CAMPOS Ez PARA O MODO 3D
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

# nearest ou bilinear - funcionou
# cm.jet ou cm.RdYlGn ou cm.winter ou cm.hot
# ficou imagem correta com shape=None e com extent=[0,Lx,0,Ly]
# mas shape inverteu eixos x e y
# no entanto usando a transposta de ez a imagem ficou com eixos corretos

# GRÁFICOS 2D
if MOSTRAR_SOMENTE_GRAFICOS_1D == 0:
    fs1 = 14   # xlabel e ylabel
    fs2 = 15   # title
    fse = 13   # numeros nos eixos x e y

    # plano xy
```

```
    plt.figure()
    im = plt.imshow(ez_pxy2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Ly], shape=None,
                vmax= nivel*abs(ez_pxy2).max(), vmin= -nivel*abs(ez_pxy2).max()  )

    plt.colorbar(im) # funcionou
    plt.title(u'YEE 3D: Ez (V/m) NO PLANO XY (T = '+str(tempomax)+' passos)')
    plt.xlabel(u'Lx (metros)',fontsize = fs1)
    plt.ylabel(u'Ly (metros)',fontsize = fs1)
    #plt.show()

    # plano xz
    plt.figure()
    im2 = plt.imshow(ez_pxz2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Lz], shape=None,
                vmax= nivel*abs(ez_pxz2).max(), vmin= -nivel*abs(ez_pxz2).max()  )

    plt.colorbar(im2) # funcionou
    plt.title(u'YEE 3D: Ez (V/m) NO PLANO XZ (T = '+str(tempomax)+' passos)')
    plt.xlabel(u'Lx (metros)',fontsize = fs1)
    plt.ylabel(u'Lz (metros)',fontsize = fs1)
    #plt.show()

#*********************************************
# GRÁFICOS 1D
# no plano xy
plt.figure()
plt.plot(x,ez_x1,'r')
plt.grid()
plt.xlabel(u'Eixos x/y/d, '+ textof)
plt.ylabel(u'Nível')
plt.title('Campos 1D: Ez plano xy (TEMPO = '+str(tempomax)+' passos)')
plt.hold(True) # permite sobrepor vários
            # gráficos com dimensões diferentes
plt.plot(y,ez_y1,'b')
plt.plot(s,ez_d1,'k-')
texto1 = u'ez_x1 ( 0°)'
texto2 = u'ez_y1 (90°)'
texto3 = u'ez_d1 (45°)'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)

#plt.show()

# no plano xz
plt.figure()
plt.plot(x,ez_x2,'r')
plt.grid()
plt.xlabel(u'Eixos x/z/d, '+ textof)
plt.ylabel(u'Nível')
plt.title('Campos 1D: Ez plano xz (TEMPO = '+str(tempomax)+' passos)')
plt.hold(True) # permite sobrepor vários
            # gráficos com dimensões diferentes

plt.plot(z,ez_z2,'b-')
```

```
plt.plot(s2,ez_d2,'k')
texto1 = u'ez_x2 ( 0°)'
texto2 = u'ez_z2 (90°)'
texto3 = u'ez_d2 (45°)'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)
#plt.show()

#**************************************************
#**********************************************************

if MOSTRAR_SOMENTE_GRAFICOS_1D == 1:
    # fonte em função do tempo
    plt.figure()
    plt.plot(tempo_teste,ez_teste,'k')
    plt.grid()
    plt.xlabel(u'Eixo Tempo, '+ textof)
    plt.ylabel(u'Nível')
    plt.title('Fonte para campo Ez ( TEMPO = '+str(tempomax)+\
            ' passos, FATOR_AT = '+str(fator_at)+' )' )
    #plt.show()

# diagrama na forma polar e espalhamento de placa retangular
if ATIVA_POLAR == 1:
    # PLANO E (plano xz)
    plt.figure()
    plt.polar(teta,ezp,'r')
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
    plt.polar(teta2,ed2,'b')
    plt.title('Polar, raio ='+str(raio1)+' '+ textof)
    plt.legend((textop1,textop2),loc = 'lower right')
    plt.hold(False)
    # graficos para espalhamento da placa retangular
    plt.figure()
    plt.plot(tetar,ezpr,'k') # ou plt.semilogy
    plt.grid()
    plt.xlabel(u'Ângulo (graus)')
    plt.ylabel(u'RCS (dBsm)') # ou dBsw
    plt.title(u'Yee 3D: RCS (Plano E) da placa retangular (a = '+\
            str(a2) + r'$\lambda$; b = '+str(b2)+r'$\lambda$)')
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
    plt.plot(atetas3,rsc_e3,'r')
    #plt.plot(s,ez_d1,'k-')
    texto1 = u'Yee 3D'
    texto2 = u'Teórico'
    #texto3 = u'ez_d1 (30°)'
    plt.legend((texto1,texto2),loc = 'upper right')
    plt.hold(False)

    # PLANO H (plano xy)
    plt.figure()
    plt.polar(tetah,ezph,'r')
    plt.hold(True) # permite sobrepor vários
```

```
                # gráficos com dimensões diferentes
    plt.polar(teta2,ed2,'b')
    plt.title('Polar, raio ='+str(raio2)+' '+ textof)
    plt.legend((textop1,textop2),loc = 'lower right')
    plt.hold(False)
    # graficos para espalhamento da placa retangular
    plt.figure()
    plt.plot(tetarh,ezprh,'k') # ou plt.semilogy
    plt.grid()
    plt.xlabel(u'Ângulo (graus)')
    plt.ylabel(u'RCS (dBsm)') # ou dBsm
    plt.title(u'Yee 3D: RCS (Plano H) da placa retangular (a = '+\
            str(a2) + r'$\lambda$; b = '+str(b2)+r'$\lambda$)')
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
    plt.plot(atetas3,rsc_h3,'r')
    #plt.plot(s,ez_d1,'k-')
    texto1 = u'Yee 3D'
    texto2 = u'Teórico'
    #texto3 = u'ez_d1 (30°)'
    plt.legend((texto1,texto2),loc = 'upper right')
    plt.hold(False)

plt.show()

# FINAL DO PROGRAMA
#########################################################################
```

ARQUIVO 2: MEDIDAS DE CAMPO ESPALHADO PARA PLACA RETANGULAR USANDO O MÉTODO FDTD COM GRADE DE PRISMAS HEXAGONAIS E COMPARANDO COM OS RESULTADOS DO MÉTODO FDTD YEE (ARQUIVO ANTERIOR)

```
# -*- coding: cp1252 -*-
#           cp1252 ou utf-8
# programa programa_fdtd_3d_h_2c_visualizacao_1d.py em python 2.7.3
# derivado do programa: programa_fdtd_3d_h_2c_visualizacao_1c.py
# AUTOR: MARINOEL JOAQUIM

#*********************************************************
# BIBLIOTECAS NECESSÁRIAS

import numpy as np
import time
import scipy.special as sp
#*********************************************************
tempo1 = time.time()

# INICIO DAS FUNÇÕES

# Função para hz e h3
def p2a(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx-1)
```

```
        p2 = (p1 + 1)/2
        if (p2 > PH3):
            p2 = 0
    return p2

# Função para ez e e3
def p2b(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx)
        p2 = (p1 + 1)/2
        if (p2 > PEZ):
            p2 = 0
    return p2

# Função para h2 e e1
def p2c(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx)
        p2 = (p1)/2
        if (p2 > PH2):
            p2 = 0
    return p2

# Função para h1 e e2
def p2d(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx-1)
        p2 = (p1 + 1)/2
        if (p2 > PH1):
            p2 = 0
    return p2
#*************************************

# FUNÇÃO PARA FONTE DO TIPO PULSO RICKER OU SENOIDE
def fonte_1(tempo2,posicao,tipo_fonte,fator_at):
    if tipo_fonte == 0: # pulso ricker
        arg = np.pi*( (cdtds*tempo2 - posicao)/ppw - 1.0 )
        arg2 = arg*arg
        funcao =(1.0 - 2.0*arg2)*np.exp(-arg2)

    else:  # senóide com amortecimento inicial
        periodo = ppw/cdtds # periodo da senóide
                        # ou tempo para o pico do pulso ricker
        cte_tempo = fator_at*periodo
        funcao = ( 1.0 - np.exp(-1.0*tempo2/cte_tempo) )*\
                np.sin(2*np.pi*tempo2*cdtds/ppw)
    return funcao
#***********************************
```

```
# função para obter i,j a partir de p para campo ez
def ez_ij(p2):
    p1 = 2*p2 - 1
    j = p1/Nx + 1
    if (p1%Nx==0 and p1>=Nx):
        j = j - 1
    i = p1 - (j-1)*Nx
    return i,j
#*********************************************************

# Função de interpolação para gráfico de ez no plano xy
def pteste(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        i = 0
        j = 0
    return i, j

# Função de interpolação para gráfico de ez no plano xz
def pteste2(i,k):
    if ((i<=0) or (i>=Nx+1) or (k<=0) or (k>=Nz+1)):
        i = 0
        k = 0
    return i, k
#*********************************************************

# Função para criar grafico na forma polar
def gera_polar1(Max_polarx,porcentagemdepx, Einc, dw, dw2, pw, pw2,
                ez_puv,fator_raio):
    tamanhox = int(porcentagemdepx*pw2) # parte inicial não usada
    ez_x1polar = np.zeros((tamanhox))   # 0 a tamanhox
    ez_x1polar[:] = ez_pxy[:tamanhox,py]
    nxp = ez_x1polar.argmax()  # valor reduzido da dimensão central no eixo x
    if nxp <=10: nxp = 10
    if  MAXPOLAR_AUTOMATICO == 0:
        n = int(round((pw2 - nxp - 1)*dw2/dw)) # raio da medidas = n*dw
    else:
        n = int((pw2 - Max_polarx)*dw2/dw)

    dw = 1.0*dw # referencia principal dz ou dy: 0 < k < (n+1)
    dw2 = 1.0*dw2 # referencia dependente dx: 0 < m[k] < (n+1)*dw/dw2
    raio = n*dw
    #raio2 = int(raio)
    teta = np.zeros((4*n+1))
    ezp = np.zeros((4*n+1))
    a = np.zeros((n+1))
    b = np.zeros((n+1))
    m = np.zeros((n+1))
    tetar = np.zeros((2*n+1))
    ezpr = np.zeros((2*n+1))
    for k in range(0,n+1):
        a[k] = k*dw
        b[k] = np.sqrt(raio**2 - a[k]**2)
        m[k] = int(round(b[k]/dw2)) # modificado
        #teta[k] = np.arctan2(a[k],b[k])
        teta[k] = np.arctan2(a[k],m[k]*dw2)
```

```
        ezp[k] = abs(ez_puv[pw2 + m[k],pw + k])
    for k in range(0,n+1):
        teta[k+n] = np.pi - teta[n-k]
        ezp[k+n] = abs(ez_puv[pw2 - m[n-k],pw + n - k])
        teta[k+2*n] = np.pi + teta[k]
        ezp[k+2*n] = abs(ez_puv[pw2 - m[k],pw - k])
        teta[k+3*n] = 2*np.pi - teta[n-k]
        ezp[k+3*n] = abs(ez_puv[pw2 + m[n-k],pw - n + k])
    for k in range(0,2*n+1): # novas variaveis para medir espalhamento (RCS)
        tetar[k] = (teta[k+n] - np.pi/2.0)*180.0/np.pi
        ezpr[k] = ezp[k+n]
        comp = 1.0*ds*ppw # usado ds para Hex 3d
        raioesf = comp
        einc2 = Einc*(raio - raioesf )/(raio - raioesf +\
                        raioesf*(1.0 + np.cos(teta[k])))
        #einc2 = Einc*(np.cos(teta[k])*np.cos(fix) - np.sin(fix))
        einc2 = Einc
        raio3 = raio
        ezpr[k] = ezpr[k]*ezpr[k]*4.0*np.pi*raio3*raio3/((comp*einc2)**2)
        ezpr[k] = 10*np.log10(ezpr[k])
    return raio, teta, ezp, tetar, ezpr
    # Final função para gráfico polar

#*********************************************
def label(x,y, text,tam_fonte):
    y = y - 0.0 # shift y-value for label so that it's below the artist
    plt.text(x, y, text, ha="center", family='sans-serif',
            size=tam_fonte,fontweight ='bold',color='k')

# FINAL DAS FUNÇÕES
#***********************************************************
#***********************************************************

# INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS
Nx = 81 # numero de pontos na direção x
Ny = 141  # numero de pontos na direção y
Nz = 71 # numero de pontos na direção z
ppw = 15 # numero de pontos por comprimento de onda
tempomax = 40

tipo_fonte = 1  # 0=pulso ricker ou 1=senoide
HARD_SOFT  = 1   # fonte: 0=hard ou 1=soft

rdsdz = 1.0 #np.sqrt(1.0) # razão ds/dz
fator_comp = 1.4 # em comprimentos de onda para dipolo_1

arqsaida1 = "ez_pxy_1.txt"
arqsaida2 = "ez_pxz_2.txt"
fator_at = 0.5

# FINAL DAS PRINCIPAIS VARIÁVEIS

#************************************************

# VARIÁVEIS DE CONTROLE DOS GRÁFICOS
```

```
USAR_ESTES_ARQUIVOS = 0 # usar 1 ou 2, ou 0

MOSTRAR_SOMENTE_GRAFICOS_1D = 0 # 1= graficos 1D, 0 = mostra tudo

ATIVA_POLAR = 1 # 0 = desativa 1 = ativa diagrama polar
TESTE_REFLEXAO = 0 # 0 = desativa 1 = ativa
nivel = 1.0e-1 # controla nível máximo graficos 2D

ENTRADA_ARQUIVOS_PLACA_YEE = 1   # 0 = desativa 1 = ativa
MAXPOLAR_AUTOMATICO = 1 # automatico=0 ou usar 1:Max_polarx
Max_polarx = 54 # 54 ou 66 ou 44
 # posição do máximo no eixo x (0 a Max_polarx)
Max_polarx2 = 54
fator_raio = 0.5
eincidente = .86   #0.55      # plano E (plano xz)
eincidente2 = .86  #0.55      # plano H (plano xy)
largura_a = 1.0
largura_b = 2.0

porcentagemdepx = 0.5 # para obter o maxpolar automático
#*****************************************************

# LEITURA DE ARQUIVO DE ENTRADA
if USAR_ESTES_ARQUIVOS == 0:
    arquivo3 = open("entrada3.txt","r")

if USAR_ESTES_ARQUIVOS == 1:
    arquivo3 = open("entrada_ta1.txt","r")

if USAR_ESTES_ARQUIVOS == 2:
    arquivo3 = open("entrada_ta2.txt","r")

Nx = int(arquivo3.readline())#funcionou
Ny =  int(arquivo3.readline())
Nz =  int(arquivo3.readline())
ppw =  int(arquivo3.readline())
tempomax =  int(arquivo3.readline())
tipo_fonte =  int(arquivo3.readline())
HARD_SOFT =  int(arquivo3.readline())
ds =  float(arquivo3.readline())
rdsdz =  float(arquivo3.readline())
fator_cdtds = float(arquivo3.readline())
fator_at =  float(arquivo3.readline())
delta_passo = int(arquivo3.readline())
tempomax2 =  int(arquivo3.readline())
tempomax3 =  int(arquivo3.readline())
CPML =  int(arquivo3.readline())
nxx =  int(arquivo3.readline())
Reflexao = float(arquivo3.readline())
Reflexao2 = float(arquivo3.readline())
me = float(arquivo3.readline())
me2 = float(arquivo3.readline())
numero_materiais = int(arquivo3.readline())
numero_objetos = int(arquivo3.readline())
```

```
arquivo3.close()

fator_geometria = 2
# RETIRANDO "\n" DAS STRINGS
txx1 = list(arqsaida1)
txx1.pop()
arqsaida1 = "".join(txx1)

txx2 = list(arqsaida2)
txx2.pop()
arqsaida2 = "".join(txx2)

if USAR_ESTES_ARQUIVOS == 0:
   arqsaida1 = "ez_pxy_1_439.txt"
   arqsaida2 = "ez_pxz_2_439.txt"
   if TESTE_REFLEXAO == 1:
      arqsaida3  = "ez_A_nxx_20_CPML_0_Temp_257.txt"
      arqsaida3b = "ez_A_nxx_20_CPML_1_Temp_257.txt"
      arqsaida4  = "ez_B_nxx_20_CPML_0_Temp_257.txt"
      arqsaida4b = "ez_B_nxx_20_CPML_1_Temp_257.txt"
      arqsaida5  = "ez_C_nxx_20_CPML_0_Temp_257.txt"
      arqsaida5b = "ez_C_nxx_20_CPML_1_Temp_257.txt"
      arqsaida6  = "ez_D_nxx_20_CPML_0_Temp_257.txt"
      arqsaida6b = "ez_D_nxx_20_CPML_1_Temp_257.txt"

if USAR_ESTES_ARQUIVOS == 1:
   arqsaida1 = "ez_pxy_1_ta1.txt"
   arqsaida2 = "ez_pxz_2_ta1.txt"

if USAR_ESTES_ARQUIVOS == 2:
   arqsaida1 = "ez_pxy_1_ta2.txt"
   arqsaida2 = "ez_pxz_2_ta2.txt"

cdtds = 1.0/np.sqrt(fator_cdtds) # número de Courant

#####################################################
eo = 1.0e-9/(36*np.pi)
uo = np.pi*4e-7
co = 1.0/np.sqrt(uo*eo)
#print 'co =',co

# IMPRIME VALORES OBTIDOS DO ARQUIVO DE ENTRADA
print 'Nx =',Nx
print 'Ny =',Ny
print 'Nz =',Nz
print 'ppw =',ppw
print 'tempomax =',tempomax
print 'tipo_fonte =',tipo_fonte
print 'HARD_SOFT =',HARD_SOFT
print 'rdsdz =', rdsdz
print 'cdtds =', cdtds
#print 'fator comprimento dipolo  =',fator_comp
print 'fator atenuação (senóide) =',fator_at
#print 'fator geometria (1, 2, 3) =',fator_geometria
#print 'arqsaida1 =',arqsaida1
```

```
#print 'arqsaida2 =',arqsaida2
# FINAL DA LEITURA


#************************************

razaodsdz = round(rdsdz,2) # para gráfico
CFL = int(cdtds*100)/100.0 # número de Courant para gráfico


ativo = 1  # tipo dipolo_1
passivo = 0
comprimento_dipolo = int(fator_comp*rdsdz*ppw)
compdipolo = int(fator_comp*100)/100.0 # para gráfico
imp0 = 120.0*np.pi

#************************************************

if (fator_geometria <= 2):
    ds = 1.0  # dimensão dos lados do hexagono maior 2D
    dx = ds*np.cos(np.pi/6) # tamanho da célula na direção x
    dy = ds/2.0   # tamanho da célula na direção y
    dz = ds/rdsdz
    Lx = (Nx)*dx  # comprimento na direção x
    Ly = (Ny)*dy  # comprimento na direção y
    Lz = (Nz)*dz  # comprimento na direção z

#*****************************************************

tipo_1d = 0 # grade 1D auxiliar
tipo_3d = 3 # grade 3D

# calculando dimensões das matrizes para os campos E e H
ix = Nx/2
rx = Nx%2
jy = Ny/2
ry = Ny%2
jy2 = (Ny-1)/2
ry2 = (Ny-1)%2

PEZ = (ix + rx)*(jy + ry) + (ix)*(jy)
PH2 = (ix)*(jy + ry) + (ix + rx)*(jy)
PH1 = (ix)*(jy2 + ry2) + (ix)*(jy2)
PH3 = PH1
#print 'PEZ= ',PEZ

# campos

# matrizes para os gráficos de Ez nos planos xy e xz
ez_pxy = np.zeros((Nx+1,Ny+1))
ez_pxz = np.zeros((Nx+1,Nz+1))

ez_pxy2 = np.zeros((Ny+1,Nx+1)) # transposta de ez_pxy
ez_pxz2 = np.zeros((Nz+1,Nx+1)) # transposta de ez_pxz
```

```
#************************************
px = Nx/2 + 1 # posição da fonte ricker
py = Ny/2 + 1
pz = Nz/2 + 1
pcentro = p2b(px,py)
#*****************************************

# textos para gráficos
texto2 =  'dim='+str(Nx)+'x'+str(Ny)+'x'+str(Nz)+\
          '  ppw='+str(ppw)+'  CFL='+str(CFL)+\
          '  ds/dz='+str(razaodsdz) + ' PMLx='+\
             str(int(nxx*dx)) +' m'

if tipo_fonte == 0:
   texto1 = u'p_ricker '
else:
   texto1 = u'senóide '
if HARD_SOFT == 0:
   texto1 = texto1 + 'hard '
else:
   texto1 = texto1 + 'soft '
if ATIVANDO_DIPOLO_1 == 1 :
   texto3 = '  dipolo '+str(compdipolo)
else:
   texto3 = ''
textof = texto1 + texto2 + texto3
#************************************

# testando a fonte sozinha no tempo
ez_teste = np.zeros((tempomax))
tempo_teste = np.zeros((tempomax))

#******************************

# INICIO DOS CÁLCULOS PARA GRADE HEXAGONAL 3D

print u'\nINICIO FDTD'
print

# inicio loop do tempo

for tempo in range(0,tempomax):

   tempo_teste[tempo] = tempo
   ez_teste[tempo] = fonte_1(tempo,0.0,tipo_fonte,fator_at)

   #print '\n',tempo+1,'/',tempomax

# final loop do tempo
#************************************************

#*****************************************

# LEITURA DAS MATRIZES PARA PLANOS XY E XZ
```

```
arquivo4 = open(arqsaida1,"r")
linha = None
for i in range(0,Nx+1):
    for j in range(0,Ny+1):
       linha = arquivo4.readline() #funcionou
       #print i
       #print j
       #print linha
       ez_pxy[i,j] = float(linha)
arquivo4.close()

arquivo5 = open(arqsaida2,"r")
#linha = None
for i in range(0,Nx+1):
    for j in range(0,Nz+1):
       linha = arquivo5.readline() #funcionou
       ez_pxz[i,j] = float(linha)
arquivo5.close()

#*************************************************
#*****************************************

# eliminando valores zero das matrizes para os gráficos de ez
for i in range(1,Nx+1): # plano xy
   for j in range(1,Ny+1):
       if (j%2==0 and i%2==1 and i<=Nx) or\
          (j%2==1 and i%2==0 and i<=Nx-1):
             inx, jnx = pteste(i,j+1)
             isx, jsx = pteste(i,j-1)
             ie, je   = pteste(i+1,j)
             iw, jw   = pteste(i-1,j)
             ez_pxy[i,j] = 0.25*(ez_pxy[inx,jnx] + ez_pxy[isx,jsx] +\
                         ez_pxy[ie,je] + ez_pxy[iw,jw])
j = py
for i in range(1,Nx+1):  # plano xz
   for k in range(1,Nz+1):
       if (j%2==0 and i%2==1 and i<=Nx) or\
          (j%2==1 and i%2==0 and i<=Nx-1):
             ie, ke   = pteste2(i+1,k)
             iw, kw   = pteste2(i-1,k)
             ez_pxz[i,k] = 0.5*(ez_pxz[ie,ke] + ez_pxz[iw,kw])

#*************************************************

# gerando gráficos 1D a partir dos gráficos 2D
Nx2 = Nx - px
Ny2 = Ny - py
Nz2 = Nz - pz
ez_x1 = np.zeros((Nx2+1))
ez_y1 = np.zeros((Ny2+1))
ez_x2 = np.zeros((Nx2+1))
ez_z2 = np.zeros((Nz2+1))
x = np.zeros((Nx2+1))
y = np.zeros((Ny2+1))
z = np.zeros((Nz2+1))
```

```
for i in range(0,Nx2+1):
    x[i] = i*dx
    ez_x1[i] = ez_pxy[px+i,py] # plano xy
    ez_x2[i] = ez_pxz[px+i,pz] # plano xz

for j in range(0,Ny2+1):
    y[j] = j*dy
    ez_y1[j] = ez_pxy[px,py+j] # plano xy

for k in range(0,Nz2+1):
    z[k] = k*dz
    ez_z2[k] = ez_pxz[px,pz+k] # plano xz


if Nx < Ny: # diagonal do plano xy
    Nd = Nx2
else:
    Nd = Ny2
ez_d1 = np.zeros((Nd+1))
s = np.zeros((Nd+1))

for k in range(0,Nd+1):
    s[k] = k*ds
    ez_d1[k] = ez_pxy[px+k,py+k]

if Nx < Nz: # diagonal do plano xz
    Nd2 = Nx2
else:
    Nd2 = Nz2
ez_d2 = np.zeros((Nd2+1))
s2 = np.zeros((Nd2+1))

ds2 = np.sqrt(dx**2 + dz**2)
for k in range(0,Nd2+1):
    s2[k] = k*ds2

    ez_d2[k] = ez_pxz[px+k,pz+k]

#*****************************************

# diagrama na forma polar no plano xz do campo ez
# modificações em 05/01/2016
pw = pz - 0
pw2 = px -0
dw = dz
dw2 = dx
Eincx = eincidente
Einc = Eincx #*4*np.pi
if ATIVA_POLAR == 1:
    # PLANO E (plano xz)
    raio1, teta, ezp, tetar, ezpr = gera_polar1(Max_polarx,porcentagemdepx,
                                Einc, dw, dw2, pw, pw2,ez_pxz,fator_raio)
    vmaxpolar = abs(ezp).max()
    ezp = ezp/vmaxpolar
```

```
# PLANO H (plqno xy)
pw = py - 0
pw2 = px - 0
dw = dy
dw2 = dx
Einc = eincidente2
Einc = Eincx #*4*np.pi
raio2, tetah, ezph, tetarh, ezprh = gera_polar1(Max_polarx2,porcentagemdepx,
                          Einc, dw, dw2, pw, pw2, ez_pxy,fator_raio)
vmaxpolarh = abs(ezph).max()
ezph = ezph/vmaxpolarh

# diagrama teorico de dipolos
N2 = 100
teta2 = np.zeros((N2+1))
ed2 = np.zeros((N2+1))
dteta = 2*np.pi/N2
for i in range(0,N2+1):
    if ATIVANDO_DIPOLO_1 == 1:
        teta2[i] = i*dteta
        rot = np.pi/2
        ed2[i] = abs((np.cos(np.pi*fator_comp*np.cos(teta2[i]+rot)) -\
            np.cos(np.pi*fator_comp))/(np.sin(teta2[i]+rot) + 1e-8))
        textop = texto3
    else:
        teta2[i] = i*dteta
        ed2[i] = abs(np.cos(teta2[i])**1)
        textop = u'dipolo curto'
textop1 = textop + u' fdtd' # para gráfico na forma polar
textop2 = textop + u' teórico'
#****************************************
# ESPALHAMENTO 3D DE PLACA RETANGULAR MODO TMz
Nteta2 = 500
fator_compa2 = largura_a    # gráfico rsc_h2x
fator_compb2 = largura_b    # gráfico rsc_e2x
ajuste_final = .98
fis = 1.*np.pi/2
atetai = 0.*np.pi/6
comp2 = 1.0 # usar 1.0: normaliza em relação ao comprimento de onda
atetas = np.zeros((Nteta2 + 1))
atetas2 = np.zeros((Nteta2 + 1))
rsc_e2 = np.zeros((Nteta2 + 1))
rsc_h2 = np.zeros((Nteta2 + 1))
atetas3 = np.zeros((2*Nteta2 + 1))
rsc_e3 = np.zeros((2*Nteta2 + 1))
rsc_h3 = np.zeros((2*Nteta2 + 1))
rsc_e2x = np.zeros((Nteta2 + 1))
rsc_h2x = np.zeros((Nteta2 + 1))

a2 = fator_compa2*comp2
b2 = fator_compb2*comp2
b = 2.0*np.pi/comp2
dteta2 = ajuste_final*np.pi/(2.0*Nteta2)
```

```
    for i in range(0,Nteta2+1):
        atetas[i] = i*dteta2
        atetas2[i] = i*dteta2*180/np.pi
        stetas2 = np.sin(atetas[i])*np.sin(atetas[i])
        ctetas2 = np.cos(atetas[i])*np.cos(atetas[i])
        sfis2 = np.sin(fis)*np.sin(fis)
        cfis2 = np.cos(fis)*np.cos(fis)
        ctetai2 = np.cos(atetai)*np.cos(atetai)
        Xp = (b*a2/2.0)*np.sin(atetas[i])*np.cos(fis)
        Yp = (b*b2/2.0)*(np.sin(atetas[i])*np.sin(fis) - np.sin(atetai))
        sinX2 = (np.sin(Xp)/(Xp + .0e-29))**2
        sinY2 = (np.sin(Yp)/(Yp + .0e-29))**2
        ab2 = (a2*b2/comp2)**2
        rsc_e2x[i] = 4.0*np.pi*ab2*(ctetai2*(ctetas2*cfis2 + sfis2))*sinX2*sinY2
        rsc_e2x[i] = 10.0*np.log10(np.abs(rsc_e2x[i]))

    for i in range(0,Nteta2+1):
        atetas[i] = i*dteta2
        atetas2[i] = i*dteta2*180/np.pi
        stetas2 = np.sin(atetas[i])*np.sin(atetas[i])
        ctetas2 = np.cos(atetas[i])*np.cos(atetas[i])
        sfis2 = np.sin(fis)*np.sin(fis)
        cfis2 = np.cos(fis)*np.cos(fis)
        ctetai2 = np.cos(atetai)*np.cos(atetai)
        Xp = (b*b2/2.0)*np.sin(atetas[i])*np.cos(fis)
        Yp = (b*a2/2.0)*(np.sin(atetas[i])*np.sin(fis) - np.sin(atetai))
        sinX2 = (np.sin(Xp)/(Xp + .0e-29))**2
        sinY2 = (np.sin(Yp)/(Yp + .0e-29))**2
        ab2 = (a2*b2/comp2)**2
        rsc_h2x[i] = 4.0*np.pi*ab2*(ctetai2*(ctetas2*cfis2 + sfis2))*sinX2*sinY2
        rsc_h2x[i] = 10.0*np.log10(np.abs(rsc_h2x[i]))

    #*********************************************************
    # PLACA RETANGULAR COM FORMULAS SIMPLIFICADAS ??
    for i in range(0,Nteta2+1):
        atetas[i] = i*dteta2
        atetas2[i] = i*dteta2*180/np.pi
        stetas2 = np.sin(atetas[i])*np.sin(atetas[i])
        ctetas2 = np.cos(atetas[i])*np.cos(atetas[i])
        esintetab = np.sin(b*b2*np.sin(atetas[i]))/(b*b2*np.sin(atetas[i]))
        ab2 = (a2*b2/comp2)**2
        rsc_e2[i] = 4.0*np.pi*ab2*ctetas2*esintetab*esintetab
        rsc_e2[i] = 10.0*np.log10(np.abs(rsc_e2[i]))
        esintetaa = np.sin(b*a2*np.sin(atetas[i]))/(b*a2*np.sin(atetas[i]))
        ab2 = (a2*b2/comp2)**2
        rsc_h2[i] = 4.0*np.pi*ab2*ctetas2*esintetaa*esintetaa
        rsc_h2[i] = 10.0*np.log10(np.abs(rsc_h2[i]))
    for i in range(0,Nteta2+1):
        atetas3[i] = 90.0 - atetas2[Nteta2-i]
        rsc_e3[i] = rsc_e2x[Nteta2-i] # corrigido para equações exatas
        rsc_h3[i] = rsc_h2x[Nteta2-i]
        atetas3[Nteta2+i] = 90.0 + atetas2[i]
        rsc_e3[Nteta2+i] = rsc_e2x[i]
        rsc_h3[Nteta2+i] = rsc_h2x[i]
# final da placa retangular
```

```
#****************************************
# ENTRADA DE ARQUIVOS PARA A PLACA YEE 3D QUE SERÃO USADOS NA
# COMPARAÇÃO COM A PLACA HEX 3D E PLACA TEÓRICA

if (ENTRADA_ARQUIVOS_PLACA_YEE == 1):
    arquivox1 = open('pyee_dados_placa_T420.txt', 'r')
    tempomax_yee = int(float(arquivox1.readline()))
    a_yee = float(arquivox1.readline())
    b_yee = float(arquivox1.readline())
    compE = int(float(arquivox1.readline()))
    compH = int(float(arquivox1.readline()))
    arquivox1.close()
    #print 'tempomax_yee =', tempomax_yee
    ezp_yee = np.zeros((compE))
    teta_yee = np.zeros((compE))
    ezph_yee = np.zeros((compH))
    tetah_yee = np.zeros((compH))

    # PLANO E
    esf_planoE = 'pyee_planoE_T' + str(tempomax_yee) + '.txt'
    arquivox1 = open(esf_planoE, 'r')
    for i in range(0,compE):
        linha = arquivox1.readline()
        ezp_yee[i] = float(linha)
    arquivox1.close()

    esf_anguloE = 'pyee_anguloE_T' + str(tempomax_yee) + '.txt'
    arquivox1 = open(esf_anguloE, 'r')
    for i in range(0,compE):
        linha = arquivox1.readline()
        teta_yee[i] = float(linha)
    arquivox1.close()

    # PLANO H
    esf_planoH = 'pyee_planoH_T' + str(tempomax_yee) + '.txt'
    arquivox1 = open(esf_planoH, 'r')
    for i in range(0,compH):
        linha = arquivox1.readline()
        ezph_yee[i] = float(linha)
    arquivox1.close()

    esf_anguloH = 'pyee_anguloH_T' + str(tempomax_yee) + '.txt'
    arquivox1 = open(esf_anguloH, 'r')
    for i in range(0,compH):
        linha = arquivox1.readline()
        tetah_yee[i] = float(linha)
    arquivox1.close()

    # FINAL SAIDA DOS ARQUIVOS DA ESFERA YEE 3D

#****************************************************

ez_pxy2 = ez_pxy.T # transposta para corrigir imagem
ez_pxz2 = ez_pxz.T # transposta para corrigir imagem
```

```
#*****************************************************

if ATIVA_POLAR == 1:
    print 'px = ',px
    print 'py = ',py
    print 'pz = ',pz
    print 'raio1_plano_E = ',raio1
    print 'raio2_plano_H = ',raio2
#print'\n'
print u'FINAL'

#****************************************

# PLOTANDO CAMPOS Ez PARA O MODO 3D
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

# nearest ou bilinear - funcionou
# cm.jet ou cm.RdYlGn ou cm.winter ou cm.hot
# ficou imagem correta com shape=None e com extent=[0,Lx,0,Ly]
# mas shape inverteu eixos x e y
# no entanto usando a transposta de ez a imagem ficou com eixos corretos

# GRÁFICOS 2D
if MOSTRAR_SOMENTE_GRAFICOS_1D == 0:
    fs1 = 14   # xlabel e ylabel
    fs2 = 15   # title
    fse = 13   # numeros nos eixos x e y
    # plano xy
    plt.figure()
    im = plt.imshow(ez_pxy2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Ly], shape=None,
                vmax= nivel*abs(ez_pxy2).max(), vmin= -nivel*abs(ez_pxy2).max()  )

    plt.colorbar(im) # funcionou
    plt.title(u'HEX 3D: Ez (V/m) NO PLANO XY (T = '+str(tempomax)+' passos)')
    plt.xlabel(u'Lx (metros)',fontsize = fs1)
    plt.ylabel(u'Ly (metros)',fontsize = fs1)
    #plt.show()

    # plano xz
    plt.figure()
    im2 = plt.imshow(ez_pxz2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Lz], shape=None,
                vmax= nivel*abs(ez_pxz2).max(), vmin= -nivel*abs(ez_pxz2).max()  )

    plt.colorbar(im2) # funcionou
    plt.title(u'HEX 3D: Ez (V/m) NO PLANO XZ (T = '+str(tempomax)+' passos)')
    plt.xlabel(u'Lx (metros)',fontsize = fs1)
    plt.ylabel(u'Lz (metros)',fontsize = fs1)
    #plt.show()

#*********************************************
# GRÁFICOS 1D
```

```
# no plano xy
plt.figure()
plt.plot(x,ez_x1,'r')
plt.grid()
plt.xlabel(u'Eixos x/y/d, '+ textof)
plt.ylabel(u'Nível')
plt.title('Campos 1D: Ez plano xy (TEMPO = '+str(tempomax)+' passos)')
plt.hold(True) # permite sobrepor vários
               # gráficos com dimensões diferentes
plt.plot(y,ez_y1,'b')
plt.plot(s,ez_d1,'k-')
texto1 = u'ez_x1 ( 0°)'
texto2 = u'ez_y1 (90°)'
texto3 = u'ez_d1 (30°)'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)

#plt.show()

# no plano xz
plt.figure()
plt.plot(x,ez_x2,'r')
plt.grid()
plt.xlabel(u'Eixos x/z/d, '+ textof)
plt.ylabel(u'Nível')
plt.title('Campos 1D: Ez plano xz (TEMPO = '+str(tempomax)+' passos)')
plt.hold(True) # permite sobrepor vários
               # gráficos com dimensões diferentes

plt.plot(z,ez_z2,'b-')
plt.plot(s2,ez_d2,'k')
texto1 = u'ez_x2 ( 0°)'
texto2 = u'ez_z2 (90°)'
texto3 = u'ez_d2 (45°)'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)
#plt.show()

#*****************************************************

if MOSTRAR_SOMENTE_GRAFICOS_1D == 1:
   # fonte em função do tempo
   plt.figure()
   plt.plot(tempo_teste,ez_teste,'k')
   plt.grid()
   plt.xlabel(u'Eixo Tempo, '+ textof)
   plt.ylabel(u'Nível')
   plt.title('Fonte para campo Ez ( TEMPO = '+str(tempomax)+\
           ' passos, FATOR_AT = '+str(fator_at)+' )' )
   #plt.show()

#*************************************************************
# diagrama na forma polar e espalhamento de placa retangular
if ATIVA_POLAR == 1:
   # PLANO E (plano xz)
```

```
plt.figure()
plt.polar(teta,ezp,'r')
plt.hold(True) # permite sobrepor vários
               # gráficos com dimensões diferentes
plt.polar(teta2,ed2,'b')
plt.title('Polar, raio ='+str(raio1)+' '+ textof)
plt.legend((textop1,textop2),loc = 'lower right')
plt.hold(False)
# graficos para espalhamento da placa retangular
plt.figure()
plt.plot(tetar,ezpr,'k') # ou plt.semilogy
plt.grid()
plt.xlabel(u'Ângulo (graus)')
plt.ylabel(u'RCS (dBsm)') # ou dBsw
plt.title(u'Hex 3D: RCS (Plano E) da placa retangular (a = '+\
          str(a2) + r'$\lambda$; b = '+str(b2)+r'$\lambda$)')
plt.hold(True) # permite sobrepor vários
               # gráficos com dimensões diferentes
plt.plot(atetas3,rsc_e3,'r')
#plt.plot(s,ez_d1,'k-')
texto1 = u'Hex 3D'
texto2 = u'Teórico'
#texto3 = u'ez_d1 (30°)'
plt.legend((texto1,texto2),loc = 'upper right')
plt.hold(False)

# PLANO H (plano xy)
plt.figure()
plt.polar(tetah,ezph,'r')
plt.hold(True) # permite sobrepor vários
               # gráficos com dimensões diferentes
plt.polar(teta2,ed2,'b')
plt.title('Polar, raio ='+str(raio2)+' '+ textof)
plt.legend((textop1,textop2),loc = 'lower right')
plt.hold(False)
# graficos para espalhamento da placa retangular
plt.figure()
plt.plot(tetarh,ezprh,'k') # ou plt.semilogy
plt.grid()
plt.xlabel(u'Ângulo (graus)')
plt.ylabel(u'RCS (dBsm)') # ou dBsm
plt.title(u'Hex 3D: RCS (Plano H) da placa retangular (a = '+\
          str(a2) + r'$\lambda$; b = '+str(b2)+r'$\lambda$)')
plt.hold(True) # permite sobrepor vários
               # gráficos com dimensões diferentes
plt.plot(atetas3,rsc_h3,'r')
#plt.plot(s,ez_d1,'k-')
texto1 = u'Hex 3D'
texto2 = u'Teórico'
#texto3 = u'ez_d1 (30°)'
plt.legend((texto1,texto2),loc = 'upper right')
plt.hold(False)

if (ENTRADA_ARQUIVOS_PLACA_YEE == 1):
   # Plano E: graficos para espalhamento da placa
```

```
        plt.figure()
        plt.plot(tetar,ezpr,'k') # ou plt.semilogy
        plt.grid()
        plt.xlabel(u'Ângulo (graus)')
        plt.ylabel(u'RCS (dBsm)') # ou dBsw
        plt.title(u'RCS (Plano E) da placa retangular (a = '+\
              str(a2) + r'$\lambda$; b = '+str(b2)+r'$\lambda$)')
        plt.hold(True) # permite sobrepor vários
                       # gráficos com dimensões diferentes
        plt.plot(teta_yee,ezp_yee,'b-')
        plt.plot(atetas3,rsc_e3,'r')
        texto1 = u'Hex 3D'
        texto2 = u'Yee 3D'
        texto3 = u'Teórico'
        plt.legend((texto1,texto2,texto3),loc = 'lower center')
        plt.hold(False)
        # Plano H: graficos para espalhamento da placa
        plt.figure()
        plt.plot(tetarh,ezprh,'k') # ou plt.semilogy
        plt.grid()
        plt.xlabel(u'Ângulo (graus)')
        plt.ylabel(u'RCS (dBsm)') # ou dBsw
        plt.title(u'RCS (Plano H) da placa retangular (a = '+\
              str(a2) + r'$\lambda$; b = '+str(b2)+r'$\lambda$)')
        plt.hold(True) # permite sobrepor vários
                       # gráficos com dimensões diferentes
        plt.plot(tetah_yee,ezph_yee,'b-')
        plt.plot(atetas3,rsc_h3,'r')
        texto1 = u'Hex 3D'
        texto2 = u'Yee 3D'
        texto3 = u'Teórico'
        plt.legend((texto1,texto2,texto3),loc = 'lower center')
        plt.hold(False)
        plt.show()

plt.show()

# FINAL DO PROGRAMA
### FINAL DO APÊNDICE 8
################################################################################
```

APÊNDICE 9. PROGRAMAS PARA MEDIR O ESPALHAMENTO DE CAMPO DE ESFERA METÁLICA PARA AMBOS OS MÉTODOS FDTD

ESTE APÊNDICE CONTÉM 2 ARQUIVOS

ARQUIVO 1:MEDIDAS DE CAMPO ESPALHADO PARA A GRADE YEE

```
# -*- coding: cp1252 -*-
#            cp1252 ou utf-8
# programa programa_fdtd_3d_h_2c_visualizacao_1d.py em python 2.7.3
# derivado do programa: programa_fdtd_3d_h_2c_visualizacao_1c.py
# AUTOR: MARINOEL JOAQUIM

#*********************************************************
# BIBLIOTECAS NECESSÁRIAS

import numpy as np
import time
import scipy.special as sp
#*********************************************************
tempo1 = time.time()

# INICIO DAS FUNÇÕES

# FUNÇÃO PARA FONTE DO TIPO PULSO RICKER OU SENOIDE
def fonte_1(tempo2,posicao,tipo_fonte,fator_at):
    if tipo_fonte == 0: # pulso ricker
        arg = np.pi*( (cdtds*tempo2 - posicao)/ppw - 1.0 )
        arg2 = arg*arg
        funcao =(1.0 - 2.0*arg2)*np.exp(-arg2)

    else:  # senóide com amortecimento inicial
        periodo = ppw/cdtds # periodo da senóide
                        # ou tempo para o pico do pulso ricker
        cte_tempo = fator_at*periodo
        funcao = ( 1.0 - np.exp(-1.0*tempo2/cte_tempo) )*\
                np.sin(2*np.pi*tempo2*cdtds/ppw)
    return funcao
#***********************************

# função para obter i,j a partir de p para campo ez
def ez_ij(p2):
    p1 = 2*p2 - 1
    j = p1/Nx + 1
    if (p1%Nx==0 and p1>=Nx):
        j = j - 1
    i = p1 - (j-1)*Nx
    return i,j
#*********************************************************

# Função de interpolação para gráfico de ez no plano xy
```

```
def pteste(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        i = 0
        j = 0
    return i, j

# Função de interpolação para gráfico de ez no plano xz
def pteste2(i,k):
    if ((i<=0) or (i>=Nx+1) or (k<=0) or (k>=Nz+1)):
        i = 0
        k = 0
    return i, k
#*********************************************************

# Função para criar grafico na forma polar
def gera_polar1(Max_polarx,porcentagemdepx, Einc, dw, dw2, pw, pw2,ez_puv):
    tamanhox = int(porcentagemdepx*pw2) # parte inicial não usada
    ez_x1polar = np.zeros((tamanhox))   # 0 a tamanhox
    ez_x1polar[:] = ez_pxy[:tamanhox,py]
    nxp = ez_x1polar.argmax()  # valor reduzido da dimensão central no eixo x
    if nxp <=10: nxp = 10
    if  MAXPOLAR_AUTOMATICO == 0:
        n = int(round((pw2 - nxp - 1)*dw2/dw)) # raio da medidas = n*dw
    else:
        n = int((pw2 - Max_polarx)*dw2/dw)

    dw = 1.0*dw # referencia principal dz ou dy: 0 < k < (n+1)
    dw2 = 1.0*dw2 # referencia dependente dx: 0 < m[k] < (n+1)*dw/dw2
    raio = n*dw
    #raio2 = int(raio)
    teta = np.zeros((4*n+1))
    ezp = np.zeros((4*n+1))
    a = np.zeros((n+1))
    b = np.zeros((n+1))
    m = np.zeros((n+1))
    tetar = np.zeros((2*n+1))
    ezpr = np.zeros((2*n+1))
    for k in range(0,n+1):
        a[k] = k*dw
        b[k] = np.sqrt(raio**2 - a[k]**2)
        m[k] = int(round(b[k]/dw2)) # modificado
        #teta[k] = np.arctan2(a[k],b[k])
        teta[k] = np.arctan2(a[k],m[k]*dw2)
        ezp[k] = abs(ez_puv[pw2 + m[k],pw + k])
    for k in range(0,n+1):
        teta[k+n] = np.pi - teta[n-k]
        ezp[k+n] = abs(ez_puv[pw2 - m[n-k],pw + n - k])
        teta[k+2*n] = np.pi + teta[k]
        ezp[k+2*n] = abs(ez_puv[pw2 - m[k],pw - k])
        teta[k+3*n] = 2*np.pi - teta[n-k]
        ezp[k+3*n] = abs(ez_puv[pw2 + m[n-k],pw - n + k])
    for k in range(0,2*n+1):
        tetar[k] = teta[k]*180/np.pi
        ezpr[k] = ezp[k]
        raioesf = 1.0*dx*ppw
```

```
        einc2 = Einc*(raio - raioesf )/(raio - raioesf +\
                        raioesf*(1.0 + np.cos(teta[k])))
        #einc2 = Einc*(np.cos(teta[k])*np.cos(fix) - np.sin(fix))
        einc2 = Einc
        raio3 = raio
        ezpr[k] = ezpr[k]*ezpr[k]*4.0*np.pi*raio3*raio3/((raioesf*einc2)**2)
        ezpr[k] = 10*np.log10(ezpr[k])
    return raio, teta, ezp, tetar, ezpr
    # Final função para gráfico polar

#*********************************************
# FINAL DAS FUNÇÕES
#*********************************************************
#*********************************************************

# INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS
Nx = 81 # numero de pontos na direção x
Ny = 141  # numero de pontos na direção y
Nz = 71 # numero de pontos na direção z
ppw = 15 # numero de pontos por comprimento de onda
tempomax = 40

tipo_fonte = 1  # 0=pulso ricker ou 1=senoide
HARD_SOFT  = 1   # fonte: 0=hard ou 1=soft
ATIVANDO_DIPOLO_1 = 0 # 1=ativa 0=desativa
ATIVANDO_DIPOLO_2 = 0 # 1=ativa 0=desativa

rdsdz = 1.0 #np.sqrt(1.0) # razão ds/dz
fator_comp = 1.4 # em comprimentos de onda para dipolo_1

arqsaida1 = "ez_pxy_1.txt"
arqsaida2 = "ez_pxz_2.txt"
fator_at = 0.5

# FINAL DAS PRINCIPAIS VARIÁVEIS

#*************************************************

# VARIÁVEIS DE CONTROLE DOS GRÁFICOS

USAR_ESTES_ARQUIVOS = 0 # usar 1 ou 2, ou 0

MOSTRAR_SOMENTE_GRAFICOS_1D = 0 # 1= graficos 1D, 0 = mostra tudo

ATIVA_POLAR = 1 # 0 = desativa 1 = ativa diagrama polar
nivel = 1e-1  # controla nível máximo graficos 2D

SAIDA_ARQUIVOS_ESFERA_YEE = 1  # 0=desativa ou 1=ativa
MAXPOLAR_AUTOMATICO = 1 # automatico=0 ou usar Max_polarx
# posiçãodo máximo no eixo x (0 a Max_polarx)
Max_polarx  = 38
Max_polarx2 = 38
Eincx = 1.0
raio_esfera = .5
porcentagemdepx = 0.3# para obter o maxpolar automático
```

```
#*****************************************************


# LEITURA DE ARQUIVO DE ENTRADA
if USAR_ESTES_ARQUIVOS == 0:
   arquivo3 = open("entrada4.txt","r")

if USAR_ESTES_ARQUIVOS == 1:
   arquivo3 = open("entrada_ta1.txt","r")

if USAR_ESTES_ARQUIVOS == 2:
   arquivo3 = open("entrada_ta2.txt","r")

Nx = int(arquivo3.readline())#funcionou
Ny =  int(arquivo3.readline())
Nz =  int(arquivo3.readline())
ppw =  int(arquivo3.readline())
tempomax =  int(arquivo3.readline())
tipo_fonte =  int(arquivo3.readline())
HARD_SOFT =  int(arquivo3.readline())
ds =  float(arquivo3.readline())
rdsdz =  float(arquivo3.readline())
fator_cdtds = float(arquivo3.readline())
fator_at =  float(arquivo3.readline())
delta_passo = int(arquivo3.readline())
tempomax2 =  int(arquivo3.readline())
tempomax3 =  int(arquivo3.readline())
CPML =  int(arquivo3.readline())
nxx =  int(arquivo3.readline())
Reflexao = float(arquivo3.readline())
Reflexao2 = float(arquivo3.readline())
me = float(arquivo3.readline())
me2 = float(arquivo3.readline())
numero_materiais = int(arquivo3.readline())
numero_objetos = int(arquivo3.readline())
arquivo3.close()

fator_geometria = 2
# RETIRANDO "\n" DAS STRINGS
txx1 = list(arqsaida1)
txx1.pop()
arqsaida1 = "".join(txx1)

txx2 = list(arqsaida2)
txx2.pop()
arqsaida2 = "".join(txx2)

if USAR_ESTES_ARQUIVOS == 0:
   arqsaida1 = "ez_pxy_yee1_693.txt"
   arqsaida2 = "ez_pxz_yee2_693.txt"
   if TESTE_REFLEXAO == 1:
      arqsaida3  = "ez_YA_nxx_18_CPML_0_Temp_243.txt"
      arqsaida3b = "ez_YA_nxx_18_CPML_1_Temp_243.txt"
      arqsaida4  = "ez_YB_nxx_18_CPML_0_Temp_243.txt"
      arqsaida4b = "ez_YB_nxx_18_CPML_1_Temp_243.txt"
```

```
        arqsaida5  = "ez_YC_nxx_18_CPML_0_Temp_243.txt"
        arqsaida5b = "ez_YC_nxx_18_CPML_1_Temp_243.txt"
        arqsaida6  = "ez_YD_nxx_18_CPML_0_Temp_243.txt"
        arqsaida6b = "ez_YD_nxx_18_CPML_1_Temp_243.txt"

if USAR_ESTES_ARQUIVOS == 1:
    arqsaida1 = "ez_pxy_1_ta1.txt"
    arqsaida2 = "ez_pxz_2_ta1.txt"

if USAR_ESTES_ARQUIVOS == 2:
    arqsaida1 = "ez_pxy_1_ta2.txt"
    arqsaida2 = "ez_pxz_2_ta2.txt"

cdtds = 1.0/np.sqrt(fator_cdtds) # número de Courant

##################################################
eo = 1.0e-9/(36*np.pi)
uo = np.pi*4e-7
co = 1.0/np.sqrt(uo*eo)
print 'co =',co

# IMPRIME VALORES OBTIDOS DO ARQUIVO DE ENTRADA
print 'Nx =',Nx
print 'Ny =',Ny
print 'Nz =',Nz
print 'ppw =',ppw
print 'tempomax =',tempomax
print 'tipo_fonte =',tipo_fonte
print 'HARD_SOFT =',HARD_SOFT
print 'ATIVANDO_DIPOLO_1 =',ATIVANDO_DIPOLO_1
print 'ATIVANDO_DIPOLO_2 =',ATIVANDO_DIPOLO_2
print 'rdsdz =', rdsdz
print 'cdtds =', cdtds
print 'fator comprimento dipolo  =',fator_comp
print 'fator atenuação (senóide) =',fator_at
#print 'fator geometria (1, 2, 3) =',fator_geometria
#print 'arqsaida1 =',arqsaida1
#print 'arqsaida2 =',arqsaida2
# FINAL DA LEITURA


#**************************************

razaodsdz = round(rdsdz,2) # para gráfico
CFL = int(cdtds*100)/100.0 # número de Courant para gráfico

ativo = 1  # tipo dipolo_1
passivo = 0
comprimento_dipolo = int(fator_comp*rdsdz*ppw)

compdipolo = int(fator_comp*100)/100.0 # para gráfico
imp0 = 120.0*np.pi

#**********************************************
```

```
if (fator_geometria <= 2):
   ds = 1.0  # dimensão dos lados do hexagono maior 2D
   dx = ds # tamanho da célula na direção x
   dy = ds   # tamanho da célula na direção y
   dz = ds
   Lx = (Nx)*dx  # comprimento na direção x
   Ly = (Ny)*dy  # comprimento na direção y
   Lz = (Nz-1)*dz  # comprimento na direção z

#*****************************************************

tipo_1d = 0 # grade 1D auxiliar
tipo_3d = 3 # grade 3D

# calculando dimensões das matrizes para os campos E e H
ix = Nx/2
rx = Nx%2
jy = Ny/2
ry = Ny%2
jy2 = (Ny-1)/2
ry2 = (Ny-1)%2

PEZ = (ix + rx)*(jy + ry) + (ix)*(jy)
PH2 = (ix)*(jy + ry) + (ix + rx)*(jy)
PH1 = (ix)*(jy2 + ry2) + (ix)*(jy2)
PH3 = PH1
#print 'PEZ= ',PEZ

# matrizes para os gráficos de Ez nos planos xy e xz
ez_pxy = np.zeros((Nx,Ny)) # modificado para YEE 3D
ez_pxz = np.zeros((Nx,Nz-1))

ez_pxy2 = np.zeros((Ny,Nx)) # transposta de ez_pxy
ez_pxz2 = np.zeros((Nz-1,Nx)) # transposta de ez_pxz

#************************************
px = Nx/2 + 1 # posição da fonte ricker
py = Ny/2 + 1
pz = Nz/2 + 1
pcentro = p2b(px,py)
#*****************************************

# textos para gráficos
texto2 =  'dim='+str(Nx)+'x'+str(Ny)+'x'+str(Nz)+\
          '  ppw='+str(ppw)+'  CFL='+str(CFL)+\
          ' dx=dy=dz='+str(ds) + ' PML='+ str(int(nxx*dx)) + ' m'
if tipo_fonte == 0:
   texto1 = u'pulso '
else:
   texto1 = u'seno '
if HARD_SOFT == 0:
   texto1 = texto1 + 'hard '
else:
   texto1 = texto1 + 'soft '
if ATIVANDO_DIPOLO_1 == 1 :
```

```
   texto3 = '  dipolo '+str(compdipolo)
else:
   texto3 = ''
textof = texto1 + texto2 + texto3
#************************************

# testando a fonte sozinha no tempo
ez_teste = np.zeros((tempomax))
tempo_teste = np.zeros((tempomax))

#******************************

# INICIO DOS CÁLCULOS PARA GRADE HEXAGONAL 3D

print u'\nINICIO FDTD'
print

# inicio loop do tempo

for tempo in range(0,tempomax):

   tempo_teste[tempo] = tempo
   ez_teste[tempo] = fonte_1(tempo,0.0,tipo_fonte,fator_at)

   #print '\n',tempo+1,'/',tempomax

# final loop do tempo
#*********************************************

#*****************************************

# LEITURA DAS MATRIZES PARA PLANOS XY E XZ
arquivo4 = open(arqsaida1,"r")
#linha = None
for i in range(0,Nx): # modificado para YEE 3D
    for j in range(0,Ny):
       linha = arquivo4.readline() #funcionou
       ez_pxy[i,j] = float(linha)
arquivo4.close()

arquivo5 = open(arqsaida2,"r")
#linha = None
for i in range(0,Nx): # modificado para YEE 3D
    for j in range(0,Nz-1):
       linha = arquivo5.readline() #funcionou
       ez_pxz[i,j] = float(linha)
arquivo5.close()

#************************************************
# gerando gráficos 1D a partir dos gráficos 2D
Nx2 = Nx - px
Ny2 = Ny - py
Nz2 = Nz - 1 - pz
ez_x1 = np.zeros((Nx2+1))
ez_y1 = np.zeros((Ny2+1))
```

```
ez_x2 = np.zeros((Nx2+1))
ez_z2 = np.zeros((Nz2+1))
x = np.zeros((Nx2+1))
y = np.zeros((Ny2+1))
z = np.zeros((Nz2+1))

for i in range(0,Nx2):
    x[i] = i*dx
    ez_x1[i] = ez_pxy[px+i,py] # plano xy
    ez_x2[i] = ez_pxz[px+i,pz] # plano xz

for j in range(0,Ny2):
    y[j] = j*dy
    ez_y1[j] = ez_pxy[px,py+j] # plano xy

for k in range(0,Nz2):
    z[k] = k*dz
    ez_z2[k] = ez_pxz[px,pz+k] # plano xz


if Nx < Ny: # diagonal do plano xy
    Nd = Nx2
else:
    Nd = Ny2
ez_d1 = np.zeros((Nd+1))
s = np.zeros((Nd+1))
ds3 = np.sqrt(dx**2 + dy**2) # alterando para Yee 3D
for k in range(0,Nd):
    s[k] = k*ds3
    ez_d1[k] = ez_pxy[px+k,py+k]

if Nx < Nz: # diagonal do plano xz
    Nd2 = Nx2
else:
    Nd2 = Nz2
ez_d2 = np.zeros((Nd2+1))
s2 = np.zeros((Nd2+1))

ds2 = np.sqrt(dx**2 + dz**2)
for k in range(0,Nd2):
    s2[k] = k*ds2

    ez_d2[k] = ez_pxz[px+k,pz+k]

#*****************************************

# diagrama na forma polar no plano xz do campo ez
# modificações em 05/01/2016
pw = pz - 1
pw2 = px - 1
dw = dz
dw2 = dx
Einc = Eincx     #*4*np.pi
if ATIVA_POLAR == 1:
    raio1, teta, ezp, tetar, ezpr = gera_polar1(Max_polarx,porcentagemdepx,
```

```
                                Einc, dw, dw2, pw, pw2,ez_pxz)
vmaxpolar = abs(ezp).max()
ezp = ezp/vmaxpolar
pw = py - 1
pw2 = px - 1
dw = dy
dw2 = dx
#Einc = 2.
Einc = Eincx   #*4*np.pi
raio2, tetah, ezph, tetarh, ezprh = gera_polar1(Max_polarx2,porcentagemdepx,
                                Einc, dw, dw2, pw, pw2, ez_pxy)
vmaxpolarh = abs(ezph).max()
ezph = ezph/vmaxpolarh

# diagrama teorico de dipolos
N2 = 100
teta2 = np.zeros((N2+1))
ed2 = np.zeros((N2+1))
dteta = 2*np.pi/N2
for i in range(0,N2+1):
    if ATIVANDO_DIPOLO_1 == 1:
        teta2[i] = i*dteta
        rot = np.pi/2
        ed2[i] = abs((np.cos(np.pi*fator_comp*np.cos(teta2[i]+rot)) -\
            np.cos(np.pi*fator_comp))/(np.sin(teta2[i]+rot) + 1e-8))
        textop = texto3
    else:
        teta2[i] = i*dteta
        ed2[i] = abs(np.cos(teta2[i])**1)
        textop = u'dipolo curto'
textop1 = textop + u' fdtd' # para gráfico na forma polar
textop2 = textop + u' teórico'
#****************************************

# CALCULANDO ESPALHAMENTO TEORICO DO CAMPO POR ESFERA
N = 100
Nteta = 360
fator_comp = raio_esfera
comp = 1.0 # usar 1.0 para normalizar o raio a em relação ao comp
ateta = np.zeros((Nteta + 1))
ateta2 = np.zeros((Nteta + 1))
an = np.zeros((Nteta + 1),dtype = complex)
bn = np.zeros((Nteta + 1),dtype = complex)
cn = np.zeros((Nteta + 1),dtype = complex)
Ateta = np.zeros((Nteta + 1),dtype = complex)
Afi = np.zeros((Nteta + 1),dtype = complex)
rsc_e = np.zeros((Nteta + 1))
rsc_h = np.zeros((Nteta + 1))

a = fator_comp*comp
b = 2.0*np.pi/comp
dteta = np.pi/Nteta

for n in range(1,N+1):
    an[n] = np.power(1.0j, -n)*(2.0*n+1.0)/(1.0*n*(n+1.0))
```

```
        jv = sp.jn(n+0.5,b*a)
        h2 = sp.hankel2(n+0.5,b*a)
        jvd = b*sp.jvp(n+0.5,b*a,1)    #corrigido para b*
        h2d = b*sp.h2vp(n+0.5,b*a,1)   #corrigido para b*
        jve = np.sqrt(np.pi*b*a/2.0)*jv
        h2e = np.sqrt(np.pi*b*a/2.0)*h2
        jved = np.sqrt(np.pi*b/(8.0*a))*jv + np.sqrt(np.pi*b*a/2.0)*jvd
        h2ed = np.sqrt(np.pi*b/(8.0*a))*h2 + np.sqrt(np.pi*b*a/2.0)*h2d
        bn[n] = -an[n]*jved/h2ed
        cn[n] = -an[n]*jve/h2e
        #print cn[n]

    for i in range(0,Nteta+1):
        ateta[i] = i*dteta
        ateta2[i] = i*dteta*180.0/np.pi
        xr = np.cos(ateta[i])
        yr = np.sin(ateta[i])
        lp, lpd = sp.lpmn(1,N,xr)
        for n in range(1,N+1):
            Ateta[i] += np.power(1.0j,n)*(bn[n]*yr*lpd[1,n] -\
                    cn[n]*lp[1,n]/(yr + 1e-26))
            Afi[i] += np.power(1.0j,n)*(bn[n]*lp[1,n]/(yr + 1e-26) -\
                    cn[n]*yr*lpd[1,n])
        Ateta[i] = np.abs(Ateta[i])
        Ateta2 = np.abs( Ateta[i]*Ateta[i] )
        Afi[i] = np.abs(Afi[i])
        Afi2 = np.abs(Afi[i]*Afi[i])
        fi = 0.0
        co2 = np.cos(fi)*np.cos(fi)
        si2 = np.sin(fi)*np.sin(fi)
        rsc_e[i] = (comp*comp/np.pi)*(co2*Ateta2 + si2*Afi2)
        fi = np.pi/2.0
        co2 = np.cos(fi)*np.cos(fi)
        si2 = np.sin(fi)*np.sin(fi)
        rsc_h[i] =(comp*comp/np.pi)*(co2*Ateta2 + si2*Afi2)
        rsc_e[i] = 10.0*np.log10(rsc_e[i])
        rsc_h[i] = 10.0*np.log10(rsc_h[i])
#************************************************************

# SAIDA DE ARQUIVOS PARA A ESFERA YEE 3D QUE SERÃO USADOS NA
# COMPARAÇÃO COM A ESFERA HEX 3D E ESFERA TEÓRICA

if (SAIDA_ARQUIVOS_ESFERA_YEE == 1):
    tempomax_yee = tempomax
    raio_yee = raio_esfera
    compE = len(ezpr)
    compH = len(ezprh)
    #print 'compE =' , compE
    dados_esfera = [tempomax_yee,raio_yee, compE, compH]
    saida_dados = 'yee_dados_esfera_T' + str(tempomax) + '.txt'
    arquivox1 = open(saida_dados, "w")
    for i in range(0,len(dados_esfera)):
        arquivox1.write("%f\n" % dados_esfera[i])
    arquivox1.close()
```

```
    # PLANO E
    esf_planoE = 'yee_planoE_T' + str(tempomax) + '.txt'
    arquivox1 = open(esf_planoE, 'w')
    for i in range(0,compE):
        arquivox1.write("%18.17f\n" % ezpr[i])
    arquivox1.close()

    esf_anguloE = 'yee_anguloE_T' + str(tempomax) + '.txt'
    arquivox1 = open(esf_anguloE, 'w')
    for i in range(0,compE):
        arquivox1.write("%18.17f\n" % tetar[i])
    arquivox1.close()

    # PLANO H
    esf_planoH = 'yee_planoH_T' + str(tempomax) + '.txt'
    arquivox1 = open(esf_planoH, 'w')
    for i in range(0,compH):
        arquivox1.write("%18.17f\n" % ezprh[i])
    arquivox1.close()

    esf_anguloH = 'yee_anguloH_T' + str(tempomax) + '.txt'
    arquivox1 = open(esf_anguloH, 'w')
    for i in range(0,compH):
        arquivox1.write("%18.17f\n" % tetarh[i])
    arquivox1.close()
# FINAL SAIDA DOS ARQUIVOS DA ESFERA YEE 3D

#*****************************************************************

ez_pxy2 = ez_pxy.T # transposta para corrigir imagem
ez_pxz2 = ez_pxz.T # transposta para corrigir imagem

#*********************************************************

if ATIVA_POLAR == 1:
    #print 'px = ',px
    #print 'px - n = ',px - n
    #print 'n = ',n
    print 'raio1 (plano E) = ',raio1
    print 'raio2 (plano E) = ',raio2
#print'\n'
print u'FINAL'

#****************************************

# PLOTANDO CAMPOS Ez PARA O MODO 3D
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

# nearest ou bilinear - funcionou
# cm.jet ou cm.RdYlGn ou cm.winter ou cm.hot
# ficou imagem correta com shape=None e com extent=[0,Lx,0,Ly]
# mas shape inverteu eixos x e y
# no entanto usando a transposta de ez a imagem ficou com eixos corretos
```

```
# GRÁFICOS 2D
if MOSTRAR_SOMENTE_GRAFICOS_1D == 0:
    fs1 = 14
    fs2 = 15
    fse = 13
    # plano xy
    plt.figure()
    im = plt.imshow(ez_pxy2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Ly], shape=None,
                vmax= nivel*abs(ez_pxy2).max(), vmin= -nivel*abs(ez_pxy2).max()  )

    plt.colorbar(im) # funcionou
    plt.title(u'YEE 3D: Ez (V/m) NO PLANO XY (T = '+str(tempomax)+' passos)')
    plt.xlabel(u'Lx (metros)',fontsize = fs1)
    plt.ylabel(u'Ly (metros)',fontsize = fs1)
    #plt.show()

    # plano xz
    plt.figure()
    im2 = plt.imshow(ez_pxz2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Lz], shape=None,
                vmax= nivel*abs(ez_pxz2).max(), vmin= -nivel*abs(ez_pxz2).max()  )

    plt.colorbar(im2) # funcionou
    plt.title(u'YEE 3D: Ez (V/m) NO PLANO XZ (T = '+str(tempomax)+' passos)')
    plt.xlabel(u'Lx (metros)',fontsize = fs1)
    plt.ylabel(u'Lz (metros)',fontsize = fs1)
    #plt.show()

#*********************************************
# GRÁFICOS 1D
# no plano xy
plt.figure()
plt.plot(x,ez_x1,'r')
plt.grid()
plt.xlabel(u'Eixos x/y/d, '+ textof)
plt.ylabel(u'Nível')
plt.title('Campos 1D: Ez plano xy (TEMPO = '+str(tempomax)+' passos)')
plt.hold(True) # permite sobrepor vários
            # gráficos com dimensões diferentes
plt.plot(y,ez_y1,'b')
plt.plot(s,ez_d1,'k-')
texto1 = u'ez_x1 ( 0°)'
texto2 = u'ez_y1 (90°)'
texto3 = u'ez_d1 (45°)'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)

#plt.show()

# no plano xz
plt.figure()
plt.plot(x,ez_x2,'r')
plt.grid()
```

```
plt.xlabel(u'Eixos x/z/d, '+ textof)
plt.ylabel(u'Nível')
plt.title('Campos 1D: Ez plano xz (TEMPO = '+str(tempomax)+' passos)')
plt.hold(True) # permite sobrepor vários
               # gráficos com dimensões diferentes

plt.plot(z,ez_z2,'b-')
plt.plot(s2,ez_d2,'k')
texto1 = u'ez_x2 ( 0°)'
texto2 = u'ez_z2 (90°)'
texto3 = u'ez_d2 (45°)'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)
#plt.show()

#**************************************************
if MOSTRAR_SOMENTE_GRAFICOS_1D == 1:
    # fonte em função do tempo
    plt.figure()
    plt.plot(tempo_teste,ez_teste,'k')
    plt.grid()
    plt.xlabel(u'Eixo Tempo, '+ textof)
    plt.ylabel(u'Nível')
    plt.title('Fonte para campo Ez ( TEMPO = '+str(tempomax)+\
            ' passos, FATOR_AT = '+str(fator_at)+' )' )
    #plt.show()

# diagrama na forma polar
if ATIVA_POLAR == 1:
    # PLANO E
    plt.figure()
    plt.polar(teta,ezp,'r')
    plt.hold(True) # permite sobrepor vários
                   # gráficos com dimensões diferentes
    plt.polar(teta2,ed2,'b')
    plt.title('Polar, raio ='+str(raio1)+' '+ textof)
    plt.legend((textop1,textop2),loc = 'lower right')
    plt.hold(False)
    # graficos para espalhamento da esfera
    plt.figure()
    plt.plot(tetar,ezpr,'k') # ou plt.semilogy
    plt.grid()
    plt.xlabel(u'Ângulo (graus)')
    plt.ylabel(u'RCS (dBsm)') # ou dBsw
    plt.title(u'Yee 3D: RCS da esfera no plano E (raio = ' +\
              str(a) + r'$\lambda$)')
    plt.hold(True) # permite sobrepor vários
                   # gráficos com dimensões diferentes
    plt.plot(ateta2,rsc_e,'r')
    #plt.plot(s,ez_d1,'k-')
    texto1 = u'Yee 3D'
    texto2 = u'Teórico'
    #texto3 = u'ez_d1 (30°)'
    plt.legend((texto1,texto2),loc = 'upper right')
    plt.hold(False)
```

```
    # PLANO H
    plt.figure()
    plt.polar(tetah,ezph,'r')
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
    plt.polar(teta2,ed2,'b')
    plt.title('Polar, raio ='+str(raio2)+' '+ textof)
    plt.legend((textop1,textop2),loc = 'lower right')
    plt.hold(False)
    # graficos para espalhamento da esfera
    plt.figure()
    plt.plot(tetarh,ezprh,'k') # ou plt.semilogy
    plt.grid()
    plt.xlabel(u'Ângulo (graus)')
    plt.ylabel(u'RCS (dBsm)') # ou dBsm
    plt.title(u'Yee 3D: RCS da esfera no plano H (raio = ' +\
              str(a) + r'$\lambda$)')
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
    plt.plot(ateta2,rsc_h,'r')
    #plt.plot(s,ez_d1,'k-')
    texto1 = u'Yee 3D'
    texto2 = u'Teórico'
    #texto3 = u'ez_d1 (30°)'
    plt.legend((texto1,texto2),loc = 'upper right')
    plt.hold(False)

plt.show()
# FINAL DO PROGRAMA
#########################################################################
```

ARQUIVO 2: MEDIDAS DE CAMPO ESPALHADO NA ESFERA METÁLICA PARA GRADE DE PRISMAS HEXAGONAIS E COMPARAÇÕES COM O MÉTODO FDTD YEE

```
# -*- coding: cp1252 -*-
#            cp1252 ou utf-8
# programa programa_fdtd_3d_h_2c_visualizacao_1d.py em python 2.7.3
# derivado do programa: programa_fdtd_3d_h_2c_visualizacao_1c.py
# AUTOR: MARINOEL JOAQUIM

#*********************************************************
# BIBLIOTECAS NECESSÁRIAS

import numpy as np
import time
import scipy.special as sp
#*********************************************************
tempo1 = time.time()

# INICIO DAS FUNÇÕES

# Função para hz e h3
def p2a(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
```

```
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx-1)
        p2 = (p1 + 1)/2
        if (p2 > PH3):
            p2 = 0
    return p2

# Função para ez e e3
def p2b(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx)
        p2 = (p1 + 1)/2
        if (p2 > PEZ):
            p2 = 0
    return p2

# Função para h2 e e1
def p2c(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx)
        p2 = (p1)/2
        if (p2 > PH2):
            p2 = 0
    return p2

# Função para h1 e e2
def p2d(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx-1)
        p2 = (p1 + 1)/2
        if (p2 > PH1):
            p2 = 0
    return p2
#**************************************

# FUNÇÃO PARA FONTE DO TIPO PULSO RICKER OU SENOIDE
def fonte_1(tempo2,posicao,tipo_fonte,fator_at):
    if tipo_fonte == 0: # pulso ricker
        arg = np.pi*( (cdtds*tempo2 - posicao)/ppw - 1.0 )
        arg2 = arg*arg
        funcao =(1.0 - 2.0*arg2)*np.exp(-arg2)

    else:  # senóide com amortecimento inicial
        periodo = ppw/cdtds # periodo da senóide
                            # ou tempo para o pico do pulso ricker
        cte_tempo = fator_at*periodo
        funcao = ( 1.0 - np.exp(-1.0*tempo2/cte_tempo) )*\
                 np.sin(2*np.pi*tempo2*cdtds/ppw)
```

```
    return funcao
#**********************************

# função para obter i,j a partir de p para campo ez
def ez_ij(p2):
    p1 = 2*p2 - 1
    j = p1/Nx + 1
    if (p1%Nx==0 and p1>=Nx):
        j = j - 1
    i = p1 - (j-1)*Nx
    return i,j
#***********************************************************

# Função de interpolação para gráfico de ez no plano xy
def pteste(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        i = 0
        j = 0
    return i, j

# Função de interpolação para gráfico de ez no plano xz
def pteste2(i,k):
    if ((i<=0) or (i>=Nx+1) or (k<=0) or (k>=Nz+1)):
        i = 0
        k = 0
    return i, k
#***********************************************************

# Função para criar grafico na forma polar
def gera_polar1(Max_polarx,porcentagemdepx, Einc, dw, dw2, pw, pw2,ez_puv):
    tamanhox = int(porcentagemdepx*pw2) # parte inicial não usada
    ez_x1polar = np.zeros((tamanhox))   # 0 a tamanhox
    ez_x1polar[:] = ez_pxy[:tamanhox,py]
    nxp = ez_x1polar.argmax()  # valor reduzido da dimensão central no eixo x
    if nxp <=10: nxp = 10
    if  MAXPOLAR_AUTOMATICO == 0:
        n = int(round((pw2 - nxp - 1)*dw2/dw)) # raio da medidas = n*dw
    else:
        n = int((pw2 - Max_polarx)*dw2/dw)

    dw = 1.0*dw # referencia principal dz ou dy: 0 < k < (n+1)
    dw2 = 1.0*dw2 # referencia dependente dx: 0 < m[k] < (n+1)*dw/dw2
    raio = n*dw
    #raio2 = int(raio)
    teta = np.zeros((4*n+1))
    ezp = np.zeros((4*n+1))
    a = np.zeros((n+1))
    b = np.zeros((n+1))
    m = np.zeros((n+1))
    tetar = np.zeros((2*n+1))
    ezpr = np.zeros((2*n+1))
    for k in range(0,n+1):
        a[k] = k*dw
        b[k] = np.sqrt(raio**2 - a[k]**2)
        m[k] = int(round(b[k]/dw2)) # modificado
```

```
        #teta[k] = np.arctan2(a[k],b[k])
        teta[k] = np.arctan2(a[k],m[k]*dw2)
        ezp[k] = abs(ez_puv[pw2 + m[k],pw + k])
    for k in range(0,n+1):
        teta[k+n] = np.pi - teta[n-k]
        ezp[k+n] = abs(ez_puv[pw2 - m[n-k],pw + n - k])
        teta[k+2*n] = np.pi + teta[k]
        ezp[k+2*n] = abs(ez_puv[pw2 - m[k],pw - k])
        teta[k+3*n] = 2*np.pi - teta[n-k]
        ezp[k+3*n] = abs(ez_puv[pw2 + m[n-k],pw - n + k])
    for k in range(0,2*n+1):
        tetar[k] = teta[k]*180/np.pi
        ezpr[k] = ezp[k]
        raioesf = 1.0*ds*ppw
        einc2 = Einc*(raio - raioesf )/(raio - raioesf +\
                        raioesf*(1.0 + np.cos(teta[k])))
        #einc2 = Einc*(np.cos(teta[k])*np.cos(fix) - np.sin(fix))
        einc2 = Einc
        raio3 = raio
        ezpr[k] = ezpr[k]*ezpr[k]*4.0*np.pi*raio3*raio3/((raioesf*einc2)**2)
        ezpr[k] = 10*np.log10(ezpr[k])
    return raio, teta, ezp, tetar, ezpr
    # Final função para gráfico polar

#*********************************************
# FINAL DAS FUNÇÕES
#*********************************************************

# INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS
Nx = 81 # numero de pontos na direção x
Ny = 141  # numero de pontos na direção y
Nz = 71 # numero de pontos na direção z
ppw = 15 # numero de pontos por comprimento de onda
tempomax = 40

tipo_fonte = 1  # 0=pulso ricker ou 1=senoide
HARD_SOFT  = 1   # fonte: 0=hard ou 1=soft
ATIVANDO_DIPOLO_1 = 0 # 1=ativa 0=desativa
ATIVANDO_DIPOLO_2 = 0 # 1=ativa 0=desativa

rdsdz = 1.0 #np.sqrt(1.0) # razão ds/dz
fator_comp = 1.4 # em comprimentos de onda para dipolo_1

arqsaida1 = "ez_pxy_1.txt"
arqsaida2 = "ez_pxz_2.txt"
fator_at = 0.5

# FINAL DAS PRINCIPAIS VARIÁVEIS

#************************************************

# VARIÁVEIS DE CONTROLE DOS GRÁFICOS

USAR_ESTES_ARQUIVOS = 0 # usar 1 ou 2, ou 0
```

```
MOSTRAR_SOMENTE_GRAFICOS_1D = 0 # 1= graficos 1D, 0 = mostra tudo

ATIVA_POLAR = 1 # 0 = desativa 1 = ativa diagrama polar
nivel = 1.0e-1 # controla nível máximo graficos 2D

ENTRADA_ARQUIVOS_ESFERA_YEE = 1 # 0=desativa ou 1=ativa
MAXPOLAR_AUTOMATICO = 1 # automatico=0 ou usar Max_polarx
Max_polarx = 45 # 54 ou 66 ou 44 PLANO XZ
 # posição do máximo no eixo x (0 a Max_polarx)
Max_polarx2 = 70             # PLANO XY
Eincx = 1.0
raio_esfera = .5
porcentagemdepx = 0.5 # para obter o maxpolar automático
#*****************************************************

# LEITURA DE ARQUIVO DE ENTRADA
if USAR_ESTES_ARQUIVOS == 0:
    arquivo3 = open("entrada3.txt","r")

if USAR_ESTES_ARQUIVOS == 1:
    arquivo3 = open("entrada_ta1.txt","r")

if USAR_ESTES_ARQUIVOS == 2:
    arquivo3 = open("entrada_ta2.txt","r")

Nx = int(arquivo3.readline())#funcionou
Ny =  int(arquivo3.readline())
Nz =  int(arquivo3.readline())
ppw =  int(arquivo3.readline())
tempomax =  int(arquivo3.readline())
tipo_fonte =  int(arquivo3.readline())
HARD_SOFT =  int(arquivo3.readline())
ds =  float(arquivo3.readline())
rdsdz =  float(arquivo3.readline())
fator_cdtds = float(arquivo3.readline())
fator_at =  float(arquivo3.readline())
delta_passo = int(arquivo3.readline())
tempomax2 =  int(arquivo3.readline())
tempomax3 =  int(arquivo3.readline())
CPML =  int(arquivo3.readline())
nxx =  int(arquivo3.readline())
Reflexao = float(arquivo3.readline())
Reflexao2 = float(arquivo3.readline())
me = float(arquivo3.readline())
me2 = float(arquivo3.readline())
numero_materiais = int(arquivo3.readline())
numero_objetos = int(arquivo3.readline())
arquivo3.close()

fator_geometria = 2
# RETIRANDO "\n" DAS STRINGS
txx1 = list(arqsaida1)
txx1.pop()
arqsaida1 = "".join(txx1)
```

```
txx2 = list(arqsaida2)
txx2.pop()
arqsaida2 = "".join(txx2)

if USAR_ESTES_ARQUIVOS == 0:
   arqsaida1 = "ez_pxy_1_730.txt"
   arqsaida2 = "ez_pxz_2_730.txt"

if USAR_ESTES_ARQUIVOS == 1:
   arqsaida1 = "ez_pxy_1_ta1.txt"
   arqsaida2 = "ez_pxz_2_ta1.txt"

if USAR_ESTES_ARQUIVOS == 2:
   arqsaida1 = "ez_pxy_1_ta2.txt"
   arqsaida2 = "ez_pxz_2_ta2.txt"

cdtds = 1.0/np.sqrt(fator_cdtds) # número de Courant

#####################################################
eo = 1.0e-9/(36*np.pi)
uo = np.pi*4e-7
co = 1.0/np.sqrt(uo*eo)
#print 'co =',co

# IMPRIME VALORES OBTIDOS DO ARQUIVO DE ENTRADA
print 'Nx =',Nx
print 'Ny =',Ny
print 'Nz =',Nz
print 'ppw =',ppw
print 'tempomax =',tempomax
print 'tipo_fonte =',tipo_fonte
print 'HARD_SOFT =',HARD_SOFT
#print 'ATIVANDO_DIPOLO_1 =',ATIVANDO_DIPOLO_1
#print 'ATIVANDO_DIPOLO_2 =',ATIVANDO_DIPOLO_2
print 'rdsdz =', rdsdz
print 'cdtds =', cdtds
#print 'fator comprimento dipolo  =',fator_comp
print 'fator atenuação (senóide) =',fator_at
#print 'fator geometria (1, 2, 3) =',fator_geometria
#print 'arqsaida1 =',arqsaida1
#print 'arqsaida2 =',arqsaida2
# FINAL DA LEITURA

#************************************

razaodsdz = round(rdsdz,2) # para gráfico
CFL = int(cdtds*100)/100.0 # número de Courant para gráfico

ativo = 1  # tipo dipolo_1
passivo = 0
comprimento_dipolo = int(fator_comp*rdsdz*ppw)
compdipolo = int(fator_comp*100)/100.0 # para gráfico
imp0 = 120.0*np.pi

#*********************************************
```

```
if (fator_geometria <= 2):
    ds = 1.0  # dimensão dos lados do hexagono maior 2D
    dx = ds*np.cos(np.pi/6) # tamanho da célula na direção x
    dy = ds/2.0   # tamanho da célula na direção y
    dz = ds/rdsdz
    Lx = (Nx)*dx  # comprimento na direção x
    Ly = (Ny)*dy  # comprimento na direção y
    Lz = (Nz)*dz  # comprimento na direção z

#*****************************************************

tipo_1d = 0 # grade 1D auxiliar
tipo_3d = 3 # grade 3D

# calculando dimensões das matrizes para os campos E e H
ix = Nx/2
rx = Nx%2
jy = Ny/2
ry = Ny%2
jy2 = (Ny-1)/2
ry2 = (Ny-1)%2

PEZ = (ix + rx)*(jy + ry) + (ix)*(jy)
PH2 = (ix)*(jy + ry) + (ix + rx)*(jy)
PH1 = (ix)*(jy2 + ry2) + (ix)*(jy2)
PH3 = PH1
#print 'PEZ= ',PEZ

# matrizes para os gráficos de Ez nos planos xy e xz
ez_pxy = np.zeros((Nx+1,Ny+1))
ez_pxz = np.zeros((Nx+1,Nz+1))

ez_pxy2 = np.zeros((Ny+1,Nx+1)) # transposta de ez_pxy
ez_pxz2 = np.zeros((Nz+1,Nx+1)) # transposta de ez_pxz

#************************************
px = Nx/2 + 1 # posição da fonte ricker
py = Ny/2 + 1
pz = Nz/2 + 1
pcentro = p2b(px,py)
#*****************************************

# textos para gráficos
texto2 =  'dim='+str(Nx)+'x'+str(Ny)+'x'+str(Nz)+\
          '  ppw='+str(ppw)+'  CFL='+str(CFL)+\
          '  ds/dz='+str(razaodsdz) + ' PMLx='+\
             str(int(nxx*dx)) +' m'

if tipo_fonte == 0:
    texto1 = u'p_ricker '
else:
    texto1 = u'senóide '
if HARD_SOFT == 0:
    texto1 = texto1 + 'hard '
```

```
else:
   texto1 = texto1 + 'soft '
if ATIVANDO_DIPOLO_1 == 1 :
   texto3 = '  dipolo '+str(compdipolo)
else:
   texto3 = ''
textof = texto1 + texto2 + texto3
#************************************

# testando a fonte sozinha no tempo
ez_teste = np.zeros((tempomax))
tempo_teste = np.zeros((tempomax))

#******************************

# INICIO DOS CÁLCULOS PARA GRADE HEXAGONAL 3D

print u'\nINICIO FDTD'
print

# inicio loop do tempo

for tempo in range(0,tempomax):

   tempo_teste[tempo] = tempo
   ez_teste[tempo] = fonte_1(tempo,0.0,tipo_fonte,fator_at)

   #print '\n',tempo+1,'/',tempomax

# final loop do tempo
#************************************************

#*****************************************

# LEITURA DAS MATRIZES PARA PLANOS XY E XZ

arquivo4 = open(arqsaida1,"r")
linha = None
for i in range(0,Nx+1):
    for j in range(0,Ny+1):
       linha = arquivo4.readline() #funcionou
       #print i
       #print j
       #print linha
       ez_pxy[i,j] = float(linha)
arquivo4.close()

arquivo5 = open(arqsaida2,"r")
#linha = None
for i in range(0,Nx+1):
    for j in range(0,Nz+1):
       linha = arquivo5.readline() #funcionou
       ez_pxz[i,j] = float(linha)
arquivo5.close()
```

```
#************************************************

# eliminando valores zero das matrizes para os gráficos de ez
for i in range(1,Nx+1): # plano xy
    for j in range(1,Ny+1):
        if (j%2==0 and i%2==1 and i<=Nx) or\
           (j%2==1 and i%2==0 and i<=Nx-1):
              inx, jnx = pteste(i,j+1)
              isx, jsx = pteste(i,j-1)
              ie, je   = pteste(i+1,j)
              iw, jw   = pteste(i-1,j)
              ez_pxy[i,j] = 0.25*(ez_pxy[inx,jnx] + ez_pxy[isx,jsx] +\
                        ez_pxy[ie,je] + ez_pxy[iw,jw])
j = py
for i in range(1,Nx+1):  # plano xz
    for k in range(1,Nz+1):
        if (j%2==0 and i%2==1 and i<=Nx) or\
           (j%2==1 and i%2==0 and i<=Nx-1):
              ie, ke   = pteste2(i+1,k)
              iw, kw   = pteste2(i-1,k)
              ez_pxz[i,k] = 0.5*(ez_pxz[ie,ke] + ez_pxz[iw,kw])

#************************************************

# gerando gráficos 1D a partir dos gráficos 2D
Nx2 = Nx - px
Ny2 = Ny - py
Nz2 = Nz - pz
ez_x1 = np.zeros((Nx2+1))
ez_y1 = np.zeros((Ny2+1))
ez_x2 = np.zeros((Nx2+1))
ez_z2 = np.zeros((Nz2+1))
x = np.zeros((Nx2+1))
y = np.zeros((Ny2+1))
z = np.zeros((Nz2+1))

for i in range(0,Nx2+1):
    x[i] = i*dx
    ez_x1[i] = ez_pxy[px+i,py] # plano xy
    ez_x2[i] = ez_pxz[px+i,pz] # plano xz

for j in range(0,Ny2+1):
    y[j] = j*dy
    ez_y1[j] = ez_pxy[px,py+j] # plano xy

for k in range(0,Nz2+1):
    z[k] = k*dz
    ez_z2[k] = ez_pxz[px,pz+k] # plano xz


if Nx < Ny: # diagonal do plano xy
    Nd = Nx2
else:
    Nd = Ny2
ez_d1 = np.zeros((Nd+1))
```

```
s = np.zeros((Nd+1))

for k in range(0,Nd+1):
    s[k] = k*ds
    ez_d1[k] = ez_pxy[px+k,py+k]

if Nx < Nz: # diagonal do plano xz
    Nd2 = Nx2
else:
    Nd2 = Nz2
ez_d2 = np.zeros((Nd2+1))
s2 = np.zeros((Nd2+1))

ds2 = np.sqrt(dx**2 + dz**2)
for k in range(0,Nd2+1):
    s2[k] = k*ds2

    ez_d2[k] = ez_pxz[px+k,pz+k]

#*****************************************
# diagrama na forma polar no plano xz do campo ez
# modificações em 05/01/2016
if ATIVA_POLAR == 1:
    # ESFERA: PLANO E (plano xz)
    pw = pz - 1
    pw2 = px - 1
    dw = dz
    dw2 = dx
    Einc = Eincx  #*4*np.pi
    raio1, teta, ezp, tetar, ezpr = gera_polar1(Max_polarx,porcentagemdepx,
                                Einc, dw, dw2, pw, pw2,ez_pxz)
    vmaxpolar = abs(ezp).max()
    ezp = ezp/vmaxpolar
    # ESFERA: PLANO H (plano xy)
    pw = py - 1
    pw2 = px - 1
    dw = dy
    dw2 = dx
    #Eincx = 1.3
    Einc = Eincx    #*4*np.pi
    raio2, tetah, ezph, tetarh, ezprh = gera_polar1(Max_polarx2,porcentagemdepx,
                                Einc, dw, dw2, pw, pw2, ez_pxy)
    vmaxpolarh = abs(ezph).max()
    ezph = ezph/vmaxpolarh

    # diagrama teorico de dipolos
    N2 = 100
    teta2 = np.zeros((N2+1))
    ed2 = np.zeros((N2+1))
    dteta = 2*np.pi/N2
    for i in range(0,N2+1):
        if ATIVANDO_DIPOLO_1 == 1:
            teta2[i] = i*dteta
            rot = np.pi/2
            ed2[i] = abs((np.cos(np.pi*fator_comp*np.cos(teta2[i]+rot)) -\
```

```
            np.cos(np.pi*fator_comp))/(np.sin(teta2[i]+rot) + 1e-8))
        textop = texto3
    else:
        teta2[i] = i*dteta
        ed2[i] = abs(np.cos(teta2[i])**1)
        textop = u'dipolo curto'
textop1 = textop + u' fdtd' # para gráfico na forma polar
textop2 = textop + u' teórico'
#****************************************

# CALCULANDO ESPALHAMENTO TEORICO DO CAMPO POR ESFERA
N = 100
Nteta = 360
fator_comp = raio_esfera
comp = 1.0 # usar 1.0 para normalizar o raio a em relação ao comp
ateta = np.zeros((Nteta + 1))
ateta2 = np.zeros((Nteta + 1))
an = np.zeros((Nteta + 1),dtype = complex)
bn = np.zeros((Nteta + 1),dtype = complex)
cn = np.zeros((Nteta + 1),dtype = complex)
Ateta = np.zeros((Nteta + 1),dtype = complex)
Afi = np.zeros((Nteta + 1),dtype = complex)
rsc_e = np.zeros((Nteta + 1))
rsc_h = np.zeros((Nteta + 1))

a = fator_comp*comp
b = 2.0*np.pi/comp
dteta = np.pi/Nteta

for n in range(1,N+1):
    an[n] = np.power(1.0j, -n)*(2.0*n+1.0)/(1.0*n*(n+1.0))
    jv = sp.jn(n+0.5,b*a)
    h2 = sp.hankel2(n+0.5,b*a)
    jvd = b*sp.jvp(n+0.5,b*a,1)
    h2d = b*sp.h2vp(n+0.5,b*a,1)
    jve = np.sqrt(np.pi*b*a/2.0)*jv
    h2e = np.sqrt(np.pi*b*a/2.0)*h2
    jved = np.sqrt(np.pi*b/(8.0*a))*jv + np.sqrt(np.pi*b*a/2.0)*jvd
    h2ed = np.sqrt(np.pi*b/(8.0*a))*h2 + np.sqrt(np.pi*b*a/2.0)*h2d
    bn[n] = -an[n]*jved/h2ed
    cn[n] = -an[n]*jve/h2e
    #print cn[n]

for i in range(0,Nteta+1):
    ateta[i] = i*dteta
    ateta2[i] = i*dteta*180.0/np.pi
    xr = np.cos(ateta[i])
    yr = np.sin(ateta[i])
    lp, lpd = sp.lpmn(1,N,xr)
    for n in range(1,N+1):
        Ateta[i] += np.power(1.0j,n)*(bn[n]*yr*lpd[1,n] -\
                cn[n]*lp[1,n]/(yr + 1e-26))
        Afi[i] += np.power(1.0j,n)*(bn[n]*lp[1,n]/(yr + 1e-26) -\
                cn[n]*yr*lpd[1,n])
    Ateta[i] = np.abs(Ateta[i])
```

```
        Ateta2 = np.abs( Ateta[i]*Ateta[i] )
        Afi[i] = np.abs(Afi[i])
        Afi2 = np.abs(Afi[i]*Afi[i])
        fi = 0.0
        co2 = np.cos(fi)*np.cos(fi)
        si2 = np.sin(fi)*np.sin(fi)
        rsc_e[i] = (comp*comp/np.pi)*(co2*Ateta2 + si2*Afi2)
        fi = np.pi/2.0
        co2 = np.cos(fi)*np.cos(fi)
        si2 = np.sin(fi)*np.sin(fi)
        rsc_h[i] =(comp*comp/np.pi)*(co2*Ateta2 + si2*Afi2)
        rsc_e[i] = 10.0*np.log10(rsc_e[i])
        rsc_h[i] = 10.0*np.log10(rsc_h[i])
#************************************************************
# ENTRADA DE ARQUIVOS PARA A ESFERA YEE 3D QUE SERÃO USADOS NA
# COMPARÇÃO COM A ESFERA HEX 3D E ESFERA TEÓRICA

if (ENTRADA_ARQUIVOS_ESFERA_YEE == 1):
    arquivox1 = open('yee_dados_esfera_T693.txt', 'r')
    tempomax_yee = int(float(arquivox1.readline()))
    raio_yee = float(arquivox1.readline())
    compE = int(float(arquivox1.readline()))
    compH = int(float(arquivox1.readline()))
    arquivox1.close()
    print 'tempomax_yee =', tempomax_yee
    ezp_yee = np.zeros((compE))
    teta_yee = np.zeros((compE))
    ezph_yee = np.zeros((compH))
    tetah_yee = np.zeros((compH))

    # PLANO E
    esf_planoE = 'yee_planoE_T' + str(tempomax_yee) + '.txt'
    arquivox1 = open(esf_planoE, 'r')
    for i in range(0,compE):
        linha = arquivox1.readline()
        ezp_yee[i] = float(linha)
    arquivox1.close()

    esf_anguloE = 'yee_anguloE_T' + str(tempomax_yee) + '.txt'
    arquivox1 = open(esf_anguloE, 'r')
    for i in range(0,compE):
        linha = arquivox1.readline()
        teta_yee[i] = float(linha)
    arquivox1.close()

    # PLANO H
    esf_planoH = 'yee_planoH_T' + str(tempomax_yee) + '.txt'
    arquivox1 = open(esf_planoH, 'r')
    for i in range(0,compH):
        linha = arquivox1.readline()
        ezph_yee[i] = float(linha)
    arquivox1.close()

    esf_anguloH = 'yee_anguloH_T' + str(tempomax_yee) + '.txt'
    arquivox1 = open(esf_anguloH, 'r')
```

```
    for i in range(0,compH):
        linha = arquivox1.readline()
        tetah_yee[i] = float(linha)
    arquivox1.close()

    # FINAL SAIDA DOS ARQUIVOS DA ESFERA YEE 3D

#****************************************
ez_pxy2 = ez_pxy.T # transposta para corrigir imagem
ez_pxz2 = ez_pxz.T # transposta para corrigir imagem

#*********************************************************

if ATIVA_POLAR == 1:
    #print 'px = ',px
    #print 'nxp = ',nxp
    #print 'n = ',n
    print 'raio1 (plano E) = ',raio1
    print 'raio2 (plano H) = ',raio2
#print'\n'
print u'FINAL'

#****************************************

# PLOTANDO CAMPOS Ez PARA O MODO 3D
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

# nearest ou bilinear - funcionou
# cm.jet ou cm.RdYlGn ou cm.winter ou cm.hot
# ficou imagem correta com shape=None e com extent=[0,Lx,0,Ly]
# mas shape inverteu eixos x e y
# no entanto usando a transposta de ez a imagem ficou com eixos corretos

# GRÁFICOS 2D
if MOSTRAR_SOMENTE_GRAFICOS_1D == 0:
    fs1 = 14
    fs2 = 15
    fse = 13
    # plano xy
    plt.figure()
    im = plt.imshow(ez_pxy2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Ly], shape=None,
                vmax= nivel*abs(ez_pxy2).max(), vmin= -nivel*abs(ez_pxy2).max()  )

    plt.colorbar(im) # funcionou
    plt.title(u'HEX 3D: Ez (V/m) NO PLANO XY (T = '+str(tempomax)+' passos)')
    plt.xlabel(u'Lx (metros)',fontsize = fs1)
    plt.ylabel(u'Ly (metros)',fontsize = fs1)
    #plt.show()

    # plano xz
    plt.figure()
    im2 = plt.imshow(ez_pxz2, interpolation='nearest', cmap=cm.jet,
```

```
                origin='lower', extent=[0,Lx,0,Lz], shape=None,
                vmax= nivel*abs(ez_pxz2).max(), vmin= -nivel*abs(ez_pxz2).max()  )

    plt.colorbar(im2) # funcionou
    plt.title(u'HEX 3D: Ez (V/m) NO PLANO XZ (T = '+str(tempomax)+' passos)')
    plt.xlabel(u'Lx (metros)',fontsize = fs1)
    plt.ylabel(u'Lz (metros)',fontsize = fs1)
    #plt.show()

#*****************************************
# GRÁFICOS 1D
# no plano xy
plt.figure()
plt.plot(x,ez_x1,'r')
plt.grid()
plt.xlabel(u'Eixos x/y/d, '+ textof)
plt.ylabel(u'Nível')
plt.title('Campos 1D: Ez plano xy (TEMPO = '+str(tempomax)+' passos)')
plt.hold(True) # permite sobrepor vários
            # gráficos com dimensões diferentes
plt.plot(y,ez_y1,'b')
plt.plot(s,ez_d1,'k-')
texto1 = u'ez_x1 ( 0°)'
texto2 = u'ez_y1 (90°)'
texto3 = u'ez_d1 (30°)'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)

#plt.show()

# no plano xz
plt.figure()
plt.plot(x,ez_x2,'r')
plt.grid()
plt.xlabel(u'Eixos x/z/d, '+ textof)
plt.ylabel(u'Nível')
plt.title('Campos 1D: Ez plano xz (TEMPO = '+str(tempomax)+' passos)')
plt.hold(True) # permite sobrepor vários
            # gráficos com dimensões diferentes

plt.plot(z,ez_z2,'b-')
plt.plot(s2,ez_d2,'k')
texto1 = u'ez_x2 ( 0°)'
texto2 = u'ez_z2 (90°)'
texto3 = u'ez_d2 (45°)'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)
#plt.show()

#*********************************************************

if MOSTRAR_SOMENTE_GRAFICOS_1D == 1:
    # fonte em função do tempo
    plt.figure()
    plt.plot(tempo_teste,ez_teste,'k')
```

```
    plt.grid()
    plt.xlabel(u'Eixo Tempo, '+ textof)
    plt.ylabel(u'Nível')
    plt.title('Fonte para campo Ez ( TEMPO = '+str(tempomax)+\
            ' passos, FATOR_AT = '+str(fator_at)+' )' )
    #plt.show()

# diagrama na forma polar e espalhamento nas esferas teorico e numerico
if ATIVA_POLAR == 1:
    # PLANO E
    plt.figure()
    plt.polar(teta,ezp,'r')
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
    plt.polar(teta2,ed2,'b')
    textop1 = u'Esfera Plano E (Hex 3D)'
    plt.title('Polar, raio ='+str(raio1)+' '+ textof)
    plt.legend((textop1,textop2),loc = 'lower right')
    plt.hold(False)
    # graficos para espalhamento da esfera
    plt.figure()
    plt.plot(tetar,ezpr,'k') # ou plt.semilogy
    plt.grid()
    plt.xlabel(u'Ângulo (graus)')
    plt.ylabel(u'RCS (dBsm)') # ou dBsw
    plt.title(u'Hex 3D: RCS da esfera no plano E (raio = '+\
              str(a) + r'$\lambda$)')
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
    plt.plot(ateta2,rsc_e,'r')
    #plt.plot(s,ez_d1,'k-')
    texto1 = u'Hex 3D'
    texto2 = u'Teórico'
    #texto3 = u'ez_d1 (30°)'
    plt.legend((texto1,texto2),loc = 'upper right')
    plt.hold(False)

    # PLANO H
    plt.figure()
    plt.polar(tetah,ezph,'r')
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
    plt.polar(teta2,ed2,'b')
    textop1 = u'Esfera Plano H (Hex 3D)'
    plt.title('Polar, raio='+str(raio2)+' '+ textof)
    plt.legend((textop1,textop2),loc = 'lower right')
    plt.hold(False)
    # graficos para espalhamento da esfera
    plt.figure()
    plt.plot(tetarh,ezprh,'k') # ou plt.semilogy
    plt.grid()
    plt.xlabel(u'Ângulo (graus)')
    plt.ylabel(u'RCS (dBsm)') # ou dBsw
    plt.title(u'Hex 3D: RCS da esfera no plano H (raio = '+\
              str(a) + r'$\lambda$)')
```

```
    plt.hold(True) # permite sobrepor vários
                # gráficos com dimensões diferentes
    plt.plot(ateta2,rsc_h,'r')
    #plt.plot(s,ez_d1,'k-')
    texto1 = u'Hex 3D'
    texto2 = u'Teórico'
    #texto3 = u'ez_d1 (30°)'
    plt.legend((texto1,texto2),loc = 'upper right')
    plt.hold(False)
    #plt.show()

    if (ENTRADA_ARQUIVOS_ESFERA_YEE == 1):
        # Plano E: graficos para espalhamento da esfera
        plt.figure()
        plt.plot(tetar,ezpr,'k') # ou plt.semilogy
        plt.grid()
        plt.xlabel(u'Ângulo (graus)')
        plt.ylabel(u'RCS (dBsm)') # ou dBsw
        plt.title(u'RCS da esfera no plano E (raio = '+\
                str(a) + r'$\lambda$)')
        plt.hold(True) # permite sobrepor vários
                    # gráficos com dimensões diferentes
        plt.plot(teta_yee,ezp_yee,'b-')
        plt.plot(ateta2,rsc_e,'r')
        texto1 = u'Hex 3D'
        texto2 = u'Yee 3D'
        texto3 = u'Teórico'
        plt.legend((texto1,texto2,texto3),loc = 'lower right')
        plt.hold(False)
        # Plano H: graficos para espalhamento da esfera
        plt.figure()
        plt.plot(tetarh,ezprh,'k') # ou plt.semilogy
        plt.grid()
        plt.xlabel(u'Ângulo (graus)')
        plt.ylabel(u'RCS (dBsm)') # ou dBsw
        plt.title(u'RCS da esfera no plano H (raio = '+\
                str(a) + r'$\lambda$)')
        plt.hold(True) # permite sobrepor vários
                    # gráficos com dimensões diferentes
        plt.plot(tetah_yee,ezph_yee,'b-')
        plt.plot(ateta2,rsc_h,'r')
        texto1 = u'Hex 3D'
        texto2 = u'Yee 3D'
        texto3 = u'Teórico'
        plt.legend((texto1,texto2,texto3),loc = 'upper right')
        plt.hold(False)
        plt.show()

plt.show()
# FINAL DO PROGRAMA
#### FINAL DO APÊNDICE 9
#######################################################################
```

APÊNDICE 10. PROGRAMAS PARA MEDIR A DISPERSÃO NUMÉRICA EM AMBOS OS MÉTODOS FDTD

ESTE APÊNDICE CONTÉM 3 ARQUIVOS

ARQUIVO 1: MEDIDAS DE DISPERSÃO PARA O MÉTODO FDTD YEE

```
# -*- coding: cp1252 -*-
#            cp1252 ou utf-8
# programa programa_fdtd_3d_h_2c_visualizacao_1d.py em python 2.7.3
# derivado do programa: programa_fdtd_3d_h_2c_visualizacao_1c.py
# AUTOR: MARINOEL JOAQUIM
#*********************************************************

# BIBLIOTECAS NECESSÁRIAS

import numpy as np
import time
import scipy.special as sp
#*********************************************************
tempo1 = time.time()

# INICIO DAS FUNÇÕES

# FUNÇÃO PARA FONTE DO TIPO PULSO RICKER OU SENOIDE
def fonte_1(tempo2,posicao,tipo_fonte,fator_at):
    if tipo_fonte == 0: # pulso ricker
        arg = np.pi*( (cdtds*tempo2 - posicao)/ppw - 1.0 )
        arg2 = arg*arg
        funcao =(1.0 - 2.0*arg2)*np.exp(-arg2)

    else:  # senóide com amortecimento inicial
        periodo = ppw/cdtds # periodo da senóide
                        # ou tempo para o pico do pulso ricker
        cte_tempo = fator_at*periodo
        funcao = ( 1.0 - np.exp(-1.0*tempo2/cte_tempo) )*\
                np.sin(2*np.pi*tempo2*cdtds/ppw)
    return funcao
#************************************

#*********************************************
# nova função copiada em 23/02/2016
def medidav2(ez_t2, Nt, limiar, dist, fator_ppw, novo_alg):
    ez_t = np.zeros((Nt+1))
    ez_t[:] = ez_t2[:]
    n_limiar = Nt # modificado
    for i in range(Nt,0,-1):
        if (ez_t[i] > limiar):
            n_limiar = i
            break
    n_limiar -= int(fator_ppw*ppw*ds/dist)
    Nlimiar = Nt - n_limiar
    eztemp = np.zeros((Nlimiar))
    for i in range(0,Nlimiar):
```

```
        eztemp[i] = ez_t[n_limiar + i]
    n_max = eztemp.argmax()
    n_max += n_limiar
    ni = n_max # modificado
    for i in range(n_max,Nt):
        if ez_t[i] <= 0.0:
            ni = i
            break
    a = (ez_t[ni] - ez_t[ni-1])/dist
    b = ez_t[ni] - a*ni*dist
    x0 = -b/a
    # novo algoritmo em 18/06/2015
    if novo_alg == 1:
        d_ang = (2.0*np.pi/ppw)*dist/ds
        c_onda = ppw*ds
        f_alfa = ((ni-1.0)/ni)*(ez_t[ni-1]/ez_t[ni])
        B = f_alfa*np.cos(d_ang)
        C = f_alfa*np.sin(d_ang)
        teta_1 = np.arctan(C/(1.0-B))
        x0 = (c_onda/2.0)*(-teta_1)/np.pi + (ni-1.0)*dist
    return x0

#***********************************************
# NOVA FUNÇÂO
def medidav3(ez_t2, Nt, limiar, dist, fator_ppw, novo_alg):
    ez_t = np.zeros((Nt+1))
    ez_t[:] = ez_t2[:]
    n_limiar = Nt # modificado
    for i in range(Nt,0,-1):
        if (ez_t[i] > limiar):
            n_limiar = i
            break
    #n_limiar -= int(fator_ppw*ppw*ds/dist) + teste_disp
    n_limiar -= int(round(fator_ppw*ppw*ds/dist )) + teste_disp
    Nlimiar = Nt - n_limiar
    eztemp = np.zeros((Nlimiar))
    N_ppw = int(0.9*ppw*ds/dist )
    for i in range(0,N_ppw):
        eztemp[i] = ez_t[n_limiar + i]
    n_max = eztemp.argmax()
    n_max += n_limiar
    ni = n_max # modificado
    for i in range(n_max,Nt):
        if ez_t[i] <= 0.0:
            ni = i
            break
    a = (ez_t[ni] - ez_t[ni-1])/dist
    b = ez_t[ni] - a*ni*dist
    x0 = -b/a
    # novo algoritmo em 18/06/2015
    if novo_alg == 1:
        d_ang = (2.0*np.pi/ppw)*dist/ds
        c_onda = ppw*ds
        f_alfa = ((ni-1.0)/ni)*(ez_t[ni-1]/ez_t[ni])
        B = f_alfa*np.cos(d_ang)
```

```
        C = f_alfa*np.sin(d_ang)
        teta_1 = np.arctan(C/(1.0 - B))
        x0 = (c_onda/2.0)*(-teta_1)/np.pi + (ni - 1.0)*dist
    return x0

#*************************************************************
def func_senoide(A2,r,fa,fb,sc,Nc,Nt,d,alfa,teta):
    c = fa*3.0e8
    dt = sc*d/c
    t = Nt*dt*fb
    fc = 1.0 # fc = -c para derivda no tempo
    co = Nc*d
    f = c/ co
    ct = alfa/f # com fator de atenuação: alfa
    k = 2.0*np.pi/co
    #fs2 = (A2*k/r )*np.cos(2.0*pi*f*t - r*k)
    fs2 = ( -A2/(c*ct*r) )*np.exp((-t + r/c)/ct)*\
       np.sin(2.0*np.pi*f*t - r*k + teta) + ( -A2*k/r )*( 1.0 -\
       np.exp((-t + r/c)/ct) )*np.cos(2.0*np.pi*f*t - r*k + teta)
    return fs2

def func_senoide2(A2,r,fa,fb,sc,Nc,Nt,d,alfa,teta):
    c = fa*3.0e8
    dt = sc*d/c
    t = Nt*dt*fb
    alfa2 = alfa
    fc = 1.0 # fc = -c para derivda no tempo
    co = Nc*d
    f = c/ co
    k = 2.0*np.pi/co
    fs2 = (-A2*k/r )*np.cos(2.0*np.pi*f*t - r*k + teta)
    return fs2

#*********************************************
# FINAL DAS FUNÇÕES
#*************************************************************

# INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS
Nx = 81 # numero de pontos na direção x
Ny = 141  # numero de pontos na direção y
Nz = 71 # numero de pontos na direção z
ppw = 15 # numero de pontos por comprimento de onda
tempomax = 40

tipo_fonte = 1  # 0=pulso ricker ou 1=senoide
HARD_SOFT  = 1   # fonte: 0=hard ou 1=soft
ATIVANDO_DIPOLO_1 = 0 # 1=ativa 0=desativa
ATIVANDO_DIPOLO_2 = 0 # 1=ativa 0=desativa

rdsdz = 1.0 #np.sqrt(1.0) # razão ds/dz
fator_comp = 1.4 # em comprimentos de onda para dipolo_1

arqsaida1 = "ez_pxy_1.txt"
arqsaida2 = "ez_pxz_2.txt"
fator_at = 0.5
```

```
# FINAL DAS PRINCIPAIS VARIÁVEIS

#************************************************

# VARIÁVEIS DE CONTROLE DOS GRÁFICOS

USAR_ESTES_ARQUIVOS = 0 # usar 1 ou 2, ou 0

MOSTRAR_SOMENTE_GRAFICOS_1D = 0 # 1= graficos 1D, 0 = mostra tudo

nivel = 1e-4  # controla nível máximo graficos 2D

MAXPOLAR_AUTOMATICO = 1 # automatico=0 ou usar Max_polarx
Max_polarx = 38 # posição do máximo no eixo x (0 a Max_polarx)
Max_polarx2 = 38
fator_raio = 0.75
eincidente = .45
porcentagemdepx = 0.3# para obter o maxpolar automático
#*****************************************************

# COPIADO E APERFEIÇOADO EM 23/02/2016
# NESTA VERSÃO NOVA APROXIMAÇÃO SENOIDAL NA FUNÇÃO medidav2 EM 18/06/2015
novo_alg            =   1     # 1 = USA NOVO ALGORITMO
NP                  =  21
DELTA_COMP1         =  14
teste_disp          = 0
#xteste             = -1.5 + 0.*dxt

NUMERO_COMPRIMENTOS1   = NP - 0.  # x1
NUMERO_COMPRIMENTOS1Y  = NP - 0. # y1
NUMERO_COMPRIMENTOS1B  = NP  + 0.  # d1
NUMERO_COMPRIMENTOS1C  = NP  - 1.   #  a1 (senoide analitica)

LIMIAR1  = 5e-7   # x1
LIMIAR1Y = 5e-7  # y1
LIMIAR1D = 5e-7   # d1
LIMIAR1A = 5e-7   # a1
###################ARQUIVOS T381 a=156.985965961; d=33.8131474598
angulo_inicial_senoide = 33.8131474598
amplitude_inicial_senoide = 0.14
comprimento_seno_usado = 216.0

CURVA_DISPERSAO = 1 # 0 = desativa; 1 = ativa
CURVA_DISPERSAO2 = 1 # 0 = desativa; 1 = ativa
ajuste_x = 0.
ajuste_y = 0.
ajuste_d = 0.
ajuste_a = -1.
er = 1.0e-25
AJUSTE_CURVA_ANALITICA = 1 # 0 = desativa; 1 = ativa
INICIO_DELTA_COMP1 = 5
REDUCAO_DISTANCIA_DIPERSAO = 1 # 0 = desativa; 1 = ativa
```

```
#*************************************************

# LEITURA DE ARQUIVO DE ENTRADA
if USAR_ESTES_ARQUIVOS == 0:
    arquivo3 = open("entrada4.txt","r")

if USAR_ESTES_ARQUIVOS == 1:
    arquivo3 = open("entrada_ta1.txt","r")

if USAR_ESTES_ARQUIVOS == 2:
    arquivo3 = open("entrada_ta2.txt","r")

Nx = int(arquivo3.readline())#funcionou
Ny =  int(arquivo3.readline())
Nz =  int(arquivo3.readline())
ppw =  int(arquivo3.readline())
tempomax =  int(arquivo3.readline())
tipo_fonte =  int(arquivo3.readline())
HARD_SOFT =  int(arquivo3.readline())
ds =  float(arquivo3.readline())
rdsdz =  float(arquivo3.readline())
fator_cdtds = float(arquivo3.readline())
fator_at =  float(arquivo3.readline())
delta_passo = int(arquivo3.readline())
tempomax2 =  int(arquivo3.readline())
tempomax3 =  int(arquivo3.readline())
CPML =  int(arquivo3.readline())
nxx =  int(arquivo3.readline())
Reflexao = float(arquivo3.readline())
Reflexao2 = float(arquivo3.readline())
me = float(arquivo3.readline())
me2 = float(arquivo3.readline())
numero_materiais = int(arquivo3.readline())
numero_objetos = int(arquivo3.readline())
arquivo3.close()

fator_geometria = 2
# RETIRANDO "\n" DAS STRINGS
txx1 = list(arqsaida1)
txx1.pop()
arqsaida1 = "".join(txx1)

txx2 = list(arqsaida2)
txx2.pop()
arqsaida2 = "".join(txx2)

if USAR_ESTES_ARQUIVOS == 0:
    arqsaida1 = "ez_pxy_yee1_381d.txt"
    arqsaida2 = "ez_pxz_yee2_381d.txt"

if USAR_ESTES_ARQUIVOS == 1:
    arqsaida1 = "ez_pxy_1_ta1.txt"
    arqsaida2 = "ez_pxz_2_ta1.txt"

if USAR_ESTES_ARQUIVOS == 2:
```

```
    arqsaida1 = "ez_pxy_1_ta2.txt"
    arqsaida2 = "ez_pxz_2_ta2.txt"

cdtds = 1.0/np.sqrt(fator_cdtds) # número de Courant

#####################################################
eo = 1.0e-9/(36*np.pi)
uo = np.pi*4e-7
co = 1.0/np.sqrt(uo*eo)
#print 'co =',co

# IMPRIME VALORES OBTIDOS DO ARQUIVO DE ENTRADA
print 'Nx =',Nx
print 'Ny =',Ny
print 'Nz =',Nz
print 'ppw =',ppw
print 'tempomax =',tempomax
print 'tipo_fonte =',tipo_fonte
print 'HARD_SOFT =',HARD_SOFT
print 'fator_cdtds =',fator_cdtds
print 'cdtds =', cdtds
print 'rdsdz =', rdsdz
#print 'fator comprimento dipolo  =',fator_comp
print 'fator atenuação (senóide) =',fator_at
print

#**************************************

razaodsdz = round(rdsdz,2) # para gráfico
CFL = int(cdtds*100)/100.0 # número de Courant para gráfico

ativo = 1  # tipo dipolo_1
passivo = 0
comprimento_dipolo = int(fator_comp*rdsdz*ppw)
compdipolo = int(fator_comp*100)/100.0 # para gráfico
imp0 = 120.0*np.pi

#**********************************************

if (fator_geometria <= 2):
    ds = 1.0  # dimensão dos lados do hexagono maior 2D
    dx = ds # tamanho da célula na direção x
    dy = ds   # tamanho da célula na direção y
    dz = ds
    Lx = (Nx)*dx  # comprimento na direção x
    Ly = (Ny)*dy  # comprimento na direção y
    Lz = (Nz-1)*dz  # comprimento na direção z

#****************************************************

tipo_1d = 0 # grade 1D auxiliar
tipo_3d = 3 # grade 3D

# calculando dimensões das matrizes para os campos E e H
ix = Nx/2
```

```
rx = Nx%2
jy = Ny/2
ry = Ny%2
jy2 = (Ny-1)/2
ry2 = (Ny-1)%2

PEZ = (ix + rx)*(jy + ry) + (ix)*(jy)
PH2 = (ix)*(jy + ry) + (ix + rx)*(jy)
PH1 = (ix)*(jy2 + ry2) + (ix)*(jy2)
PH3 = PH1
#print 'PEZ= ',PEZ

# matrizes para os gráficos de Ez nos planos xy e xz
ez_pxy = np.zeros((Nx,Ny)) # modificado para YEE 3D
ez_pxz = np.zeros((Nx,Nz-1))

ez_pxy2 = np.zeros((Ny,Nx)) # transposta de ez_pxy
ez_pxz2 = np.zeros((Nz-1,Nx)) # transposta de ez_pxz

#************************************
px = Nx/2 + 1 # posição da fonte ricker
py = Ny/2 + 1
pz = Nz/2 + 1
pcentro = p2b(px,py)
#*****************************************

# textos para gráficos
texto2 =  'dim='+str(Nx)+'x'+str(Ny)+'x'+str(Nz)+\
          '  ppw='+str(ppw)+'  CFL='+str(CFL)+\
          ' dx=dy=dz='+str(ds) + ' PML='+ str(int(nxx*dx)) + ' m'
if tipo_fonte == 0:
   texto1 = u'pulso '
else:
   texto1 = u'seno '
if HARD_SOFT == 0:
   texto1 = texto1 + 'hard '
else:
   texto1 = texto1 + 'soft '
textof = texto1 + texto2 + texto3
#************************************

# testando a fonte sozinha no tempo
ez_teste = np.zeros((tempomax))
tempo_teste = np.zeros((tempomax))

#*******************************

# INICIO DOS CÁLCULOS PARA GRADE HEXAGONAL 3D

#print u'\nINICIO FDTD'
#print

# inicio loop do tempo

for tempo in range(0,tempomax):
```

```
    tempo_teste[tempo] = tempo
    ez_teste[tempo] = fonte_1(tempo,0.0,tipo_fonte,fator_at)

    #print '\n',tempo+1,'/',tempomax

# final loop do tempo
#************************************************

#*****************************************

# LEITURA DAS MATRIZES PARA PLANOS XY E XZ
arquivo4 = open(arqsaida1,"r")
#linha = None
for i in range(0,Nx): # modificado para YEE 3D
     for j in range(0,Ny):
        linha = arquivo4.readline() #funcionou
        ez_pxy[i,j] = float(linha)
arquivo4.close()

arquivo5 = open(arqsaida2,"r")
#linha = None
for i in range(0,Nx): # modificado para YEE 3D
     for j in range(0,Nz-1):
        linha = arquivo5.readline() #funcionou
        ez_pxz[i,j] = float(linha)
arquivo5.close()

#*************************************************

# gerando gráficos 1D a partir dos gráficos 2D
Nx2 = Nx - px
Ny2 = Ny - py
Nz2 = Nz - 1 - pz
ez_x1 = np.zeros((Nx2+1))
ez_y1 = np.zeros((Ny2+1))
ez_x2 = np.zeros((Nx2+1))
ez_z2 = np.zeros((Nz2+1))
x = np.zeros((Nx2+1))
y = np.zeros((Ny2+1))
z = np.zeros((Nz2+1))

for i in range(0,Nx2):
    x[i] = i*dx
    ez_x1[i] = ez_pxy[px+i,py] # plano xy
    ez_x2[i] = ez_pxz[px+i,pz] # plano xz

for j in range(0,Ny2):
    y[j] = j*dy
    ez_y1[j] = ez_pxy[px,py+j] # plano xy

for k in range(0,Nz2):
    z[k] = k*dz
    ez_z2[k] = ez_pxz[px,pz+k] # plano xz
```

```
if Nx < Ny: # diagonal do plano xy
   Nd = Nx2
else:
   Nd = Ny2
ez_d1 = np.zeros((Nd+1))
s = np.zeros((Nd+1))
ds3 = np.sqrt(dx**2 + dy**2) # alterando para Yee 3D
for k in range(0,Nd):
   s[k] = k*ds3
   ez_d1[k] = ez_pxy[px+k,py+k]

if Nx < Nz: # diagonal do plano xz
   Nd2 = Nx2
else:
   Nd2 = Nz2
ez_d2 = np.zeros((Nd2+1))
s2 = np.zeros((Nd2+1))

ds2 = np.sqrt(dx**2 + dz**2)
for k in range(0,Nd2):
   s2[k] = k*ds2

   ez_d2[k] = ez_pxz[px+k,pz+k]
#*************************************************************

ez_pxy2 = ez_pxy.T # transposta para corrigir imagem
ez_pxz2 = ez_pxz.T # transposta para corrigir imagem

#*********************************************************

# ENTRADA DE VARIÁVEIS PARA FUNÇAO ANALITICA - SENOIDE
N = 900 # para definir resolução espacial

fr = np.zeros((N))
fsa = np.zeros((N))
fsa2 = np.zeros((N))
r = np.zeros((N))
pi = np.pi

Ar = 0.25
Nt = tempomax    # passos no tempo
fa = 1.0    # não altera forma de onda
fb = 1.0     # fb = 0.8 para pulso
          # fb = 1.0 para senoide
A = amplitude_inicial_senoide   #0.11 # amplitude da senoide na direção x
tetas = angulo_inicial_senoide
teta1 = tetas*np.pi/180 # fase inicial

A2 = -0.0196    # amplitude da senoide na direção z
fb2 = 1.0
teta2 = 160*np.pi/180 # fase inicial

alfa = fator_at   # fator de atenuação da cossenoide
L = 350.0
```

```
c = fa*3.0e8
sc = 1.0/np.sqrt(fator_cdtds) # numero de Courant
Nc = ppw   # numero de pontos por comprimento de onda
d = 1.0     # tamanho do lado do hexagono grande
dt = sc*d/c
t = Nt*dt*fb
fc = 1.0        # fc = -c para derivda no tempo
comp = Nc*d
f = c/ comp
ro = t*c
cdr = 1.1 #/np.sqrt(3.0)
dr = ro/N #  1.0 para senoide ou >1.0 para pulso
# 118.0 é um valor empirico; maiores valores a senoide analitica
# aumenta a sua amplitude e é um pouco maior que ro=115.47
dr = comprimento_seno_usado/N
ro2 = N*dr

for i in range(1,N):
    r[i] = i*dr
    r2 = i*dr
    fsa[i] = func_senoide(A,r2,fa,fb,sc,Nc,Nt,d,alfa,teta1)
    fsa2[i] = func_senoide(A2,r2,fa,fb2,sc,Nc,Nt,d,alfa,teta2)
    fr[i] = Ar/(r2*r2)
# FINAL SENOIDE ANALITICA
#****************************************
#*********************************************************
# COPIADO EM 23/02/2016
# NOVA FORMA DE MEDIR ANISOTROPIA
# NOVA FORMA DE MEDIDA DE ANISOTROPIA NO PLANO XY
# achando pontos de nulos para medir anisotropia (plano xy)
if tipo_fonte == 1:
    limiar = LIMIAR1
    limiary = LIMIAR1Y
    limiard = LIMIAR1D
    limiara = LIMIAR1A
    ncomp = 1.0*NUMERO_COMPRIMENTOS1 # numero de comprimentos de onda
    ncompy = 1.0*NUMERO_COMPRIMENTOS1Y
    ncompB = 1.0* NUMERO_COMPRIMENTOS1B
    ncompC = 1.0* NUMERO_COMPRIMENTOS1C

    x1_limiar = medidav2(ez_x1,Nx2,limiar,dx,ncomp,novo_alg)
    y1_limiar = medidav2(ez_y1,Ny2,limiary,dy,ncompy,novo_alg)
    d1_limiar = medidav2(ez_d1,Nd,limiard,ds3,ncompB,novo_alg) # usado ds3
    a1_limiar = medidav2(fsa,N-1,limiara,dr,ncompC,novo_alg) #senoide analitica

    dt_limiar = tempomax
    print u'# 1° PONTOS DE NULOS - PLANO XY'
    print u'# [x1_limiar, y1_limiar, d1_limiar, a1_limiar,tempomax, ppw, HARD_SOFT]'
    print "{0:2}{1:>24.20f}{2:2}".format("[",x1_limiar,",")
    print "{0:>24.20f}{1:2}".format(y1_limiar,",")
    print "{0:>24.20f}{1:2}".format(d1_limiar,",")
    print "{0:>24.20f}{1:2}".format(a1_limiar,",")
    print "{0:>24.20f}{1:2}".format(dt_limiar,",")
    print "{0:>24.20f}{1:2}{2:2d}{3:2}".format(ppw,",",HARD_SOFT,"]")
    print
```

```
    limiar = LIMIAR1
    ncomp = 1.0*NUMERO_COMPRIMENTOS1 - DELTA_COMP1
    ncompy = 1.0*NUMERO_COMPRIMENTOS1Y - DELTA_COMP1
    ncompB = 1.0*NUMERO_COMPRIMENTOS1B - DELTA_COMP1
    ncompC = 1.0*NUMERO_COMPRIMENTOS1C - DELTA_COMP1

    x1_limiar2 = medidav2(ez_x1,Nx2,limiar,dx,ncomp,novo_alg)
    y1_limiar2 = medidav2(ez_y1,Ny2,limiary,dy,ncompy,novo_alg)
    d1_limiar2 = medidav2(ez_d1,Nd,limiard,ds3,ncompB,novo_alg) # usado ds3
    a1_limiar2 = medidav2(fsa,N-1,limiara,dr,ncompC,novo_alg) #senoide analitica

    dt_limiar = tempomax
    print u'# 2° PONTOS DE NULOS - PLANO XY'
    print u'# [x1_limiar, y1_limiar, d1_limiar, a1_limiar,tempomax, ppw, HARD_SOFT]'
    print "{0:2}{1:>24.20f}{2:2}".format("[",x1_limiar2,",")
    print "{0:>24.20f}{1:2}".format(y1_limiar2,",")
    print "{0:>24.20f}{1:2}".format(d1_limiar2,",")
    print "{0:>24.20f}{1:2}".format(a1_limiar2,",")
    print "{0:>24.20f}{1:2}".format(dt_limiar,",")
    print "{0:>24.20f}{1:2}{2:2d}{3:2}".format(ppw,",",HARD_SOFT,"]")
    print
    avy1 = 100.0*( y1_limiar2 - y1_limiar - (x1_limiar2 -\
            x1_limiar))/(x1_limiar2 - x1_limiar)
    avd1 = 100.0*( d1_limiar2 - d1_limiar - (x1_limiar2 -\
            x1_limiar))/(x1_limiar2 - x1_limiar)
    ahex = ((np.pi/ppw)**4)*(100.0/360)
    ayee = ((np.pi/ppw)**2)*(100.0/12)
    dispx = (x1_limiar2/a1_limiar2 - 1.0)*100 # calcula dispersão
    dispy = (y1_limiar2/a1_limiar2 - 1.0)*100
    dispd = (d1_limiar2/a1_limiar2 - 1.0)*100
    print u'RESULTADOS DE ANISOTROPIA E DISPERSÂO (YEE 3D)'
    print u'TIPO_INTERPOLAÇÃO (LINEAR=0; SENO=1) =',novo_alg
    print 'NUMERO_COMPRIMENTOS1 =',NUMERO_COMPRIMENTOS1
    print 'DELTA_COMPRIMENTOS1  =',DELTA_COMP1
    print 'PLANO XY'
    print 'avy1(%)  =',avy1
    print 'avd1(%)  =',avd1
    print 'dispx(%) =',dispx
    print 'dispy(%) =',dispy
    print 'dispd(%) =',dispd
    print 'ahex(%)  =',ahex
    print 'ayee(%)  =',ayee
    print u'Nível senóide analítica  =',A
    print u'Ângulo senóide (graus)   =',tetas
    print u'Comprimento seno (usado) =',ro2
    print u'Comprimento seno (max)   =',ro
    print
# FINAL MEDIDAS ANISOTROPIA E DISPERSÃO
#**************************************
# CALCULANDO CURVAS DE DISPERSÃO EM FUNÇÃO DA DISTÂNCIA
# (PONTOS DE NULOS)
if (CURVA_DISPERSAO == 1):
    x1_lim = np.zeros((DELTA_COMP1 + 1))
    y1_lim = np.zeros((DELTA_COMP1 + 1))
    d1_lim = np.zeros((DELTA_COMP1 + 1))
```

```
    a1_lim = np.zeros((DELTA_COMP1 + 1))
    cdispx = np.zeros((DELTA_COMP1 + 1))
    cdispy = np.zeros((DELTA_COMP1 + 1))
    cdispd = np.zeros((DELTA_COMP1 + 1))
    for i in range(0, DELTA_COMP1 + 1):
        NP2 = NP - i
        limiar = LIMIAR1
        limiary = LIMIAR1Y
        limiard = LIMIAR1D
        limiara = LIMIAR1A
        ncomp =  1.0*NP2 + ajuste_x
        ncompy = 1.0*NP2 + ajuste_y
        ncompB = 1.0*NP2 + ajuste_d
        ncompC = 1.0*NP2 + ajuste_a # senoide analitica

        x1_lim[i] = medidav2(ez_x1,Nx2,limiar,dx,ncomp,novo_alg)
        y1_lim[i] = medidav2(ez_y1,Ny2,limiary,dy,ncompy,novo_alg)
        d1_lim[i] = medidav2(ez_d1,Nd,limiard,ds3,ncompB,novo_alg)
        a1_lim[i] = medidav2(fsa,N-1,limiara,dr,ncompC,novo_alg)
        cdispx[i] = (x1_lim[i]/a1_lim[i] - 1.0)*100 # calcula dispersão
        cdispy[i] = (y1_lim[i]/a1_lim[i] - 1.0)*100
        cdispd[i] = (d1_lim[i]/a1_lim[i] - 1.0)*100
# FINAL DAS 3 CURVAS DE DISPERSÃO
# CURVAS DE DISPERSÃO COM AJUSTE ZERO PARA O INICIO DAS CURVAS
delta_dispx = a1_lim[0] - x1_lim[0] # ajusta para dispersão zero
# NOVA CURVA ANALITICA COM OS AJUSTES
teta1b = teta1 - delta_dispx*2.0*np.pi/(1.0*ppw*ds)
for i in range(1,N):
    r2 = i*dr
    #r[i] = r[i] - 0.0*delta_dispx
    fsa2[i] = func_senoide(A,r2,fa,fb,sc,Nc,Nt,d,alfa,teta1b)

if (CURVA_DISPERSAO2 == 1):
    x1_lim2 = np.zeros((DELTA_COMP1 + 1))
    y1_lim2 = np.zeros((DELTA_COMP1 + 1))
    d1_lim2 = np.zeros((DELTA_COMP1 + 1))
    a1_lim2 = np.zeros((DELTA_COMP1 + 1))
    cdispx2 = np.zeros((DELTA_COMP1 + 1))
    cdispy2 = np.zeros((DELTA_COMP1 + 1))
    cdispd2 = np.zeros((DELTA_COMP1 + 1))
    for i in range(0, DELTA_COMP1 + 1):
        NP2 = 1.0*NP - 1.0*i
        limiar = LIMIAR1
        limiary = LIMIAR1Y
        limiard = LIMIAR1D
        limiara = LIMIAR1A
        ncomp =  1.0*NP2 + ajuste_x
        ncompy = 1.0*NP2 + ajuste_y
        ncompB = 1.0*NP2 + ajuste_d
        ncompC = 1.0*NP2 + ajuste_a # senoide analitica

        x1_lim2[i] = medidav2(ez_x1,Nx2,limiar,dx,ncomp,novo_alg)
        y1_lim2[i] = medidav2(ez_y1,Ny2,limiary,dy,ncompy,novo_alg)
        d1_lim2[i] = medidav2(ez_d1,Nd,limiard,ds3,ncompB,novo_alg)
        if AJUSTE_CURVA_ANALITICA == 1:
```

```
            a1_lim2[i] = medidav2(fsa2,N-1,limiara,dr,ncompC,novo_alg)
        else:
            a1_lim2[i] = medidav2(fsa,N-1,limiara,dr,ncompC,novo_alg) -\
                  delta_dispx

        cdispx2[i] = (x1_lim2[i]/a1_lim2[i] - 1.0)*100 # calcula dispersão
        cdispy2[i] = (y1_lim2[i]/a1_lim2[i] - 1.0)*100
        cdispd2[i] = (d1_lim2[i]/a1_lim2[i] - 1.0)*100
    if REDUCAO_DISTANCIA_DIPERSAO == 1:
        for i in range(0, DELTA_COMP1 + 1):
            cdispx2[i] = ((x1_lim2[i]-x1_lim2[0]+er)/(a1_lim2[i]-a1_lim2[0]+er) - 1.0)*100
            cdispy2[i] = ((y1_lim2[i]-y1_lim2[0]+er)/(a1_lim2[i]-a1_lim2[0]+er) - 1.0)*100
            cdispd2[i] = ((d1_lim2[i]-d1_lim2[0]+er)/(a1_lim2[i]-a1_lim2[0]+er) - 1.0)*100

# FINAL DAS 3 CURVAS DE DISPERSÃO

# CALCULANDO AS DISPERSÕES MÉDIAS
cdispx2_med = 0.0
cdispy2_med = 0.0
cdispd2_med = 0.0

for i in range(INICIO_DELTA_COMP1, DELTA_COMP1 + 1):
    cdispx2_med += cdispx2[i]
    cdispy2_med += cdispy2[i]
    cdispd2_med += cdispd2[i]
cdispx2_med /= (DELTA_COMP1 + 1 - INICIO_DELTA_COMP1)
cdispy2_med /= (DELTA_COMP1 + 1 - INICIO_DELTA_COMP1)
cdispd2_med /= (DELTA_COMP1 + 1 - INICIO_DELTA_COMP1)
NUMERO_PONTOS = DELTA_COMP1 + 1 - INICIO_DELTA_COMP1
teta_final = teta1b*180.0/np.pi

print u'AJUSTE_CURVA_ANALITICA     =',AJUSTE_CURVA_ANALITICA
print u'Ângulo final (graus)       =',teta_final
print u'REDUCAO_DISTANCIA_DIPERSAO =',REDUCAO_DISTANCIA_DIPERSAO
print u'NUMERO_PONTOS     =',NUMERO_PONTOS
print u'dispersão média x =',cdispx2_med
print u'dispersão média y =',cdispy2_med
print u'dispersão média d =',cdispd2_med
print u'inicio =',INICIO_DELTA_COMP1
print u'cdispx2[inicio] = ',cdispx2[INICIO_DELTA_COMP1]
print u'cdispx2[final] = ',cdispx2[DELTA_COMP1]
print u'cdispy2[final] = ',cdispy2[DELTA_COMP1]
print u'cdispd2[final] = ',cdispd2[DELTA_COMP1]
# FINAL CALCULO DAS DISPERSÕES MÉDIAS

#*****************************************

# PLOTANDO CAMPOS Ez PARA O MODO 3D
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

# nearest ou bilinear - funcionou
# cm.jet ou cm.RdYlGn ou cm.winter ou cm.hot
# ficou imagem correta com shape=None e com extent=[0,Lx,0,Ly]
```

```
# mas shape inverteu eixos x e y
# no entanto usando a transposta de ez a imagem ficou com eixos corretos
fs1 = 16
fs2 = 17
fs3 = 17
fse = 10
fse2 = 14
fse3 = 14
# GRÁFICOS 2D
if MOSTRAR_SOMENTE_GRAFICOS_1D == 0:
    # plano xy
    plt.figure()
    im = plt.imshow(ez_pxy2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Ly], shape=None,
                vmax= nivel*abs(ez_pxy2).max(), vmin= -nivel*abs(ez_pxy2).max()  )

    plt.colorbar(im) # funcionou
    plt.title(u'YEE 3D: Ez (V/m) NO PLANO XY (T = '+\
            str(tempomax)+' passos)',fontsize = fs1)
    plt.xlabel(u'Lx (m)',fontsize = fs1)
    plt.ylabel(u'Ly (m)',fontsize = fs1)
    #plt.show()

    # plano xz
    plt.figure()
    plt.tick_params(labelsize = fse)
    im2 = plt.imshow(ez_pxz2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Lz], shape=None,
                vmax= nivel*abs(ez_pxz2).max(), vmin= -nivel*abs(ez_pxz2).max()  )

    plt.colorbar(im2) # funcionou
    plt.title(u'YEE 3D: Ez (V/m) NO PLANO XZ (T = '+\
                    str(tempomax)+' passos)',fontsize = fs1)
    plt.xlabel(u'Lx (m) ',fontsize = fs1)
    plt.ylabel(u'Lz (m)',fontsize = fs1)
    #plt.show()

#*******************************************
# GRÁFICOS 1D
# no plano xy
plt.figure()
plt.tick_params(labelsize = fse2)
plt.plot(x,ez_x1,'r')
plt.axis([0.,230.,-.007,.01])
plt.grid()
plt.xlabel(u'Eixos x/y/d (metros)',fontsize = fs2)
plt.ylabel(u'Amplitude (V/m)',fontsize = fs2)
plt.title('YEE 3D: CAMPOS Ez NO PLANO XY (TEMPO = '+\
              str(tempomax)+' passos)',fontsize = fs2)
plt.hold(True) # permite sobrepor vários
             # gráficos com dimensões diferentes
plt.plot(y,ez_y1,'b')
#plt.plot(s,ez_d1,'b') #teste
plt.plot(s,ez_d1,'k')
if AJUSTE_CURVA_ANALITICA == 1:
```

```
    plt.plot(r,fsa2,'g--')
else:
    plt.plot(r,fsa,'y')
texto1 = u'ez_x1 ( 0°)'
texto2 = u'ez_y1 (90°)'
texto3 = u'ez_d1 (45°)'
texto4 = u'Teórico'
plt.legend((texto1,texto2,texto3,texto4),loc = 'upper right')
plt.hold(False)

#plt.show()

# no plano xz
plt.figure()
plt.plot(x,ez_x2,'r')
plt.grid()
plt.xlabel(u'Eixos x/z/d, '+ textof)
plt.ylabel(u'Nível')
plt.title('Campos 1D: Ez plano xz (TEMPO = '+str(tempomax)+' passos)')
plt.hold(True) # permite sobrepor vários
               # gráficos com dimensões diferentes

plt.plot(z,ez_z2,'b-')
plt.plot(s2,ez_d2,'k')
texto1 = u'ez_x2 ( 0°)'
texto2 = u'ez_z2 (90°)'
texto3 = u'ez_d2 (45°)'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)
#plt.show()
#*****************************************************************

# PLOTANDO CURVAS DE DISPERSÃO NO PLANO XY
if (CURVA_DISPERSAO == 1):
    plt.figure()
    plt.tick_params(labelsize = fse3)
    plt.plot(x1_lim,cdispx,'r.-')
    #plt.axis([0.,180.,-.008,.008])
    plt.grid()
    plt.xlabel(u'Eixos x/y/d (metros)',fontsize = fs3)
    plt.ylabel(u'Dispersão (%)',fontsize = fs3)
    plt.title(u'YEE 3D: Curvas de dispersão no plano xy (MODO 1)',fontsize = fs3)
    plt.hold(True) # permite sobrepor vários
                   # gráficos com dimensões diferentes
    plt.plot(y1_lim,cdispy,'b')
    plt.plot(d1_lim,cdispd,'k-')
    texto1 = u'x_( 0°)'
    texto2 = u'y_(90°)'
    texto3 = u'd_(45°)'
    plt.legend((texto1,texto2,texto3),loc = 'upper right')
    plt.hold(False)
    #plt.show()
if (CURVA_DISPERSAO2 == 1):
    plt.figure()
    plt.tick_params(labelsize = fse3)
```

```
    plt.plot(x1_lim2,cdispx2,'r.-')
    #plt.plot(y1_lim2,cdispy2,'r')
    plt.grid()
    plt.xlabel(u'Eixos x/y/d (metros)',fontsize = fs3)
    plt.ylabel(u'Dispersão (%)',fontsize = fs3)
    plt.title(u'YEE 3D: Curvas de dispersão no plano xy (MODO 2)',fontsize = fs3)
    plt.hold(True) # permite sobrepor vários
                   # gráficos com dimensões diferentes
    plt.plot(y1_lim2,cdispy2,'b-')
    plt.plot(d1_lim2,cdispd2,'k-')
    texto1 = u'x_( 0°)'
    texto2 = u'y_(90°)'
    texto3 = u'd_(45°)'
    plt.legend((texto1,texto2,texto3),loc = 'upper right')
    plt.hold(False)
    #plt.show()

#*********************************************************

if MOSTRAR_SOMENTE_GRAFICOS_1D == 1:
    # fonte em função do tempo
    plt.figure()
    plt.plot(tempo_teste,ez_teste,'k')
    plt.grid()
    plt.xlabel(u'Eixo Tempo, '+ textof)
    plt.ylabel(u'Nível')
    plt.title('Fonte para campo Ez ( TEMPO = '+str(tempomax)+\
            ' passos, FATOR_AT = '+str(fator_at)+' )' )
    #plt.show()
plt.show()

# FINAL DO PROGRAMA
#################################################################
```

ARQUIVO 2: MEDIDAS DE DISPERSÃO PARA O MÉTODO FDTD COM GRADE DE PRISMAS HEXAGONAIS

```
# -*- coding: cp1252 -*-
#            cp1252 ou utf-8
# programa programa_fdtd_3d_h_2c_visualizacao_1d.py em python 2.7.3
# derivado do programa: programa_fdtd_3d_h_2c_visualizacao_1c.py
# AUTOR: MARINOEL JOAQUIM
#*********************************************************

# BIBLIOTECAS NECESSÁRIAS

import numpy as np
import time

#*********************************************************
tempo1 = time.time()

# INICIO DAS FUNÇÕES

# Função para hz e h3
```

```
def p2a(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx-1)
        p2 = (p1 + 1)/2
        if (p2 > PH3):
            p2 = 0
    return p2

# Função para ez e e3
def p2b(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx)
        p2 = (p1 + 1)/2
        if (p2 > PEZ):
            p2 = 0
    return p2

# Função para h2 e e1
def p2c(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx)
        p2 = (p1)/2
        if (p2 > PH2):
            p2 = 0
    return p2

# Função para h1 e e2
def p2d(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        p2 = 0
    else:
        p1 = i + (j-1)*(Nx-1)
        p2 = (p1 + 1)/2
        if (p2 > PH1):
            p2 = 0
    return p2
#**************************************

# FUNÇÃO PARA FONTE DO TIPO PULSO RICKER OU SENOIDE
def fonte_1(tempo2,posicao,tipo_fonte,fator_at):
    if tipo_fonte == 0: # pulso ricker
        arg = np.pi*( (cdtds*tempo2 - posicao)/ppw - 1.0 )
        arg2 = arg*arg
        funcao =(1.0 - 2.0*arg2)*np.exp(-arg2)

    else:  # senóide com amortecimento inicial
        periodo = ppw/cdtds # periodo da senóide
                        # ou tempo para o pico do pulso ricker
        cte_tempo = fator_at*periodo
```

```
        funcao = ( 1.0 - np.exp(-1.0*tempo2/cte_tempo) )*\
                 np.sin(2*np.pi*tempo2*cdtds/ppw)
    return funcao
#**********************************

# função para obter i,j a partir de p para campo ez
def ez_ij(p2):
    p1 = 2*p2 - 1
    j = p1/Nx + 1
    if (p1%Nx==0 and p1>=Nx):
        j = j - 1
    i = p1 - (j-1)*Nx
    return i,j
#*********************************************************

# Função de interpolação para gráfico de ez no plano xy
def pteste(i,j):
    if ((i<=0) or (i>=Nx+1) or (j<=0) or (j>=Ny+1)):
        i = 0
        j = 0
    return i, j

# Função de interpolação para gráfico de ez no plano xz
def pteste2(i,k):
    if ((i<=0) or (i>=Nx+1) or (k<=0) or (k>=Nz+1)):
        i = 0
        k = 0
    return i, k
#*********************************************************
# nova função copiada em 23/02/2016
def medidav2(ez_t2, Nt, limiar, dist, fator_ppw, novo_alg):
    ez_t = np.zeros((Nt+1))
    ez_t[:] = ez_t2[:]
    n_limiar = Nt # modificado
    for i in range(Nt,0,-1):
        if (ez_t[i] > limiar):
            n_limiar = i
            break
    #n_limiar -= int(fator_ppw*ppw*ds/dist) + teste_disp
    n_limiar -= int(round(fator_ppw*ppw*ds/dist )) + teste_disp
    Nlimiar = Nt - n_limiar
    eztemp = np.zeros((Nlimiar))
    for i in range(0,Nlimiar):
        eztemp[i] = ez_t[n_limiar + i]
    n_max = eztemp.argmax()
    n_max += n_limiar
    ni = n_max # modificado
    for i in range(n_max,Nt):
        if ez_t[i] <= 0.0:
            ni = i
            break
    a = (ez_t[ni] - ez_t[ni-1])/dist
    b = ez_t[ni] - a*ni*dist
    x0 = -b/a
    # novo algoritmo em 18/06/2015
```

```
    if novo_alg == 1:
        d_ang = (2.0*np.pi/ppw)*dist/ds
        c_onda = ppw*ds
        f_alfa = ((ni-1.0)/ni)*(ez_t[ni-1]/ez_t[ni])
        B = f_alfa*np.cos(d_ang)
        C = f_alfa*np.sin(d_ang)
        teta_1 = np.arctan(C/(1.0 - B))
        x0 = (c_onda/2.0)*(-teta_1)/np.pi + (ni - 1.0)*dist
    return x0
#***********************************************
# NOVA FUNÇÂO
def medidav3(ez_t2, Nt, limiar, dist, fator_ppw, novo_alg):
    ez_t = np.zeros((Nt+1))
    ez_t[:] = ez_t2[:]
    n_limiar = Nt # modificado
    for i in range(Nt,0,-1):
        if (ez_t[i] > limiar):
            n_limiar = i
            break
    #n_limiar -= int(fator_ppw*ppw*ds/dist) + teste_disp
    n_limiar -= int(round(fator_ppw*ppw*ds/dist )) + teste_disp
    Nlimiar = Nt - n_limiar
    eztemp = np.zeros((Nlimiar))
    N_ppw = int(0.9*ppw*ds/dist )
    for i in range(0,N_ppw):
        eztemp[i] = ez_t[n_limiar + i]
    n_max = eztemp.argmax()
    n_max += n_limiar
    ni = n_max # modificado
    for i in range(n_max,Nt):
        if ez_t[i] <= 0.0:
            ni = i
            break
    a = (ez_t[ni] - ez_t[ni-1])/dist
    b = ez_t[ni] - a*ni*dist
    x0 = -b/a
    # novo algoritmo em 18/06/2015
    if novo_alg == 1:
        d_ang = (2.0*np.pi/ppw)*dist/ds
        c_onda = ppw*ds
        f_alfa = ((ni-1.0)/ni)*(ez_t[ni-1]/ez_t[ni])
        B = f_alfa*np.cos(d_ang)
        C = f_alfa*np.sin(d_ang)
        teta_1 = np.arctan(C/(1.0 - B))
        x0 = (c_onda/2.0)*(-teta_1)/np.pi + (ni - 1.0)*dist
    return x0

#***********************************************
def func_senoide(A2,r,fa,fb,sc,Nc,Nt,d,alfa,teta):
    c = fa*3.0e8
    dt = sc*d/c
    t = Nt*dt*fb
    fc = 1.0 # fc = -c para derivda no tempo
    co = Nc*d
    f = c/ co
```

```
    ct = alfa/f # com fator de atenuação: alfa
    k = 2.0*np.pi/co
    #fs2 = (A2*k/r )*np.cos(2.0*pi*f*t - r*k)
    fs2 = ( -A2/(c*ct*r) )*np.exp((-t + r/c)/ct)*\
       np.sin(2.0*np.pi*f*t - r*k + teta) + ( -A2*k/r )*( 1.0 -\
       np.exp((-t + r/c)/ct) )*np.cos(2.0*np.pi*f*t - r*k + teta)
    return fs2

def func_senoide2(A2,r,fa,fb,sc,Nc,Nt,d,alfa,teta):
    c = fa*3.0e8
    dt = sc*d/c
    t = Nt*dt*fb
    alfa2 = alfa
    fc = 1.0 # fc = -c para derivda no tempo
    co = Nc*d
    f = c/ co
    k = 2.0*np.pi/co
    fs2 = (-A2*k/r )*np.cos(2.0*np.pi*f*t - r*k + teta)
    return fs2

# FINAL DAS FUNÇÕES
#*********************************************************

# INICIALIZAÇÃO DE VARIÁVEIS E ENTRADA DE DADOS
Nx = 81 # numero de pontos na direção x
Ny = 141  # numero de pontos na direção y
Nz = 71 # numero de pontos na direção z
ppw = 15 # numero de pontos por comprimento de onda
tempomax = 40

tipo_fonte = 1  # 0=pulso ricker ou 1=senoide
HARD_SOFT  = 1   # fonte: 0=hard ou 1=soft
ATIVANDO_DIPOLO_1 = 0 # 1=ativa 0=desativa
ATIVANDO_DIPOLO_2 = 0 # 1=ativa 0=desativa

rdsdz = 1.0 #np.sqrt(1.0) # razão ds/dz
fator_comp = 1.4 # em comprimentos de onda para dipolo_1

arqsaida1 = "ez_pxy_1.txt"
arqsaida2 = "ez_pxz_2.txt"
fator_at = 0.5

# FINAL DAS PRINCIPAIS VARIÁVEIS

#************************************************

# VARIÁVEIS DE CONTROLE DOS GRÁFICOS

USAR_ESTES_ARQUIVOS = 0 # usar 1 ou 2, ou 0

MOSTRAR_SOMENTE_GRAFICOS_1D = 0 # 1= graficos 1D, 0 = mostra tudo

ATIVA_POLAR = 0 # 0 = desativa 1 = ativa diagrama polar
nivel = 1.e-4  # controla nível máximo graficos 2D
```

```
MAXPOLAR_AUTOMATICO = 0 # automatico=0 ou usar Max_polarx
Max_polarx = 20 # posição do máximo no eixo x (0 a Max_polarx)
porcentagemdepx = 0.5 # para obter o maxpolar automático
#*****************************************************

# COPIADO E APERFEIÇOADO EM 23/02/2016
# NESTA VERSÃO NOVA APROXIMAÇÃO SENOIDAL NA FUNÇÃO medidav2 EM 18/06/2015
novo_alg            =   1    # 1 = USA NOVO ALGORITMO
NP                  =  29
DELTA_COMP1         =  23
dxt = np.cos(np.pi/6.0)/20.0
teste_disp                = 0
xteste                    = -1.5 + 0.*dxt
NUMERO_COMPRIMENTOS1   = NP + xteste  # x1
NUMERO_COMPRIMENTOS1Y  = NP - 1.5 # y1
NUMERO_COMPRIMENTOS1B  = NP  + 3.   # d1
NUMERO_COMPRIMENTOS1C  = NP  + 2.5   #  a1 (senoide analitica)

LIMIAR1  = 5e-7   # x1
LIMIAR1Y = 5e-7   # y1
LIMIAR1D = 5e-7   # d1
LIMIAR1A = 5e-7   # a1
#################### 29/23 =  164.043453329 OU 30/24 = 163.892563187
angulo_inicial_senoide = 164.043453329
amplitude_inicial_senoide = 0.11
comprimento_seno_usado = 341.0

CURVA_DISPERSAO  = 1 # 0 = desativa; 1 = ativa
CURVA_DISPERSAO2 = 1 # 0 = desativa; 1 = ativa
ajuste_x = xteste # usei nova funçao medidav3
ajuste_y = -1.5
ajuste_d = 3.
ajuste_a = 2.5
er = 1.0e-25
AJUSTE_CURVA_ANALITICA = 1 # 0 = desativa; 1 = ativa
INICIO_DELTA_COMP1 = 7
REDUCAO_DISTANCIA_DIPERSAO = 1 # 0 = desativa; 1 = ativa
#*****************************************************

# LEITURA DE ARQUIVO DE ENTRADA
if USAR_ESTES_ARQUIVOS == 0:
    arquivo3 = open("entrada3.txt","r")

if USAR_ESTES_ARQUIVOS == 1:
    arquivo3 = open("entrada_ta1.txt","r")

if USAR_ESTES_ARQUIVOS == 2:
    arquivo3 = open("entrada_ta2.txt","r")

Nx = int(arquivo3.readline())#funcionou
Ny =  int(arquivo3.readline())
Nz =  int(arquivo3.readline())
ppw =  int(arquivo3.readline())
tempomax =  int(arquivo3.readline())
```

```
tipo_fonte =  int(arquivo3.readline())
HARD_SOFT =  int(arquivo3.readline())
ds =  float(arquivo3.readline())
rdsdz =  float(arquivo3.readline())
fator_cdtds = float(arquivo3.readline())
fator_at =  float(arquivo3.readline())
delta_passo = int(arquivo3.readline())
tempomax2 =  int(arquivo3.readline())
tempomax3 =  int(arquivo3.readline())
CPML =  int(arquivo3.readline())
nxx =  int(arquivo3.readline())
Reflexao = float(arquivo3.readline())
Reflexao2 = float(arquivo3.readline())
me = float(arquivo3.readline())
me2 = float(arquivo3.readline())
numero_materiais = int(arquivo3.readline())
numero_objetos = int(arquivo3.readline())
arquivo3.close()

fator_geometria = 2
# RETIRANDO "\n" DAS STRINGS
txx1 = list(arqsaida1)
txx1.pop()
arqsaida1 = "".join(txx1)

txx2 = list(arqsaida2)
txx2.pop()
arqsaida2 = "".join(txx2)

if USAR_ESTES_ARQUIVOS == 0:
    arqsaida1 = "ez_pxy_1_701.txt"
    arqsaida2 = "ez_pxz_2_701.txt"

if USAR_ESTES_ARQUIVOS == 1:
    arqsaida1 = "ez_pxy_1_ta1.txt"
    arqsaida2 = "ez_pxz_2_ta1.txt"

if USAR_ESTES_ARQUIVOS == 2:
    arqsaida1 = "ez_pxy_1_ta2.txt"
    arqsaida2 = "ez_pxz_2_ta2.txt"

cdtds = 1.0/np.sqrt(fator_cdtds) # número de Courant

#####################################################
eo = 1.0e-9/(36*np.pi)
uo = np.pi*4e-7
co = 1.0/np.sqrt(uo*eo)
#print 'co =',co

# IMPRIME VALORES OBTIDOS DO ARQUIVO DE ENTRADA
print 'Nx =',Nx
print 'Ny =',Ny
print 'Nz =',Nz
print 'ppw =',ppw
print 'tempomax =',tempomax
```

```
print 'tipo_fonte =',tipo_fonte
print 'HARD_SOFT =',HARD_SOFT
print 'rdsdz =', rdsdz
print 'fator_cdtds =',fator_cdtds
print 'cdtds =', cdtds
#print 'fator comprimento dipolo  =',fator_comp
print 'fator atenuação (senóide) =',fator_at
print

#************************************

razaodsdz = round(rdsdz,2) # para gráfico
CFL = int(cdtds*100)/100.0 # número de Courant para gráfico

ativo = 1  # tipo dipolo_1
passivo = 0
comprimento_dipolo = int(fator_comp*rdsdz*ppw)
compdipolo = int(fator_comp*100)/100.0 # para gráfico
imp0 = 120.0*np.pi

#************************************************

if (fator_geometria <= 2):
    ds = 1.0  # dimensão dos lados do hexagono maior 2D
    dx = ds*np.sqrt(3.0)/2.0 # tamanho da célula na direção x
    dy = ds/2.0   # tamanho da célula na direção y
    dz = ds/rdsdz
    Lx = (Nx)*dx  # comprimento na direção x
    Ly = (Ny)*dy  # comprimento na direção y
    Lz = (Nz)*dz  # comprimento na direção z
#*****************************************************

tipo_1d = 0 # grade 1D auxiliar
tipo_3d = 3 # grade 3D

# calculando dimensões das matrizes para os campos E e H
ix = Nx/2
rx = Nx%2
jy = Ny/2
ry = Ny%2
jy2 = (Ny-1)/2
ry2 = (Ny-1)%2

PEZ = (ix + rx)*(jy + ry) + (ix)*(jy)
PH2 = (ix)*(jy + ry) + (ix + rx)*(jy)
PH1 = (ix)*(jy2 + ry2) + (ix)*(jy2)
PH3 = PH1
#print 'PEZ= ',PEZ

# matrizes para os gráficos de Ez nos planos xy e xz
ez_pxy = np.zeros((Nx+1,Ny+1))
ez_pxz = np.zeros((Nx+1,Nz+1))

ez_pxy2 = np.zeros((Ny+1,Nx+1)) # transposta de ez_pxy
ez_pxz2 = np.zeros((Nz+1,Nx+1)) # transposta de ez_pxz
```

```
#************************************
px = Nx/2 + 1 # posição da fonte ricker
py = Ny/2 + 1
pz = Nz/2 + 1
pcentro = p2b(px,py)
#*****************************************

# textos para gráficos
texto2 =  'dim='+str(Nx)+'x'+str(Ny)+'x'+str(Nz)+\
          '  ppw='+str(ppw)+'  CFL='+str(CFL)+\
          '  ds/dz='+str(razaodsdz) + ' CPML='+ str(nxx)
if tipo_fonte == 0:
   texto1 = u'p_ricker '
else:
   texto1 = u'senóide '
if HARD_SOFT == 0:
   texto1 = texto1 + 'hard '
else:
   texto1 = texto1 + 'soft '
if ATIVANDO_DIPOLO_1 == 1 :
   texto3 = '  dipolo '+str(compdipolo)
else:
   texto3 = ''
textof = texto1 + texto2 + texto3
#************************************

# testando a fonte sozinha no tempo
ez_teste = np.zeros((tempomax))
tempo_teste = np.zeros((tempomax))

#******************************

# INICIO DOS CÁLCULOS PARA GRADE HEXAGONAL 3D

#print u'\nINICIO FDTD'
#print

# inicio loop do tempo

for tempo in range(0,tempomax):

   tempo_teste[tempo] = tempo
   ez_teste[tempo] = fonte_1(tempo,0.0,tipo_fonte,fator_at)

   #print '\n',tempo+1,'/',tempomax

# final loop do tempo
#*********************************************

# LEITURA DAS MATRIZES PARA PLANOS XY E XZ
arquivo4 = open(arqsaida1,"r")
#linha = None
for i in range(0,Nx+1):
   for j in range(0,Ny+1):
```

```
        linha = arquivo4.readline() #funcionou
        #print i
        #print j
        #print linha
        ez_pxy[i,j] = float(linha)

arquivo4.close()

arquivo5 = open(arqsaida2,"r")
#linha = None
for i in range(0,Nx+1):
     for j in range(0,Nz+1):
        linha = arquivo5.readline() #funcionou
        ez_pxz[i,j] = float(linha)
arquivo5.close()

#*********************************************

# eliminando valores zero das matrizes para os gráficos de ez
for i in range(1,Nx+1): # plano xy
    for j in range(1,Ny+1):
        if (j%2==0 and i%2==1 and i<=Nx) or\
           (j%2==1 and i%2==0 and i<=Nx-1):
              inx, jnx = pteste(i,j+1)
              isx, jsx = pteste(i,j-1)
              ie, je   = pteste(i+1,j)
              iw, jw   = pteste(i-1,j)
              ez_pxy[i,j] = 0.25*(ez_pxy[inx,jnx] + ez_pxy[isx,jsx] +\
                        ez_pxy[ie,je] + ez_pxy[iw,jw])
j = py
for i in range(1,Nx+1):  # plano xz
    for k in range(1,Nz+1):
        if (j%2==0 and i%2==1 and i<=Nx) or\
           (j%2==1 and i%2==0 and i<=Nx-1):
              ie, ke   = pteste2(i+1,k)
              iw, kw   = pteste2(i-1,k)
              ez_pxz[i,k] = 0.5*(ez_pxz[ie,ke] + ez_pxz[iw,kw])

#*********************************************

# gerando gráficos 1D a partir dos gráficos 2D
Nx2 = Nx - px
Ny2 = Ny - py
Nz2 = Nz - pz
ez_x1 = np.zeros((Nx2+1))
ez_y1 = np.zeros((Ny2+1))
ez_x2 = np.zeros((Nx2+1))
ez_z2 = np.zeros((Nz2+1))
x = np.zeros((Nx2+1))
y = np.zeros((Ny2+1))
z = np.zeros((Nz2+1))

for i in range(0,Nx2+1):
    x[i] = i*dx
    ez_x1[i] = ez_pxy[px+i,py] # plano xy
```

```
    ez_x2[i] = ez_pxz[px+i,pz] # plano xz

for j in range(0,Ny2+1):
    y[j] = j*dy
    ez_y1[j] = ez_pxy[px,py+j] # plano xy

for k in range(0,Nz2+1):
    z[k] = k*dz
    ez_z2[k] = ez_pxz[px,pz+k] # plano xz


if Nx < Ny: # diagonal do plano xy
    Nd = Nx2
else:
    Nd = Ny2
ez_d1 = np.zeros((Nd+1))
s = np.zeros((Nd+1))

for k in range(0,Nd+1):
    s[k] = k*ds
    ez_d1[k] = ez_pxy[px+k,py+k]

if Nx < Nz: # diagonal do plano xz
    Nd2 = Nx2
else:
    Nd2 = Nz2
ez_d2 = np.zeros((Nd2+1))
s2 = np.zeros((Nd2+1))

ds2 = np.sqrt(dx**2 + dz**2)
for k in range(0,Nd2+1):
    s2[k] = k*ds2

    ez_d2[k] = ez_pxz[px+k,pz+k]

#****************************************

#************************************************

ez_pxy2 = ez_pxy.T # transposta para corrigir imagem
ez_pxz2 = ez_pxz.T # transposta para corrigir imagem

#******************************************************

# ENTRADA DE VARIÁVEIS PARA FUNÇAO ANALITICA - SENOIDE
N = 900 # para definir resolução espacial

fr = np.zeros((N))
fsa = np.zeros((N))
fsa2 = np.zeros((N))
r = np.zeros((N))
pi = np.pi

Ar = 0.25
Nt = tempomax    # passos no tempo
```

```
fa = 1.0    # não altera forma de onda
fb = 1.0     # fb = 0.8 para pulso
          # fb = 1.0 para senoide
A = amplitude_inicial_senoide   #0.11 # amplitude da senoide na direção x
tetas = angulo_inicial_senoide
teta1 = tetas*np.pi/180.0 # fase inicial

A2 = -0.0196    # amplitude da senoide na direção z
fb2 = 1.0
teta2 = 160*np.pi/180 # fase inicial

alfa = fator_at   # fator de atenuação da cossenoide
L = 350.0
c = fa*3.0e8
sc = 1.0/np.sqrt(fator_cdtds) # numero de Courant
Nc = ppw   # numero de pontos por comprimento de onda
d = 1.0     # tamanho do lado do hexagono grande
dt = sc*d/c
t = Nt*dt*fb
fc = 1.0        # fc = -c para derivda no tempo
comp = Nc*d
f = c/ comp
ro = t*c
cdr = 1.1 #/np.sqrt(3.0)
dr = ro/N #  1.0 para senoide ou >1.0 para pulso
# 118.0 é um valor empirico; maiores valores a senoide analitica
# aumenta a sua amplitude e é um pouco maior que ro=115.47
dr = comprimento_seno_usado/N
ro2 = N*dr

for i in range(1,N):
    r[i] = i*dr
    r2 = i*dr
    fsa[i] = func_senoide(A,r2,fa,fb,sc,Nc,Nt,d,alfa,teta1)
    fsa2[i] = func_senoide(A2,r2,fa,fb2,sc,Nc,Nt,d,alfa,teta2)
    fr[i] = Ar/(r2*r2)
# FINAL SENOIDE ANALITICA
#****************************************
#*********************************************************
# COPIADO EM 23/02/2016
# NOVA FORMA DE MEDIR ANISOTROPIA
# NOVA FORMA DE MEDIDA DE ANISOTROPIA NO PLANO XY
# achando pontos de nulos para medir anisotropia (plano xy)
if tipo_fonte == 1:
    limiar = LIMIAR1
    limiary = LIMIAR1Y
    limiard = LIMIAR1D
    limiara = LIMIAR1A
    ncomp = 1.0*NUMERO_COMPRIMENTOS1 # numero de comprimentos de onda
    ncompy = 1.0*NUMERO_COMPRIMENTOS1Y
    ncompB = 1.0* NUMERO_COMPRIMENTOS1B
    ncompC = 1.0* NUMERO_COMPRIMENTOS1C

    x1_limiar = medidav2(ez_x1,Nx2,limiar,dx,ncomp,novo_alg)
    y1_limiar = medidav2(ez_y1,Ny2,limiary,dy,ncompy,novo_alg)
```

```
d1_limiar = medidav2(ez_d1,Nd,limiard,ds,ncompB,novo_alg)
a1_limiar = medidav2(fsa,N-1,limiara,dr,ncompC,novo_alg) #senoide analitica

dt_limiar = tempomax
print u'# 1° PONTOS DE NULOS - PLANO XY'
print u'# [x1_limiar, y1_limiar, d1_limiar, a1_limiar,tempomax, ppw, HARD_SOFT]'
print "{0:2}{1:>24.20f}{2:2}".format("[",x1_limiar,",")
print "{0:>24.20f}{1:2}".format(y1_limiar,",")
print "{0:>24.20f}{1:2}".format(d1_limiar,",")
print "{0:>24.20f}{1:2}".format(a1_limiar,",")
print "{0:>24.20f}{1:2}".format(dt_limiar,",")
print "{0:>24.20f}{1:2}{2:2d}{3:2}".format(ppw,",",HARD_SOFT,"]")
print
limiar = LIMIAR1
ncomp = 1.0*NUMERO_COMPRIMENTOS1 - DELTA_COMP1
ncompy = 1.0*NUMERO_COMPRIMENTOS1Y - DELTA_COMP1
ncompB = 1.0*NUMERO_COMPRIMENTOS1B - DELTA_COMP1
ncompC = 1.0*NUMERO_COMPRIMENTOS1C - DELTA_COMP1

x1_limiar2 = medidav2(ez_x1,Nx2,limiar,dx,ncomp,novo_alg)
y1_limiar2 = medidav2(ez_y1,Ny2,limiary,dy,ncompy,novo_alg)
d1_limiar2 = medidav2(ez_d1,Nd,limiard,ds,ncompB,novo_alg)
a1_limiar2 = medidav2(fsa,N-1,limiara,dr,ncompC,novo_alg) #senoide analitica

dt_limiar = tempomax
print u'# 2° PONTOS DE NULOS - PLANO XY'
print u'# [x1_limiar, y1_limiar, d1_limiar, a1_limiar,tempomax, ppw, HARD_SOFT]'
print "{0:2}{1:>24.20f}{2:2}".format("[",x1_limiar2,",")
print "{0:>24.20f}{1:2}".format(y1_limiar2,",")
print "{0:>24.20f}{1:2}".format(d1_limiar2,",")
print "{0:>24.20f}{1:2}".format(a1_limiar2,",")
print "{0:>24.20f}{1:2}".format(dt_limiar,",")
print "{0:>24.20f}{1:2}{2:2d}{3:2}".format(ppw,",",HARD_SOFT,"]")
print
avy1 = 100.0*( y1_limiar2 - y1_limiar - (x1_limiar2 -\
        x1_limiar))/(x1_limiar2 - x1_limiar)
avd1 = 100.0*( d1_limiar2 - d1_limiar - (x1_limiar2 -\
        x1_limiar))/(x1_limiar2 - x1_limiar)
ahex = ((np.pi/ppw)**4)*(100.0/360)
ayee = ((np.pi/ppw)**2)*(100.0/12)
dispx = (x1_limiar2/a1_limiar2 - 1.0)*100 # calcula dispersão
dispy = (y1_limiar2/a1_limiar2 - 1.0)*100
dispd = (d1_limiar2/a1_limiar2 - 1.0)*100
print u'RESULTADOS DE ANISOTROPIA E DISPERSÂO (HEX 3D)'
print u'TIPO_INTERPOLAÇÃO (LINEAR=0; SENO=1) =',novo_alg
print 'NUMERO COMPRIMENTOS (NP) =',NP
print 'NUMERO_COMPRIMENTOS1    =',NUMERO_COMPRIMENTOS1
print 'DELTA_COMPRIMENTOS1     =',DELTA_COMP1
print 'PLANO XY'
print 'avy1(%)  =',avy1
print 'avd1(%)  =',avd1
print 'dispx(%) =',dispx
print 'dispy(%) =',dispy
print 'dispd(%) =',dispd
print 'ahex(%)  =',ahex
```

```
    print 'ayee(%)  =',ayee
    print u'Nível senóide analítica  =',A
    print u'Ângulo senóide (graus)   =',tetas
    print u'Comprimento seno (usado) =',ro2
    print u'Comprimento seno (max)   =',ro
    print
# FINAL MEDIDAS ANISOTROPIA E DISPERSÃO
#*******************************************************

# CALCULANDO CURVAS DE DISPERSÃO EM FUNÇÃO DA DISTÃNCIA
# (PONTOS DE NULOS)
if (CURVA_DISPERSAO == 1):
    x1_lim = np.zeros((DELTA_COMP1 + 1))
    y1_lim = np.zeros((DELTA_COMP1 + 1))
    d1_lim = np.zeros((DELTA_COMP1 + 1))
    a1_lim = np.zeros((DELTA_COMP1 + 1))
    cdispx = np.zeros((DELTA_COMP1 + 1))
    cdispy = np.zeros((DELTA_COMP1 + 1))
    cdispd = np.zeros((DELTA_COMP1 + 1))
    for i in range(0, DELTA_COMP1 + 1):
        NP2 = 1.0*NP - 1.0*i
        limiar = LIMIAR1
        limiary = LIMIAR1Y
        limiard = LIMIAR1D
        limiara = LIMIAR1A
        ncomp =  1.0*NP2 + ajuste_x
        ncompy = 1.0*NP2 + ajuste_y
        ncompB = 1.0*NP2 + ajuste_d
        ncompC = 1.0*NP2 + ajuste_a # senoide analitica

        x1_lim[i] = medidav3(ez_x1,Nx2,limiar,dx,ncomp,novo_alg)
        y1_lim[i] = medidav2(ez_y1,Ny2,limiary,dy,ncompy,novo_alg)
        d1_lim[i] = medidav2(ez_d1,Nd,limiard,ds,ncompB,novo_alg)
        a1_lim[i] = medidav2(fsa,N-1,limiara,dr,ncompC,novo_alg)
        cdispx[i] = (x1_lim[i]/a1_lim[i] - 1.0)*100 # calcula dispersão
        cdispy[i] = (y1_lim[i]/a1_lim[i] - 1.0)*100
        cdispd[i] = (d1_lim[i]/a1_lim[i] - 1.0)*100

# CURVAS DE DISPERSÃO COM AJUSTE ZERO PARA O INICIO DAS CURVAS
delta_dispx = a1_lim[0] - x1_lim[0] # ajusta para dispersão zero
# NOVA CURVA ANALITICA COM OS AJUSTES
teta1b = teta1 - delta_dispx*2.0*np.pi/(1.0*ppw*ds)
for i in range(1,N):
    r2 = i*dr
    #r[i] = r[i] - 0.0*delta_dispx
    fsa2[i] = func_senoide(A,r2,fa,fb,sc,Nc,Nt,d,alfa,teta1b)

if (CURVA_DISPERSAO2 == 1):
    x1_lim2 = np.zeros((DELTA_COMP1 + 1))
    y1_lim2 = np.zeros((DELTA_COMP1 + 1))
    d1_lim2 = np.zeros((DELTA_COMP1 + 1))
    a1_lim2 = np.zeros((DELTA_COMP1 + 1))
    cdispx2 = np.zeros((DELTA_COMP1 + 1))
    cdispy2 = np.zeros((DELTA_COMP1 + 1))
    cdispd2 = np.zeros((DELTA_COMP1 + 1))
```

```
    for i in range(0, DELTA_COMP1 + 1):
        NP2 = 1.0*NP - 1.0*i
        limiar = LIMIAR1
        limiary = LIMIAR1Y
        limiard = LIMIAR1D
        limiara = LIMIAR1A
        ncomp =  1.0*NP2 + ajuste_x
        ncompy = 1.0*NP2 + ajuste_y
        ncompB = 1.0*NP2 + ajuste_d
        ncompC = 1.0*NP2 + ajuste_a # senoide analitica

        x1_lim2[i] = medidav3(ez_x1,Nx2,limiar,dx,ncomp,novo_alg)
        y1_lim2[i] = medidav2(ez_y1,Ny2,limiary,dy,ncompy,novo_alg)
        d1_lim2[i] = medidav2(ez_d1,Nd,limiard,ds,ncompB,novo_alg)
        if AJUSTE_CURVA_ANALITICA == 1:
            a1_lim2[i] = medidav2(fsa2,N-1,limiara,dr,ncompC,novo_alg)
        else:
            a1_lim2[i] = medidav2(fsa,N-1,limiara,dr,ncompC,novo_alg) -\
                  delta_dispx

        cdispx2[i] = (x1_lim2[i]/a1_lim2[i] - 1.0)*100 # calcula dispersão
        cdispy2[i] = (y1_lim2[i]/a1_lim2[i] - 1.0)*100
        cdispd2[i] = (d1_lim2[i]/a1_lim2[i] - 1.0)*100
    if REDUCAO_DISTANCIA_DIPERSAO == 1:
        for i in range(0, DELTA_COMP1 + 1):
            cdispx2[i] = ((x1_lim2[i]-x1_lim2[0]+er)/(a1_lim2[i]-a1_lim2[0]+er) - 1.0)*100
            cdispy2[i] = ((y1_lim2[i]-y1_lim2[0]+er)/(a1_lim2[i]-a1_lim2[0]+er) - 1.0)*100
            cdispd2[i] = ((d1_lim2[i]-d1_lim2[0]+er)/(a1_lim2[i]-a1_lim2[0]+er) - 1.0)*100

# FINAL DAS 3 CURVAS DE DISPERSÃO

# CALCULANDO AS DISPERSÕES MÉDIAS
cdispx2_med = 0.0
cdispy2_med = 0.0
cdispd2_med = 0.0

for i in range(INICIO_DELTA_COMP1, DELTA_COMP1 + 1):
    cdispx2_med += cdispx2[i]
    cdispy2_med += cdispy2[i]
    cdispd2_med += cdispd2[i]
cdispx2_med /= (DELTA_COMP1 + 1 - INICIO_DELTA_COMP1)
cdispy2_med /= (DELTA_COMP1 + 1 - INICIO_DELTA_COMP1)
cdispd2_med /= (DELTA_COMP1 + 1 - INICIO_DELTA_COMP1)
NUMERO_PONTOS = DELTA_COMP1 + 1 - INICIO_DELTA_COMP1
teta_final = teta1b*180.0/np.pi

print u'AJUSTE_CURVA_ANALITICA     =',AJUSTE_CURVA_ANALITICA
print u'Ângulo final (graus)       =',teta_final
print u'REDUCAO_DISTANCIA_DIPERSAO =',REDUCAO_DISTANCIA_DIPERSAO
print u'NUMERO_PONTOS     =',NUMERO_PONTOS
print u'dispersão média x =',cdispx2_med
print u'dispersão média y =',cdispy2_med
print u'dispersão média d =',cdispd2_med
print u'inicio =',INICIO_DELTA_COMP1
print u'cdispx2[inicio] = ',cdispx2[INICIO_DELTA_COMP1]
```

```
print u'cdispx2[final] = ',cdispx2[DELTA_COMP1]
print u'cdispy2[final] = ',cdispy2[DELTA_COMP1]
print u'cdispd2[final] = ',cdispd2[DELTA_COMP1]

# FINAL CALCULO DAS DISPERSÕES MÉDIAS

#*********************************************************

# PLOTANDO CAMPOS Ez PARA O MODO 3D
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt # HEX 3D
fs1 = 16
fs2 = 17
fs3 = 17
fse = 10
fse2 = 14
fse3 = 14

# nearest ou bilinear - funcionou
# cm.jet ou cm.RdYlGn ou cm.winter ou cm.hot
# ficou imagem correta com shape=None e com extent=[0,Lx,0,Ly]
# mas shape inverteu eixos x e y
# no entanto usando a transposta de ez a imagem ficou com eixos corretos

# GRÁFICOS 2D
if MOSTRAR_SOMENTE_GRAFICOS_1D == 0:
   # plano xy
   plt.figure()
   im = plt.imshow(ez_pxy2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Ly], shape=None,
                vmax= nivel*abs(ez_pxy2).max(), vmin= -nivel*abs(ez_pxy2).max()  )

   plt.colorbar(im) # funcionou
   plt.title(u'HEX 3D: Ez (V/m) NO PLANO XY (T = '+\
               str(tempomax)+' passos)',fontsize = fs1)
   plt.xlabel(u'Lx (m)',fontsize = fs1)
   plt.ylabel(u'Ly (m)',fontsize = fs1)
   #plt.show()
   if TESTE_REFLEXAO == 1:
      # posições dos pontos de medida de reflexão
      label((px+71)*dx,(py-0)*dy,u'*',20)
      label((px+71+8)*dx,(py-0)*dy,u'A',18)
      label((px+0)*dx,(py + 121 -0)*dy,u'*',20)
      label((px+8)*dx,(py + 121 -0)*dy,u'B',18)
      label((px+71)*dx,(py + 121 -0)*dy,u'*',20)
      label((px+71+8)*dx,(py + 121 -0)*dy,u'C',18)

   # plano xz
   plt.figure()
   plt.tick_params(labelsize = fse)
   im2 = plt.imshow(ez_pxz2, interpolation='nearest', cmap=cm.jet,
                origin='lower', extent=[0,Lx,0,Lz], shape=None,
                vmax= nivel*abs(ez_pxz2).max(), vmin= -nivel*abs(ez_pxz2).max()  )
```

```
    plt.colorbar(im2) # funcionou
    plt.title(u'HEX 3D: Ez (V/m) NO PLANO XZ (T = '+\
                str(tempomax)+' passos)',fontsize = fs1)
    plt.xlabel(u'Lx (m)',fontsize = fs1)
    plt.ylabel(u'Lz (m)',fontsize = fs1)
    #plt.show()
    if TESTE_REFLEXAO == 1:
        # posições dos pontos de medida de reflexão
        label(px*dx,(pz - 1 + 71)*dz,u'*',20)
        label((px+ 8)*dx,(pz - 1 + 71)*dz,u'D',18)

#*********************************************
# GRÁFICOS 1D
# no plano xy
plt.figure()
plt.tick_params(labelsize = fse2)
plt.plot(x,ez_x1,'r-')
plt.axis([0.,360.,-.008,.008])
plt.grid()
plt.xlabel(u'Eixos x/y/d (metros)',fontsize = fs2)
plt.ylabel(u'Amplitude (V/m)',fontsize = fs2)
plt.title(u'HEX 3D: CAMPOS Ez NO PLANO XY (TEMPO = '+\
            str(tempomax)+' passos)',fontsize = fs2)
plt.hold(True) # permite sobrepor vários
               # gráficos com dimensões diferentes
plt.plot(y,ez_y1,'b-')
plt.plot(s,ez_d1,'k-')
if AJUSTE_CURVA_ANALITICA == 1:
    plt.plot(r,fsa2,'g--')
else:
    plt.plot(r,fsa,'y')
texto1 = u'ez_x1 ( 0°)'
texto2 = u'ez_y1 (90°)'
texto3 = u'ez_d1 (30°)'
texto4 = u'Teórico'
plt.legend((texto1,texto2,texto3,texto4),loc = 'upper right')
plt.hold(False)

#plt.show()

# no plano xz
plt.figure()
plt.plot(x,ez_x2,'r')
plt.axis([0.,360.,-.008,.008])
plt.grid()
plt.xlabel(u'Eixos x/z/d, '+ textof)
plt.ylabel(u'Nível')
plt.title('Campos 1D: Ez plano xz (TEMPO = '+str(tempomax)+' passos)')
plt.hold(True) # permite sobrepor vários
               # gráficos com dimensões diferentes

plt.plot(z,ez_z2,'b-')
plt.plot(s2,ez_d2,'k')
texto1 = u'ez_x2 ( 0°)'
texto2 = u'ez_z2 (90°)'
```

```
texto3 = u'ez_d2 (45°)'
plt.legend((texto1,texto2,texto3),loc = 'upper right')
plt.hold(False)
#plt.show()
#*************************************************

# PLOTANDO CURVAS DE DISPERSÃO NO PLANO XY
if (CURVA_DISPERSAO == 1):
    plt.figure()
    plt.tick_params(labelsize = fse3)
    plt.plot(x1_lim,cdispx,'r.-')
    #plt.plot(y1_lim,cdispy,'r')
    plt.grid()
    plt.xlabel(u'Eixos x/y/d (metros)',fontsize = fs3)
    plt.ylabel(u'Dispersão (%)',fontsize = fs3)
    plt.title(u'HEX 3D: Curvas de dispersão no plano xy (MODO 1)',
              fontsize = fs3)
    plt.hold(True) # permite sobrepor vários
                   # gráficos com dimensões diferentes
    plt.plot(y1_lim,cdispy,'b-')
    plt.plot(d1_lim,cdispd,'k-')
    texto1 = u'x_( 0°)'
    texto2 = u'y_(90°)'
    texto3 = u'd_(30°)'
    plt.legend((texto1,texto2,texto3),loc = 'lower right')
    plt.hold(False)
    #plt.show()
if (CURVA_DISPERSAO2 == 1):
    plt.figure()
    plt.tick_params(labelsize = fse3)
    plt.plot(x1_lim2,cdispx2,'r.-')
    #plt.plot(y1_lim2,cdispy2,'r')
    plt.grid()
    plt.xlabel(u'Eixos x/y/d (metros)',fontsize = fs3)
    plt.ylabel(u'Dispersão (%)',fontsize = fs3)
    plt.title(u'HEX 3D: Curvas de dispersão no plano xy (MODO 2)',
              fontsize = fs3)
    plt.hold(True) # permite sobrepor vários
                   # gráficos com dimensões diferentes
    plt.plot(y1_lim2,cdispy2,'b-')
    plt.plot(d1_lim2,cdispd2,'k-')
    texto1 = u'x_( 0°)'
    texto2 = u'y_(90°)'
    texto3 = u'd_(30°)'
    plt.legend((texto1,texto2,texto3),loc = 'lower right')
    plt.hold(False)
    #plt.show()

#*********************************************************

if MOSTRAR_SOMENTE_GRAFICOS_1D == 1:
    # fonte em função do tempo
    plt.figure()
    plt.plot(tempo_teste,ez_teste,'k')
    plt.grid()
```

```
    plt.xlabel(u'Eixo Tempo, '+ textof)
    plt.ylabel(u'Nível')
    plt.title('Fonte para campo Ez ( TEMPO = '+str(tempomax)+\
            ' passos, FATOR_AT = '+str(fator_at)+' )' )
    #plt.show()

plt.show()
# FINAL DO PROGRAMA
#########################################################################
```

ARQUIVO 3: PROGRAMA QUE GERA ARQUIVO velocidade_hex3d.txt

```
# -*- coding: cp1252 -*-

# programa: programa_fdtd_determinante_2a_com_grafico_1a.py
# AUTOR: MARINOEL JOAQUIM
# VERSÃO USADA PARA PLOTAR CURVAS DE ANISOTROPIA: HEX3D, YEE3D
# COM QUALQUER NUMERO DE ANGULOS
# 08/09/2014
# TROCADO SINAIS ERRADOS, MAS SEM ALTERAR RESULTADOS EM 16/09/2014
#*****************************************************************
# BILIOTECAS NECESSÁRIAS AO PROGRAMA
import numpy as np
#*****************************************************************

FORMA_EXPOENTE = 0 # 1 = EXP OU 0 = COS + J*SIN
TIPO_COEF = 3 # 1 ou 2 ou 3 (correto)

TIPO = 0 # 0 = 3D HEX OU  1 = 2D HEX OU 2 = 3D YEE

# PARA TIPO = 0 (3D)
TIPO_K = 3 # 1 (qqr plano), 2 (plano xy), 3 (plano xz), 4 (plano yz)
        # 5 (plano perpendicular a alfa no plano xy)
MOSTRA_ANISOTROPIA = 0 # 0= não mostra ou 1= mostra
#*****************************************************************

# INÍCIO DOS CÁLCULOS
Nf = 451
alfag = np.zeros((Nf))
nivel = np.zeros((Nf))
nivel2 = np.zeros((Nf))
nively = np.zeros((Nf)) # para o método YEE 3D
cfly = 1.0/np.sqrt(3.0)

fbx = 0.8781 # Para tipo _coef = 1
fbx2 = 0.935 # Para tipo _coef = 2

# Para tipo _coef = 3 (correto)
fax3 = 1.0  # funcionou com 0.999 para N=15
fbx3 = 1.0  # funcionou com 1.001 para N=15
FATOR_AUMENTO = 0.9998 # ajusta a velocidade: para velocidade_hex.txt
N = 10
rdz  = 1.1547 # 1.06 anisotropia próxima ao yee
fator_cfl_2d = 2.0
#fator_cfl = fator_cfl_2d + rdz**2  #aproximado
```

```
fator_cfl = (np.sqrt(9.*rdz**4 + 28.*rdz**2 + 36.)+ 3.*rdz**2 + 6.)/6 #correto

cfl = 1.0/np.sqrt(fator_cfl)

ds = 1.0

zo = 120*np.pi
k = 2*np.pi/(N*ds)
dalfa = 2*np.pi/((Nf-1)*8)

if TIPO == 2:
    fator_cfl = 3.0
    cfl = 1.0/np.sqrt(fator_cfl)
    rdz  = 1.0 #


for i in range(0,Nf):
    alfag[i] = i*dalfa*180/np.pi
    alfa = i*dalfa
    if (TIPO == 0 or TIPO == 2):  # 3D: HEX (0) OU YEE (2)
        if TIPO_K == 1: # k no espaço 3D
            teta = (np.pi/180)*45
            #teta = np.arctan(1.0/np.cos(np.pi/6))
            kx = k*np.cos(alfa)*np.sin(teta)
            ky = k*np.sin(alfa)*np.sin(teta)
            kz = k*np.cos(teta)
            texto1 = u'PLANO COM ÂNGULO TETA = '+str(round((teta*180/np.pi),2))
        if TIPO_K == 2: # k no plano xy
            teta = np.pi/2 # não usado
            #alfa = 1*np.pi/3
            kx = k*np.cos(alfa)
            ky = k*np.sin(alfa)
            kz = 0.0
            texto1 = 'PLANO XY'
        if TIPO_K == 3: # k no plano xz
            teta = np.pi/2 # não usado
            #alfa = 1*np.pi/2
            #alfa = np.arctan(1.0/np.cos(np.pi/6))
            kx = k*np.cos(alfa)
            ky = 0.0
            kz = k*np.sin(alfa)
            texto1 = 'PLANO XZ'
        if TIPO_K == 4: # k no plano yz
            teta = np.pi/2 # não usado
            #alfa = 1*np.pi/3
            ky = k*np.cos(alfa)
            kz = k*np.sin(alfa)
            kx = 0.0
            texto1 = 'PLANO YZ'
        if TIPO_K == 5: # k no espaço 3D, para um certo alfa
            teta = (np.pi/180)*45 # trocando teta com alfa por simplicidade
            kx = k*np.cos(teta)*np.sin(alfa)
            ky = k*np.sin(teta)*np.sin(alfa)
            kz = k*np.cos(alfa)
            texto1 = u'PLANO COM ÂNGULO ALFA = '+\
```

```
        str(round((teta*180/np.pi),2))+ ' (PLANO XY)'


if TIPO == 1: # 2D HEX
    teta = np.pi/2 # não usado
    #alfa = 1*np.pi/3
    kx = k*np.cos(alfa)
    ky = k*np.sin(alfa)
    kz = 0.0
    texto1 = 'PLANO XY (2D)'
#*********************************************


if TIPO == 0: # 3D HEX
    if TIPO_COEF == 1:
        fa = 3.0*np.sqrt(3.0)/4
        fb = fbx*3.0/4
        fc = 2.0/3
    if TIPO_COEF == 2:
        fa = 2.0*np.sqrt(3.0)/3
        fb = fbx2*2.0/3
        fc = 2.0/3
    if TIPO_COEF == 3:
        fa = fax3*1.0
        fb = fbx3*1.0/np.sqrt(3.0)
        fc = 2.0/3
if TIPO == 1:
    fa = 1.0
    fb = 1.0 #não usado
    fc = 2.0/3
if TIPO == 2:
    fa = 1.0
    fb = 1.0 #não usado
    fc = 1.0



ha = fa*cfl/zo
ea = fa*cfl*zo
hb = fb*rdz*cfl/zo
eb = fb*rdz*cfl*zo
hz = fc*cfl/zo
ez = fc*cfl*zo


asc1 = (3.0**(1.0/2))*kx*ds/12.0 - ky*ds/4.0
asc2 = (3.0**(1.0/2))*kx*ds/12.0 + ky*ds/4.0
asc3 = (3.0**(1.0/2))*kx*ds/6.0
s1 = np.sin(ky*ds/2.0)
s2 = np.sin((3.0**(1.0/2))*kx*ds/4.0 - ky*ds/4.0)
s3 = np.sin((3.0**(1.0/2))*kx*ds/4.0 + ky*ds/4.0)
sz = np.sin(kz*ds/(2.0*rdz))

j2 = 1.0j
sc1 = (np.cos(asc1) - np.sin(asc1)*j2)
```

```
sd1 = (np.cos(asc1) + np.sin(asc1)*j2)
sc2 = (np.cos(asc2) - np.sin(asc2)*j2)
sd2 = (np.cos(asc2) + np.sin(asc2)*j2)
sc3 = (np.cos(asc3) + np.sin(asc3)*j2)
sd3 = (np.cos(asc3) - np.sin(asc3)*j2)

#sx2 = np.sin(kx*ds/2.0) # método Yee
#sy2 = np.sin(ky*ds/2.0)
#sz2 = np.sin(kz*ds/2.0)


if  FORMA_EXPOENTE == 1:
    sc1 = np.exp(-j2*asc1)
    sd1 = np.exp(j2*asc1)
    sc2 = np.exp(-j2*asc2)
    sd2 = np.exp(j2*asc2)
    sc3 = np.exp(j2*asc3)
    sd3 = np.exp(-j2*asc3)


if TIPO == 0:
    a6 = -hb*sz*sc1
    a7 = -hb*sz*sc2
    a8 = ha*s1
    b5 = hb*sz*sc1
    b7 = -hb*sz*sc3
    b8 = -ha*s2
    c5 = hb*sz*sc2
    c6 = hb*sz*sc3
    c8 = -ha*s3
    d5 = -hz*s1
    d6 = hz*s2
    d7 = hz*s3
    e2 = eb*sz*sd1
    e3 = eb*sz*sd2
    e4 = -ea*s1
    f1 = -eb*sz*sd1
    f3 = eb*sz*sd3
    f4 = ea*s2
    g1 = -eb*sz*sd2
    g2 = -eb*sz*sd3
    g4 = ea*s3
    h1 = ez*s1
    h2 = -ez*s2
    h3 = -ez*s3
    a = 0.0
    x2 = np.matrix([[a,0,0,0,0,a6,a7,a8],
                    [0,a,0,0,b5,0,b7,b8],
                    [0,0,a,0,c5,c6,0,c8],
                    [0,0,0,a,d5,d6,d7,0],
                    [0,e2,e3,e4,a,0,0,0],
                    [f1,0,f3,f4,0,a,0,0],
                    [g1,g2,0,g4,0,0,a,0],
                    [h1,h2,h3,0,0,0,0,a]])
    x3 = np.linalg.eigvals(x2)
```

```
    x4 = (N/(np.pi*cfl))*np.arcsin(x3)
    x5 = x4.max()
    nivel[i] = np.real(x5)
    nivel2[i] = np.imag(x5)

if TIPO == 2: # método Yee
    dx = ds
    dy = ds
    dz = ds
    sx2 = np.sin(kx*dx/2)
    sy2 = np.sin(ky*dy/2)
    sz2 = np.sin(kz*dz/2)
    ha = 1.0
    hb = cfl/zo
    ea = 1.0
    eb = cfl*zo
    a5 = -hb*sz2
    a6 = hb*sy2
    b4 = hb*sz2
    b6 = -hb*sx2
    c4 = -hb*sy2
    c5 = hb*sx2
    d2 = eb*sz2
    d3 = -eb*sy2
    e1 = -eb*sz2
    e3 = eb*sx2
    f1 = eb*sy2
    f2 = -eb*sx2
    a = 0.0
    x2 = np.matrix([[a,0,0,0,a5,a6],
                    [0,a,0,b4,0,b6],
                    [0,0,a,c4,c5,0],
                    [0,d2,d3,a,0,0],
                    [e1,0,e3,0,a,0],
                    [f1,f2,0,0,0,a]])
    #x3 = np.linalg.eigvals(x2)
    x3 = cfl*np.sqrt(sx2**2 + sy2**2 + sz2**2) #equação analitica yee

    x4 = (N/(np.pi*cfl))*np.arcsin(x3)
    x5 = x4.max()
    nivel[i] = np.real(x5)
    nivel2[i] = np.imag(x5)

if TIPO == 1:
    a8 = ha*s1
    b8 = -ha*s2
    c8 = -ha*s3
    h1 = ez*s1
    h2 = -ez*s2
    h3 = -ez*s3
    a = 0.0
    x2 = np.matrix([[a,0,0,a8],
                    [0,a,0,b8],
                    [0,0,a,c8],
                    [h1,h2,h3,a]])
```

```
        x3 = np.linalg.eigvals(x2)
        x4 = (N/(np.pi*cfl))*np.arcsin(x3)
        x5 = x4.max()
        nivel[i] = np.real(x5)
        nivel2[i] = np.imag(x5)

anisotropia_max = (nivel.max() - nivel.min())*100/nivel.min()
anivelmax = nivel.max()
anivelmin = nivel.min()
if TIPO == 0:
    anivelmax = FATOR_AUMENTO*nivel.max()
    anivelmin = FATOR_AUMENTO*nivel.min()

anivelmed = (anivelmax + anivelmin)/2.0
fatormed = 1.0/anivelmed
fatormin = 1.0/anivelmin
anteorica_yee = ( (np.pi/N)**2 )*100/12
anteorica = ( (np.pi/N)**4 )*100/360
# FINAL CALCULOS DOS AUTOVALORES

# ARQUIVO DE SAÍDA PARA VELOCIDADE NORMALIZADA
if TIPO == 0:
    arquivox = open("velocidade_hex3d.txt","w")
    for i in range(0,Nf):
        nivel[i] = FATOR_AUMENTO*nivel[i]
        arquivox.write("%18.17f\n" % nivel[i])  # plano xz
    arquivox.close()

#*********************************************

# PLOTANDO CAMPOS Ez PARA O MODO 3D
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

# GRÁFICOS 1D
# no plano desejado
lw = 1.3
fs1 = 12
if TIPO == 0: # HEX 3D
    plt.figure()
    #plt.savefig('alldone',dpi=1000)
    plt.tick_params(labelsize= 12)
    plt.ticklabel_format(style='sci',useOffset=False)
    plt.plot(alfag,nivel,'k-',linewidth=lw)
    plt.grid()
    plt.xlabel(u'Angle (degrees)',fontsize=fs1)
    plt.ylabel(u'Normalized Phase Velocity',fontsize=fs1)
    #plt.title(u'ANISOTROPIA (PRISMAS HEXAGONAIS) NO '+ texto1)
if TIPO == 1: # HEX 2D
    plt.figure()
    plt.plot(alfag,nivel,'k-',linewidth=lw)
    plt.grid()
    plt.xlabel(u'Angle (degrees)',fontsize=fs1)
    plt.ylabel(u'Normalized Phase Velocity',fontsize=fs1)
```

```
    plt.title(u'ANISOTROPIA NO '+ texto1)
if TIPO == 2: # YEE 3D
    plt.figure()
    plt.savefig('alldone',dpi=1000)
    plt.tick_params(labelsize= 19)
    plt.plot(alfag,nivel,'k-',linewidth=lw)
    plt.grid()
    plt.xlabel(u'Angle (degrees)',fontsize=fs1)
    plt.ylabel(u'Normalized Phase Velocity',fontsize=fs1)
    #plt.title(u'ANISOTROPIA (MÉTODO YEE) NO '+ texto1)


plt.show()
#**************************************
# IMPRESSAO DE RESULTADOS
print 'TIPO_SIMULAÇÃO      =',TIPO
print 'TIPO_PLANO (TIPO=0) =',TIPO_K
print 'TIPO_COEFICIENTE    =',TIPO_COEF
print 'FORMA_EXPOENTE =',FORMA_EXPOENTE
print 'fator_cfl =',fator_cfl
print 'cfl  =',cfl
print 'rdz  =',rdz
print 'N    =',N
print 'Zo   =',zo
print 'teta =',teta*180/np.pi
print 'alfa =',alfa*180/np.pi
print 'k    =',k
print 'kx   =',kx
print 'ky   =',ky
print 'kz   =',kz
print 'fbx  =',fbx
print 'fbx2 =',fbx2
print 'fa   =',fa
print 'fb   =',fb
print 'fc   =',fc
print 'cha  =',ha
print 'chb  =',hb
print 'chz  =',hz
print 'cea  =',ea
print 'ceb  =',eb
print 'cez  =',ez
print
print 'ANISOTROPIA NO '+ texto1 + ' PARA TIPO =',TIPO
print 'Numero de pontos por comp_onda =',N
print 'fax3 =',fax3
print 'fbx3 =',fbx3
print 'fator_cfl =',fator_cfl
print 'cfl  =',cfl
print 'razão ds/dz =',rdz
if TIPO == 0:
    print'FATOR_AUMENTO =',FATOR_AUMENTO
print 'Nivel maximo = ',anivelmax
print 'Nivel minimo = ',anivelmin
print 'Nivel médio  = ',anivelmed
print 'Inverso do nivel médio  = ',fatormed
```

```
print 'Inverso do nivel mínimo = ',fatormin
print 'anisotropia_max (%) =',anisotropia_max
print 'anisotropia_yee (%) =',anteorica_yee
print 'anisotropia_hex (%) =',anteorica
print
print #nivel2

############################################################
#FINAL DO PROGRAMA
### FINAL APÊNDICE 10
############################################################
```