Péricles Lopes Machado

Desenvolvimento de Metodologias Automáticas para Obtenção do Parâmetro de Forma Ótimo para o Método RPIM

Belém - PA, Brasil

2012

Péricles Lopes Machado

Desenvolvimento de Metodologias Automáticas para Obtenção do Parâmetro de Forma Ótimo para o Método RPIM

DM. 22/2012
Dissertação apresentada para obtenção do Grau de Mestre em Engenharia Elétrica pela Universidade Federal do Pará.

Orientador:
Prof. Dr. Rodrigo Melo e Silva de Oliveira

INSTITUTO DE TECNOLOGIA
FACULDADE DE ENGENHARIA DA COMPUTAÇÃO
UNIVERSIDADE FEDERAL DO PARÁ

Belém - PA, Brasil

2012

Desenvolvimento de Metodologias Automáticas para Obtenção do Parâmetro de Forma Ótimo para o Método RPIM

Este trabalho foi julgado em ...../...../..... adequado para obtenção do grau de Mestre em Engenharia Elétrica, e aprovado na sua forma final pela banca examinadora.

Prof. Dr. Rodrigo Melo e Silva de Oliveira
(orientador - PPGEE/UFPA)
Universidade Federal do Pará

Prof. Dr. Victor Alexandrovich Dmitriev
(membro - PPGEE/UFPA)
Universidade Federal do Pará

Prof. Dr. Karlo Queiroz da Costa
(membro - PPGEE/UFPA)
Universidade Federal do Pará

Prof. Dr. Wilson Ricardo Matos Rabelo
(membro - ITEC/UFPA)
Universidade Federal do Pará

"*Nem todos que vagam estão perdidos.*"

J. R. R. Tolkien

Sumário

Lista de Símbolos

c	Fator de forma
C_0	Fator de forma otimizado
$\vec{E}$	Vetor Intensidade de Campo Elétrico
$\vec{H}$	Vetor Intensidade de Campo Magnético
$\vec{D}$	Vetor Densidade de Fluxo Elétrico
$\vec{B}$	Vetor Densidade de Fluxo Magnético
ε_o	Permissividade Elétrica do Vácuo
μ_o	Permeabilidade Magnética do Vácuo
ε	Permissividade Elétrica
μ	Permeabilidade Magnética
σ	Condutividade Elétrica
σ_α	Condutividades para UPML
t	Tempo
x, y e z	Coordenadas do Sistema Cartesiano
E_x, E_y e E_z	Componentes do Campo Elétrico
H_x, H_y e H_z	Componentes do Campo Magnético
D_x, D_y e D_z	Componentes de $\vec{D}$
B_x, B_y e B_z	Componentes de $\vec{B}$
$\frac{df}{d\alpha}$	Derivada de f em relação a α
$\frac{\partial f}{\partial \alpha}$	Derivada Parcial de f em relação a α
$\Omega(\overline{p}, k, V)$	Domínio de suporte do ponto $\overline{p} \in V$ com k pontos, onde V está contido num espaço métrico $E(V,d)$. V é um conjunto finito.
$\mathbb{R}^M$	Conjunto de todas M-tuplas $(x_1, x_2, ..., x_M)$, com $x_i \in \mathbb{R}$.
$\overline{p}_i$	Um ponto $\overline{p}_i$ definido em $\mathbb{R}^M$, $\overline{p}_i = (x_{i,1}, x_{i,2}, ..., x_{i,M})$, com $x_{i,k} \in \mathbb{R}, k = 1, 2, ..., M$.
$d_{e,M}(\overline{p}_1, \overline{p}_2)$	Métrica euclidiana definida em $\mathbb{R}^M$, $d_{e,M}(\overline{p}_1, \overline{p}_2) = \sqrt{(x_{1,1} - x_{2,1})^2 + (x_{1,2} - x_{2,2})^2 + ... + (x_{1,M} - x_{2,M})^2}$.

$\mathfrak{D}(V,k)$	$\mathfrak{D}(V,k) = \bigcup_{i=0}^{N-1} \Omega(\overline{p}_i,k,V)$ é a união de todos domínios de suporte de um conjunto finito V, N é o tamanho do conjunto V.
box	Mapa multidimensional usado para acelerar a busca realizada pelo *Line Sweep*.
$box_i(v_i)$	Mapa multidimensional de nível i com elementos com i-ésima coordenada igual a v_i. Usado no método *Line Sweep*.
$\Gamma(V)$	Densidade do conjunto V, $\Gamma(V) = N/(\Pi_{i=1}^{M}(x_{max,i} - x_{min,i}))$ é o número de pontos por unidade de "volume", considerando-se um espaço M-dimensional.
$\Phi(\overline{x})$	Função de base radial.
β	Parâmetro livre relativo a função de base radial.
r_{max}	Raio do subdomínio de suporte Ω.
$\vec{\nabla} \times \vec{A}$	Operador Rotacional de $\vec{A}$.
$\Psi(\overline{x})$	$\Psi(\overline{x}) = [\Psi_1(\overline{x}), \Psi_2(\overline{x}), ..., \Psi_N(\overline{x})]$, Funções de forma: vetor contendo os coeficientes de interpolação usados no RPIM. N é o número de pontos no subdomínio de suporte.
$\frac{\partial \Psi}{\partial v_\alpha}$	Derivada parcial de Ψ com relação a variável v_α

Lista de Figuras

Lista de Tabelas

Resumo

Neste trabalho, são propostas metodologias para otimização do parâmetro de forma local *c* do método RPIM (*Radial Point Interpolation Method*). Com as técnicas apresentadas, é possível reduzir problemas com inversão de matrizes comuns em métodos sem malha e, também, garantir um maior grau de liberdade e precisão para a utilização da técnica, já que se torna possível uma definição semi-automática dos fatores de forma mais adequados para cada domínio de suporte. Além disso, é apresentado um algoritmo baseado no *Line Sweep* para a geração eficiente dos domínios de suporte.

Palavras-chave: *Método de interpolação por funções radiais (RPIM), espalhamento eletromagnético, geometria computacional, Line Sweep, otimização de fator de forma.*

Abstract

In this thesis, a methodology is proposed for automatically (and locally) obtaining the shape factor c for the Gaussian basis functions, for each support domain, in order to increase numerical precision and mainly to avoid matrix inversion impossibilities. The concept of calibration function is introduced, which is used for obtaining c. The methodology developed was applied for a 2-D numerical experiment, which results are compared to analytical solution. This comparison revels that the results associated to the developed methodology are very close to the analytical solution for the entire bandwidth of the excitation pulse. The proposed methodology is called in this work Local Shape Factor Calibration Method (LSFCM).

1 *Introdução*

Métodos sem malha [2–4] são metodologias alternativas para a solução numérica de equações diferenciais. Estes métodos não exigem que seja fornecida uma estrutura com uma topologia previamente definida, como ocorre no FDTD (*finite difference time domain method*) e métodos tradicionais que necessitam de uma malha para descrever a geometria [5]. Isto permite uma maior liberdade para a representação de estruturas com geometrias complexas. Além disso, com métodos sem malha é possível obter mais fácil e rapidamente resultados precisos.

Na literatura existe uma grande diversidade de métodos que não exigem uma topologia previamente definida. Entre estes métodos, pode-se citar *Element-Free Galerkin method* [4], *Moving Least Square Reproducing Kernel Method* [4], *Smoothed Particle Electromagnetic Method* [4] e o *Radial Point Interpolation Method (RPIM)* [4].

Este trabalho aborda o método RPIM aplicado às equações de Maxwell no espaço 2-D, além de desenvolver e apresentar metodologias para resolver problemas presentes na técnica (como o problema de inversão de matrizes que surge devido a determinados posicionamentos espaciais de pontos em determinadas formas e o problema de construção eficiente de domínios de suporte em configurações de pontos genéricas).

1.1 Objetivos

Este trabalho possui os seguintes objetivos:

- Apresentar a formulação da técnica RPIM.
- Apresentar um algoritmo eficiente para geração de domínios de suporte baseado na técnica *Line Sweep* [6].
- Propor técnicas para minimizar erros de inversão de matrizes.

- Verificar a correção da técnica comparando resultados objetivos com resultados disponíveis na literatura.

1.2 Estrutura do trabalho

O texto é divido em três capítulos principais e dois apêndices.

O capítulo 2 apresenta a formulação RPIM para a solução numérica das equações de Maxwell, além de apresentar o equacionamento do método. Neste capítulo, também é apresentado um algoritmo baseado na técnica *Line Sweep* [6] para a construções dos domínios de suporte.

No capítulo 3, são apresentadas duas metodologias baseadas no *regula falsi* para aumentar a precisão da técnica RPIM e minimizar problemas de inversão de matrizes.

Por fim, no capítulo 4, são apresentados casos de validação para a implementação do RPIM e das metodologias apresentadas nesse trabalho.

No apêndice A, há uma implementação em GNU Octave da técnica RPIM e, no apêndice B, há um implementação, também em GNU Octave, das técnicas LSFCM e FLSFCM apresentadas no capítulo 3.

2 O método RPIM e a resolução das equações Maxwell

Um dos mais populares métodos numéricos empregados para a solução numérica das equações de Maxwell no domínio do tempo é o Método das diferenças finitas no domínio do tempo (*finite-difference time-domain method*-FDTD). Células de Yee são usadas para representar o espaço utilizando uma grade retangular estruturada [5], tal como foi proposto em 1966 [7]. Posteriormente, para se modelar superfícies incompatíveis com o sistema de coordenadas cartesiano com mais precisão, o uso de grades estruturadas e não estruturadas não-ortogonais foi proposto [5].

Recentemente, métodos sem malhas surgiram [2–4] para solucionar numericamente equações diferencias parciais (como as Equações de Maxwell) com grande flexibilidade em termos de representação geométrica. Isto é possível porque um conjunto de pontos V é usado para representar a região de análise (o conceito de células e elementos não está mais presente). Entre estes métodos estão o método de elementos livres de Galerkin [8], o *Moving Least Square Reproducing Kernel Method* [9], o *Smoothed Particle Electromagnetic Method* [10] e o *Radial Point Interpolation Method (RPIM)* [4] que é usado neste trabalho. Para o método RPIM, as componentes de campo são interpoladas localmente utilizando-se subconjuntos de pontos, chamados de *domínios de suporte* (conjunto dos k pontos mais próximos a um ponto de referência $\overline{x}_c$ [4]), como ilustrado na Fig. 2.1.

De modo geral, as coordenadas dos pontos são armazenadas em vetores e, já que nem células e nem elementos são empregados, os indíces dos vetores não são organizados utilizando-se as coordenadas espaciais como referência (como nas malhas estruturadas) e os pontos não são agrupados por elementos (como nas malhas não-estruturadas).

Por isso, um algoritmo geral é requerido para a geração automática dos domínios de suporte, de forma tão eficiente quanto possível. O método *Line Sweep*, que foi inicialmente usado para encontrar os dois pontos mais próximos entre si em um determinado conjunto [6], é adaptado neste capítulo e um eficiente algoritmo para encontrar os k pontos mais próximos a um ponto é

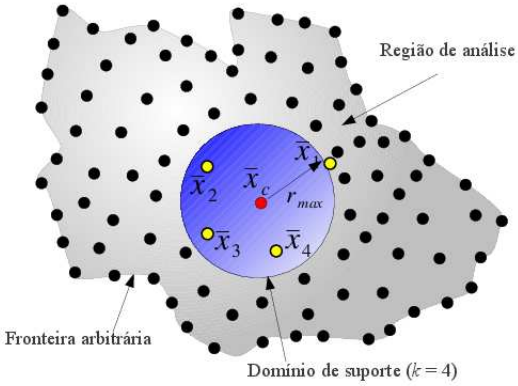


Figura 2.1: Região de análise e domínio de suporte para $\bar{x}_c$.

obtido.

O algoritmo apresentado neste capítulo possui complexidade assintótica de tempo $O(Nk(\log N)^{M-1})$, onde N é o numero de pontos na região de análise, $k \ll N$ e M é o número total de dimensões espaciais do problema. Desta forma, há um ganho de performance significante quando comparado com técnicas baseadas em *kd-trees* [11, 12], que apresenta complexidade temporal de $O(NkN^{1-1/M})$.

Além do algoritmo de geração de domínios de suporte, o método RPIM e sua aplicação nas equações de Maxwell são apresentados a seguir.

2.1 Conceitos Básicos

Nesta seção, conceitos básicos são apresentados para um melhor entendimento do algoritmo para a obtenção dos domínios de suporte.

2.1.1 Espaços Métricos

Uma métrica em V é uma função $d : V \times V \longrightarrow \mathbb{R}$ com as seguintes propriedades [13, 14]:

1. $d(\bar{x},\bar{x}) = 0$;
2. $d(\bar{x},\bar{y}) > 0$, se $\bar{x} \neq \bar{y}$;
3. $d(\bar{x},\bar{y}) = d(\bar{y},\bar{x})$;
4. $d(\bar{x},\bar{z}) \leq d(\bar{x},\bar{y}) + d(\bar{y},\bar{z})$,

em que $\overline{x},\overline{y},\overline{z} \in V$.

Um espaço métrico $E(V,d)$ é definido a partir de um conjunto V e de uma métrica d. Os elementos de $E(V,d)$ são pontos. Métricas são generalizações do conceito de distância entre pontos, o que permite que algoritmos que precisem de uma noção de distância possam ser construídos para diversos tipos de conjuntos.

Neste trabalho, a métrica usada é a $E(\mathbb{R}^M, d_{e,M})$, onde $d_{e,M}$ é a métrica euclidiana definida em (2.1). Entretanto, o algoritmo de geração de domínios apresentado é definido num espaço métrico genérico.

$$d_{e,M}(\overline{p}_1,\overline{p}_2) = \sqrt{(x_{1,1}-x_{2,1})^2 + ... + (x_{1,M}-x_{2,M})^2}, \tag{2.1}$$

onde $\overline{p}_1 = (x_{1,1}, x_{1,2}, ..., x_{1,M})$ e $\overline{p}_2 = (x_{2,1}, x_{2,2}, ..., x_{2,M}) \in \mathbb{R}^M$.

2.1.2 Domínios de Suporte

O domínio de suporte [4] $\Omega(\overline{x}_c, k, V)$ para um ponto $\overline{x}_c \in (V \subset E(V,d))$ é um conjunto finito de k pontos, com $k < N$, em que os pontos $\overline{x}_i \in V, i = 1,2,...,N$, onde N é o numero de elementos em V. Os elementos em $\Omega(\overline{x}_c, k, V)$ são os k pontos $(\overline{x}_i \neq \overline{x}_c) \in V$ mais próximos de $\overline{x}_c$ de acordo com a métrica d de $E(V,d)$. O ponto $\overline{x}_c$ é definido como o centro do domínio de suporte $\Omega(\overline{x}_c, k, V)$.

A união de todos domínios de suporte $\Omega(\overline{p}_i, k, V)$ de um conjunto V, $\mathfrak{D}(V,k) = \bigcup_{i=0}^{N-1} \Omega(\overline{p}_i, k, V)$, com $\overline{p}_i \in V$, é chamado o domínio de V.

A Fig. 2.1 é um exemplo de domínio de suporte, em que r_{max} é a distância de x_c para o ponto mais distante em Ω, de acordo com a métrica $d_{e,2}$.

2.2 O Método *Line Sweep*

O método *Line Sweep* é introduzido para resolver o problema do par de pontos mais próximos em um conjunto. É assumido que o conjunto $V = W_1 \times W_2 \times ... \times W_M$ é dado pelo produto de $M > 1$ conjuntos W_i, com $i = 1,2,...,M$.

O par de pontos mais próximos pode ser encontrado por um algoritmo trivial com complexidade temporal $O(N^2)$ se todos pares de pontos do conjunto V são considerados. Entretanto, conforme será mostrado a seguir, este problema pode ser resolvido com complexidade $O(NlogN)$.

Para o algoritmo *Line Sweep*, $\bar{x}_p \in V$ é o pivô do passo atual, $box_M(V_M)$ é um conjunto auxiliar, $h > 0$ é o raio de busca, $\bar{x}_k \in (box_M(x_M) \subset V)$, $x_{p,1}$ e $x_{k,1}$ são as primeiras coordenadas de $\bar{x}_p$ e $\bar{x}_k$, respectivamente.

Inicialmente, o conjunto V é ordenado em ordem crescente usando a coordenada x_1 como referência ($x_1 \in \bar{x} = (x_1, x_2, ..., x_M) \in V$). Aqui, é assumido que $x_i \in W_i$ e $i = 1, 2, ..., M$. Um conjunto auxiliar box_1, com seus elementos ordenados usando a coordenada x_2 como referência, é tamém criada. O conjunto box_1 é construído de tal forma que cada valor assumido por x_2, denotado aqui por v_2, é usado como chave que aponta para um conjunto $box_2(v_2)$. Da mesma forma, os elementos em $box_2(v_2)$, daqueles cujas segundas coordenadas são v_2, são ordenados usando x_3 como referência. E, por sua vez, quando $box_2(v_2)$ é considerado, cada valor assumido por x_3, denotado pro v_3, aponta para um $box_3(v_3)$, com elementos ordenados usando x_4 como referência (suas terceiras coordenadas são v_3). O processo de construção deste esquema de apontamento segue até que $box_M(v_M)$ é obtida. $box_M(v_M)$ aponta para elementos em que $|x_{k,1} - x_{p,1}| < h$, dos quais a M-ésima coordenada é dada por v_M. A Fig. 2.2 ilustra esta estrutura para o conjunto $V = \{(1,1,1), (1,1,2), (1,2,1), (1,2,2), (2,1,1), (2,1,2), (2,2,1), (2,2,2)\}$, considerando $h = 4$.

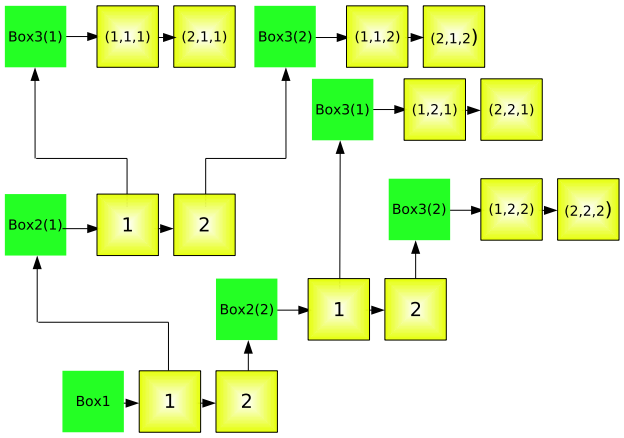


Figura 2.2: Uma estrutura *box* para oito pontos em $\mathbb{R}^3$, com $h = 4$.

Esta estrutura acelera a pesquisa pelo par de pontos mais próximos porque a busca é limitada à um hipercubo com arestas medindo h. Adicionalmente, a cada iteração do algoritmo, h é reduzido assumindo-se o valor corrente da mínima distância. Na Fig. 2.3, é ilustrado um quadrado (um hipercubo no espaço 2-D) em que a busca é limitada em uma dada iteração (espaço $\mathbb{R}^2$).

A Fig. 2.4 ilustra o algoritmo *Line Sweep* apresentado para resolver o presente problema. Ele é profundamente baseado no conceito de iteradores [15], isto é, um ponteiro para um *box* ou ponto.

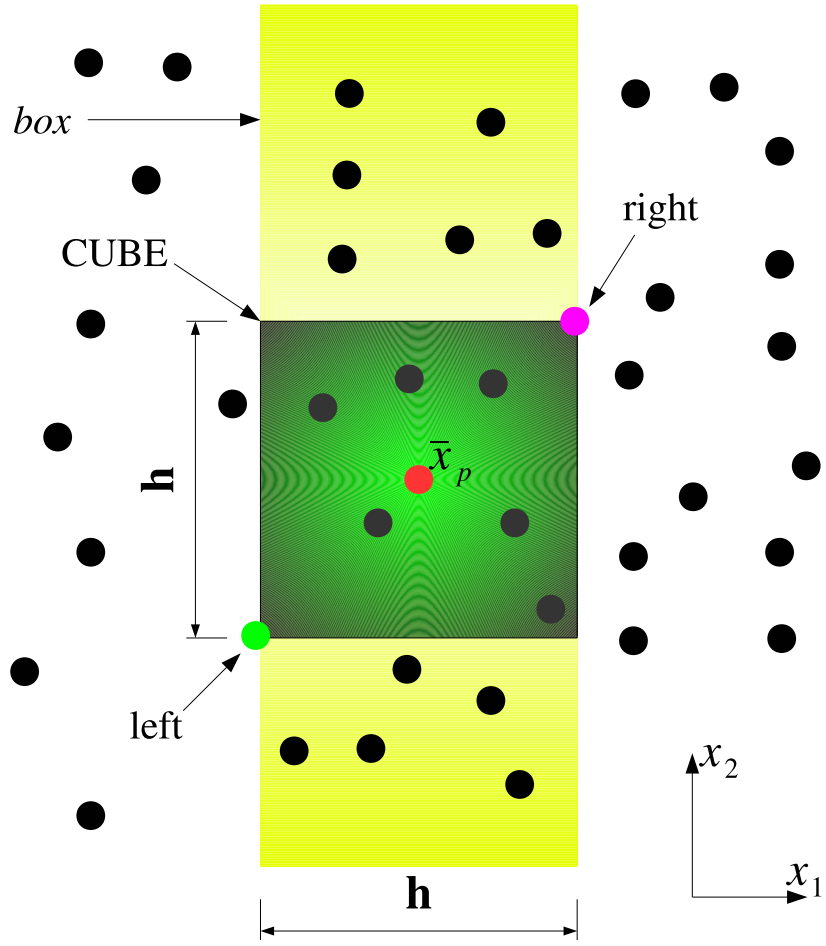


Figura 2.3: Um quadrado em cada pesquisa do *Line Sweep* é limitada em uma dada iteração (espaço $\mathbb{R}^2$).

Na Fig. 2.4, é assumido que *box* é um $multi-map$ [15] contendo os seguintes métodos:

- insert($\overline{x}$): insere uma *box* de nível mais alto ou um ponto $\overline{x}$ no conjunto box_l;
- remove($\overline{x}$): remove *box* de nível mais alto ou um ponto $\overline{x}$ do conjunto box_l;
- lower_bound(v): retorna o primeiro iterador da *box* de nível l com $x_{l+1} \geq v$;
- upper_bound(v): retorna o primeiro iterador da *box* de nível l com $x_{l+1} > v$;
- $box(v)$: retorna a *box* ou elemento apontado por v. Se o elemento é inexistente, um conjunto vazio é criado em v.

1: *Line-Sweep*(V):
{V é um conjunto ordenado em ordem crescente de arcodo com $x_{k,1}$ de $\overline{x}_k = (x_{k,1}, x_{k,2}, ..., x_{k,M}) \in V$, com $\overline{x}_i \in W_i$, $i = 1, 2, ..., M$ e $k = 0, 1, ..., N-1$. Here, $d(\overline{x}, \overline{y})$ é a métrica associada com o conjunto V.}
2: $V = \{\overline{x}_0, \overline{x}_1, ..., \overline{x}_{N-1}\}$
3: $h' \leftarrow |x_{0,1} - x_{1,1}|$, onde $\overline{x}_0, \overline{x}_1 \in V$
4: $sol \leftarrow (\overline{x}_0, \overline{x}_1)$
5: $left \leftarrow 0$
6: $right \leftarrow 0$
7: $box \leftarrow \emptyset$
8: $p \leftarrow 0$
9: $min\_dist \leftarrow d(\overline{x}_0, \overline{x}_1)$
{N é o tamanho de V.}
10: **while** $p < N$ **do**
11: $h' \leftarrow min\_dist$
12: **while** $left < p$ e $|x_{left,1} - x_{p,1}| > h'$ **do**
13: **remove-element**(box,$x_{left,2}$,1,$\overline{x}_{left}$,M)
14: $left \leftarrow left + 1$
15: **end while**
16: **while** $right < N$ e $|x_{right,1} - x_{p,1}| \leq h'$ **do**
17: **insert-element**(box,$x_{right,2}$,1,$\overline{x}_{right}$,M)
18: $right \leftarrow right + 1$
19: **end while**
20: $CUBE \leftarrow \emptyset$ {Constrói um hipercubo (arestas: $h = 2h'$)}
21: **build-hypercube**(box,1,$\overline{x}_p$,h',$CUBE$,M)
22: **for** todos elementos $\overline{q}$ em $CUBE$ **do**
23: **if** $d(\overline{q}, \overline{x}_p) < min\_dist$ e $\overline{q} \neq \overline{x}_p$ **then**
24: $min\_dist \leftarrow d(\overline{q}, \overline{x}_p)$
25: $sol \leftarrow (\overline{x}_p, \overline{q})$
26: **end if**
27: **end for**
28: $p \leftarrow p + 1$;
29: **end while**
30: *sol* é o par de pontos mais próximos em V e *min_dist* é a distância entre os elementos apontados por *sol*.
31: **end** *Line-Sweep*

Figura 2.4: O método *Line Sweep* aplicado para encontrar o par de pontos mais próximos no conjunto V.

Procedimentos auxiliares são dados pelas Figs. 2.5, 2.6 e 2.7, onde é assumido ou definido que:

- *box* é um *iterador* para um *box* de nível l;
- v é uma chave (coordenada) de interesse corrente;

- l é o nível corrente;
- $\bar{x}$ é o ponto que será inserido ou removido;
- M é a dimensão do problema, isto é, o numero total de coordenadas dos pontos em V;
- $\bar{x}_p$ é o ponto pivô da iteração corrente *Line Sweep*;
- $CUBE$ é o hipercubo em que a pesquisa do *Line Sweep* é realizada;
- h é a largura da aresta do hipercubo em que ocorre a busca;
- $*k$ representa o conteúdo do iterador k.

```
1:  insert-element (box,v,l,x̄,M):
2:  if  l = M then
3:     box.insert(x̄)
4:  else
5:     if l + 2 ≤ M then
6:        w ← x_{l+2}
7:     else
8:        w ← 0
9:     end if
10:    insert-element(box(v),w,l + 1,x̄,M)
11: end if
12: end insert-element
```

Figura 2.5: Procedimento auxiliar para inserir um elemento em uma *box*.

```
1:  remove-element(box,v,l,x̄,M):
2:  if l = M then
3:     box.remove(x̄) {remove x̄ se ele está presente em box.}
4:  else
5:     if l + 2 ≤ M then
6:        w ← x_{l+2}
7:     else
8:        w ← 0
9:     end if
10:    remove-element(box(v),w,l + 1,x̄,M)
11:    if box(v) = ∅ then
12:       box.remove(v) {remove box apontado por v.}
13:    end if
14: end if
15: end remove-element
```

Figura 2.6: Procedimento auxiliar para remover um elemento de *box*.

1: **build-hypercube**(*box*,*l*,$\bar{x}_p$,*h*,*CUBE*,*M*):
2: **if** $l = M$ **then**
3: $CUBE \leftarrow CUBE \bigcup box$ {adiciona o hipercubo *CUBE* todo ponto em $box(v)$.}
4: **else**
5: $a \leftarrow box.lower\_bound(x_{p,l+1} - h)$
6: $b \leftarrow box.upper\_bound(x_{p,l+1} + h)$
7: $k \leftarrow a$
8: **while** $k \neq b$ **do**
9: **build-hypercube** ($*k$, $l+1$, $\bar{x}_p$, h, $CUBE$, M)
10: $k \leftarrow k+1$ {percorre a *box* no nível l.}
11: **end while**
12: **end if**
13: **end build-hypercube**

Figura 2.7: Procedimento para construir o hipercubo.

Na Fig. 2.4, as operações *remove-element*, *build-hypercube* e *insert-elment* são algoritmos com complexidade de tempo $O((\log N)^{M-1})$ [16], baseados em àrvores de busca binárias. Adicionalmente, o numero de elementos de *CUBE* é significativamente reduzido a cada passo e todo ponto $\bar{x}_p$ é inserido ou removido uma vez da *box*. Por isso, a complexidade temporal do algoritmo apresentado pela Fig. 2.4 é $O(N(\log N)^{M-1})$ [16]. Fig. 2.3 é uma ilustração da execução do algoritmo *Line Sweep* em um subconjunto do $\mathbb{R}^2$. O termo *Line Sweep* é devido ao fato que, durante sua execução, o ponto $\bar{x}_p$ origina uma linha que se move ao longo do eixo formado pela primeira coordenada de $\bar{x} \in V$.

Na próxima seção, o método *Line Sweep* é adaptado para construir o domínio de suporte $\mathfrak{D}(V,k)$.

2.3 O método *Line Sweep* aplicado para construir o domínio de suporte de um conjunto V

Uma das principais características do algoritmo apresentado pela Fig. 2.4 é que todos pontos $\bar{q}$ com distâncias $d(\bar{q},\bar{x}_p) \leq h$ são mantidos em *CUBE*. Por isso, para construir $\mathfrak{D}(V,k)$, basicamente, tem-se de definir, para todo ponto $\bar{x}_p$, que valor h deve assumir de tal forma que $\Omega(\bar{x}_p,k,V) \subset CUBE$.

Uma estimativa inicial apropriada para h pode ser obtida quando considera-se uma densidade $\Gamma(V)$. Neste trabalho, a densidade considerada é dada por $\Gamma(V) = N/(\Pi_{i=1}^{M}(x_{max,i} - x_{min,i}))$, onde $x_{max,i}$ é o máximo valor que a coordenada i assume no conjunto V e $x_{min,i}$ é seu valor mínimo. Deste modo, $\Gamma(V)$ fornece o numero de pontos por unidade de hipervolume

1: **build-domain**(V,Λ,k):
{V é um conjunto ordenado em ordem crescente de acordo com $x_{j,1}$ de $\bar{x}_j = (x_{j,1}, x_{j,2}, ..., x_{j,M}) \in V$, com $\bar{x}_i \in W_i$, $i = 1,2,...,M$ e $j = 0,1,...,N-1$. $d(\bar{x},\bar{y})$ é a métrica associada ao conjunto V. }
2: $V = \{\bar{x}_0, \bar{x}_1, ..., \bar{x}_{N-1}\}$
3: $h_o \leftarrow \sqrt[M]{k/\Gamma(V)}$
4: $left \leftarrow 0$
5: $right \leftarrow 0$
6: $box \leftarrow \emptyset$
7: $p \leftarrow 0$
8: $\mathfrak{D}(V,k) \leftarrow \emptyset$ {N é o tamanho de V.}
9: **while** $p < N$ **do**
10: $h' \leftarrow h_o$;
11: $AUX \leftarrow \emptyset$;
12: **while** AUX contém menos que $k+1$ elementos **do**
13: **while** $left < p$ e $|x_{left,1} - x_{p,1}| > h'$ **do**
14: **remove-element**(box,$x_{left,2}$,1,$\bar{x}_{left}$,M)
15: $left \leftarrow left + 1$
16: **end while**
17: **while** $right > p$ e $|x_{right,1} - x_{p,1}| > h'$ **do**
18: **remove-element**(box,$x_{right,2}$,1,$\bar{x}_{right}$,M)
19: $right \leftarrow right - 1$
20: **end while**
21: **while** $right < N$ e $|x_{right,1} - x_{p,1}| \leq h'$ **do**
22: **insert-element**(box,$x_{right,2}$,1,$\bar{x}_{right}$,M)
23: $right \leftarrow right + 1$
24: **end while**
25: **while** $left > -1$ e $|x_{left,1} - x_{p,1}| \leq h'$ **do**
26: **insert-element**(box,$x_{left,2}$,1,$\bar{x}_{left}$,M)
27: $left \leftarrow left - 1$
28: **end while**
29: $CUBE \leftarrow \emptyset$
30: **build-hypercube**(box,1,$\bar{x}_p$,h',$CUBE$,M) {constrói um hipercubo com arestas medindo $h = 2h'$.}
31: $AUX \leftarrow CUBE$
32: $h' \leftarrow (1+\Lambda)h'$
33: **end while**
34: $\Omega(\bar{x}_p, k, V) \leftarrow$ os k elementos $(\bar{c} \neq \bar{x}_p) \in AUX$ mais próximos a $\bar{x}_p$ de acordo com a métrica d associada à V
35: $\mathfrak{D}(V,k) \leftarrow \mathfrak{D}(V,k) \bigcup \Omega(\bar{x}_p, k, V)$
36: $p \leftarrow p + 1$
37: **end while**
38: $\mathfrak{D}(V,k)$ é domínio de V.
39: **end build-domain**

Figura 2.8: O algoritmo apresentado para construir todos domínios de suprote de V a partir da adaptação do método *Line Sweep*.

quando o espaço M-dimensional é tratado. Já que deseja-se determinar os k pontos mais próximo à $\bar{x}_p$, uma estimativa inicial adequada é $h = \sqrt[M]{k/\Gamma(V)}$. Se h não for suficiente para assegurar que $CUBE$ contém pelo menos k elementos, então h é aumentado em Λ porcento e $CUBE$ é reconstruído. Este procedimento é repetido enquanto $CUBE$ contiver menos do que k elementos. Quando $CUBE$ contém menos que k elementos, os k elementos $(\bar{c}_k \neq \bar{x}_p) \in CUBE$ mais próximos a $\bar{x}_p$ são inseridos no domínio de suporte $\Omega(\bar{x}_p, k, V)$, e, por sua vez, $\Omega(\bar{x}_p, k, V)$ é inserido no domínio de V ($\mathfrak{D}(V,k)$).

O desempenho do algoritmo está altamente relacionado à distribuição geométrica dos pontos no espaço. Para um conjunto cuja distribuição tende a ser homogênea, o fator de crescimento Λ é pequeno. Neste caso, h tende a ser inalterado. Por isso, a performance do algoritmo tende a ser constante e sua complexidade é aproximadamente $O(Nk(\log N)^{M-1})$. Para casos onde o conjunto tende a ser heterogêneo, a complexidade de tempo é relacionada ao número de vezes que h' é aumentado. Então, para esses casos a complexidade de tempo é $O(Nk\alpha(V)(\log N)^{M-1})$, onde $\alpha(V)$ é o numero de vezes que h' (Fig. 2.8) é aumentado. Para os testes realziados neste trabalho, $\Lambda = 0,1$ foi adequado para se obter uma boa perfomance.

2.4 Análise de performance do algoritmo proposto

Nesta seção, a performance do algoritmo proposto (Fig. 2.8) é comparado com um método trivial concebido para construir domínios de suporte de V (Fig. 2.9). Pode-se facilmente ver que a complexidade de tempo do algoritmo apresenta pela Fig. 2.9 é $O(N^2 \log N)$.

Para medir o tempo de execução, ambos algoritmos foram implementados em C++ e executaram em um turion64x2 com 1 GB de RAM. Os conjuntos de teste foram criados aleatoriamente com 100 a 10^6 pontos, k variou de 1 a 32. Cada conjunto de testes foi processado por ambos algoritmos e os tempos de processamento medidos são mostrados nas tabelas 2.1 e 2.2, que ilustram a significante redução de tempo alcançada.

Tabela 2.1: Tempo computacional requerido pelo algoritmo da Fig. 2.8

	Tempo de execução (segundos) k					
N	1	2	4	8	16	32
121	0.02	0.024	0.025	0.027	0.031	0.039
1100	0.03	0.031	0.038	0.041	0.082	0.092
10^4	0.141	0.172	0.180	0.247	0.448	0.723
10^5	1.186	1.249	1.578	2.186	3.568	7.404
10^6	15.642	16.845	17.714	26.042	45.064	79.479

1: **build-domain**(V,Λ,k):
   {V é um conjunto ordenado em ordem crescente por $x_{j,1}$ de $\bar{x}_j = (x_{j,1}, x_{j,2}, ..., x_{j,M}) \in V$, com $\bar{x}_i \in W_i$, $i = 1,2,...,M$ e $j = 0,1,...,N-1$. $d(\bar{x},\bar{y})$ associado ao conjunto V. }
2: $V = \{\bar{x}_0, \bar{x}_1, ..., \bar{x}_{N-1}\}$
3: $p \leftarrow 0$
4: $\mathfrak{D}(V,k) \leftarrow \emptyset$ {N é o tamanho de V.}
5: **while** $p < N$ **do**
6: $i \leftarrow 0$
7: $AUX \leftarrow \emptyset$
8: **while** $i < N$ **do**
9: **if** $i \neq p$ **then**
10: $AUX \leftarrow AUX \bigcup \bar{x}_i$ {AUX é um conjunto organizado em ordem crescente de acordo com a distância de $\bar{x}_i$ a $\bar{x}_p$.}
11: **end if**
12: $i \leftarrow i+1$
13: **end while**
14: $i \leftarrow 0$
15: **while** $i < k$ **do**
16: $\Omega(\bar{x}_p,k,V) \leftarrow \Omega(\bar{x}_p,k,V) \bigcup AUX_i$ {AUX_i é o i-ésimo elemento de AUX.}
17: $i \leftarrow i+1$
18: **end while**
19: $\mathfrak{D}(V,k) \leftarrow \mathfrak{D}(V,k) \bigcup \Omega(\bar{x}_p,k,V)$
20: $p \leftarrow p+1$
21: **end while**
22: $\mathfrak{D}(V,k)$ é o domínio de V.
23: **end build-domain**

Figura 2.9: Algoritmo trivial para construção de domínios de suporte de V.

Para se obter a complexidade de tempo assintótica $O(Nk(\log N)^{M-1})$, é essencial observar o modo que a indexação das *boxes* é tratado Inicialmente, deve-se observar que existem conjuntos V cujos pontos nunca compartilham coordenadas. Isto deteriora o algoritmo apresentado porque sua performance está associada com a eficiência das buscas de intervalos para cada nível da *box*. Uma forma de resolver este problema, é criar um indíce inteiro w a partir do numero real de interesse v e usar este inteiro para indexar v. Como a distância média Δ_m entre os pontos de V é conhecida, têm-se que

$$w = \text{round}(v/\Delta_m). \tag{2.2}$$

Além de assegurar uma implementação prática do algoritmo proposto, a indexação baseada em inteira obtida pelo uso de (2.2) melhora a performance de execução média das buscas.

Tabela 2.2: Tempo de execução requerido pelo algoritmo da Fig. 2.9

	Tempo de execução (segundos) k					
N	1	2	4	8	16	32
121	0.029	0.031	0.035	0.042	0.043	0.035
1100	1.031	1.016	1.123	0.988	1.011	1.097
10^4	103.2	106.3	107.5	107.8	107.3	104.7
10^5	10^4	10^4	10^4	10^4	10^4	10^4
10^6	10^6	10^6	10^6	10^6	10^6	10^6
O tempo para $N = 10^5$ e $N = 10^6$ são estimados baseado na complexidade do algoritmo						

2.5 Utilização do método RPIM para resolução das equações de Maxwell

Considere uma função $u(\bar{x})$ no espaço. Esta função pode ser interpolada em um domínio de suporte Ω centrado em $\bar{x}$ por

$$u(\bar{x}) = \sum_{i=1}^{k} r_i(\bar{x})a_i + \sum_{j=1}^{M} p_j(\bar{x})b_j = R^T(\bar{x})a + P^T(\bar{x})b, \tag{2.3}$$

em que $r_i(\bar{x}) = e^{-c(r/r_{max})^2}$ é uma função de base Gaussiana (ou uma outra função de base radial qualquer), $r = \sqrt{(x-x_i)^2 + (y-y_i)^2}$, $c > 0$ é o fator de forma e r_{max} é o valor máximo assumido por r em Ω. Aqui, $\bar{x}_i$ é o i-ésimo nó de Ω, $i = 1,2,...,k$ e $P^T(\bar{x})$ é uma função polinomial com M termos. Neste trabalho, $P^T(\bar{x})$ é dado por $[1,x,y]$ e $M = 3$. Deve-se notar que o fator de forma c é um parâmetro livre.

Quando (2.3) é considerada para todos nós em Ω, obtém-se a equação matricial

$$U_s = R_o a + P_o b, \tag{2.4}$$

onde, $R_o = [R^T(\bar{x}_1), R^T(\bar{x}_2), ..., R^T(\bar{x}_k)]^T$, com $\bar{x}_i \in \Omega$, $i = 1...k$ e U_s contém todos valores assumidos por $u(\bar{x}_i)$.

Para assegurar que uma solução única seja obtida para a e b em (2.4), a condição $P_o^T a = 0$ é imposta [2]. Com alguma manipulação algébica, pode-se mostrar que

$$b = [P_o^T R_o^{-1} P_o]^{-1} P_o^T R_o^{-1} U_s = S_b U_s \tag{2.5}$$

e

$$a = (R_o^{-1} - R_o^{-1} P_o S_b) U_s = S_a U_s. \tag{2.6}$$

Deste modo, (2.3) pode ser escrita como

$$u(\bar{x}) = [R^T(\bar{x})S_a + P^T(\bar{x})S_b]U_s = \Psi(\bar{x})U_s, \tag{2.7}$$

em que $\Psi(\bar{x})$ é um vetor contendo amostras de funções de forma associados a cada nó i em Ω. É importante mecionar que as funções de forma em Ω devem satisfazer a propriedade do delta de Kronecker [2, 17].

Já que S_a e S_b são matrizes constantes (porque as coordenadas dos nós são fixas), a derivada parcial de $\Psi_l(\bar{x})$ com respeito à v é dada por

$$\frac{\partial \Psi_l}{\partial v} = \sum_{i=1}^{k} \frac{\partial R_i}{\partial v} S^a_{i,l} + \sum_{j=1}^{M} \frac{\partial P_j}{\partial v} S^b_{j,l}, \tag{2.8}$$

onde $l = 1...k$, $\frac{\partial \Psi}{\partial v} = [\frac{\partial \Psi_1}{\partial v}, \frac{\partial \Psi_2}{\partial v}, ..., \frac{\partial \Psi_k}{\partial v}]$, $S^a_{i,l}$ é o elemento da matriz S_a indexado por (i,l), $S^b_{j,l}$ é o elemento de S_b indexado por (j,l) e $v = x$ ou $v = y$.

É importante ressaltar que que S_a e S_b (e portanto Ψ) dependem somente das coordenadas espaciais e de c. Logo, a geometria e a escolha apropriada do parâmetro de forma c são de grande importância para a precisão do método.

Finalmente, a derivada parcial de u com respeito à v pode ser expressado por

$$\frac{\partial u}{\partial v} = \frac{\partial \Psi}{\partial v} U_s = \sum_{i=1}^{k} \frac{\partial \Psi_i}{\partial v} u_{s,i}. \tag{2.9}$$

Neste trabalho, as equações de Maxwell são resolvidas no modo TMz [5]. As derivadas espaciais associadas são aproximadas por (2.9) e as equações de atualziação de campo

$$H^{n+\frac{1}{2}}_{x,i} = H^{n-\frac{1}{2}}_{x,i} - \frac{\Delta t}{\mu} \left(\sum_j E^n_{z,j} \partial y \Psi_j \right), \tag{2.10}$$

$$H^{n+\frac{1}{2}}_{y,i} = H^{n-\frac{1}{2}}_{y,i} + \frac{\Delta t}{\mu} \left(\sum_j E^n_{z,j} \partial x \Psi_j \right), \tag{2.11}$$

e

$$E^{n+1}_{z,i} = E^n_{z,i} + \frac{\Delta t}{\varepsilon} \left(\sum_j H^{n+\frac{1}{2}}_{y,j} \partial x \Psi_j - \sum_j H^{n+\frac{1}{2}}_{x,j} \partial y \Psi_j \right) \tag{2.12}$$

são obtidas. Em (2.10)-(2.12), diferenças finitas centradas são usadas para aproximar as derivadas em relação ao tempo.

A formulação UPML (Uniaxial-Perfectly Matched Layer) desenvolvida por Gedney [18] foi usada para truncagem do domínio de análise. Para garantir uma maior simetria dos domínios de suporte na borda externa da região absorvente da região absorvente, foram adicionadas 10 camadas extras de metal, tal como ilustrada na Fig. 2.10(a).

A Fig. 2.10(b) ilustra o problema de simetria que ocorre quando as camadas metálicas não são empregadas, compromentendo a interpolação das funções de interesse no centro do domínio de suporte que estão nos limites do domínio de análise.

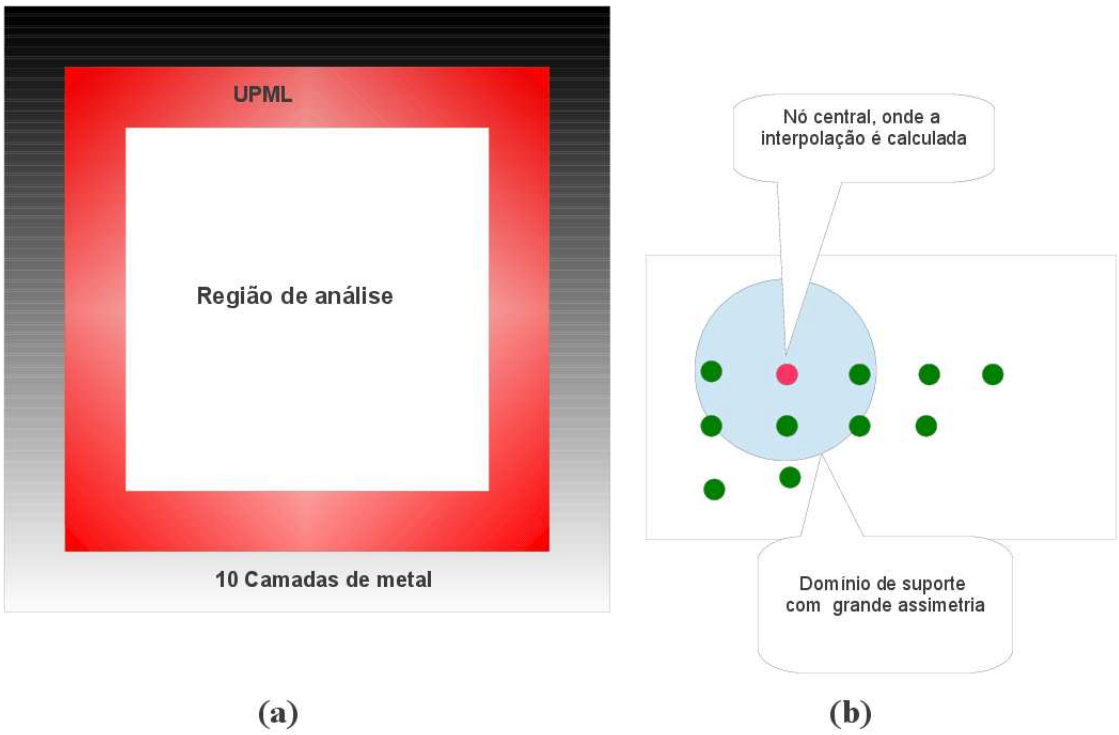


Figura 2.10: (a) Camadas de metal adicionadas à região de análise para garantir maior simetria dos domínios de suporte. (b) Assimetria presente nos domínios de suporte quando não há camadas de metal adicionais.

2.6 Considerações finais

Neste capítulo, foi apresentada uma metodologia para geração de domínios de suporte para um conjunto de pontos cuja estrutura não é conhecida *a priori*. É importante ressaltar que essa geração pode ser feita de forma mais eficiente se os domínios forem construídos paralelamente à construção do conjunto de pontos que representa a região de análise. Além disso, a formulação

RPIM para solução numérica das equações de Maxwell foi apresentada.

Com base no que foi exposto, apresenta-se a seguir a metodologia proposta neste trabalho para otimização do parâmetro de forma livre presente nas funções de base do método RPIM.

Uma implementação do RPIM é encontrada no Apêndice A.

3 *Uma metodologia automática para obtenção de parâmetros ótimos para os fatores de forma do RPIM*

Neste capítulo, uma metodologia é proposta para automaticamente, e localmente, obter o parâmetro de forma *c* para as funções de base Gaussianas, para cada domínio de suporte, de forma que a precisão numérica seja aumentada e principalmente para evitar matrizes não-inversíveis. O conceito de função de calibração é apresentado, que é usado na obtenção de *c*. A metodologia desenvolvida é aplicada em um experimento numérico 2-D, e os resultados são comparados com a solução analítica. Esta comparação mostra que os resultados associados com a metodologia desenvolvida são muito próximos aos da solução analítica para a banda inteira do pulso de excitação. A metodologia proposta é chamada neste trabalho de Método para Calibração do Fator de Forma Local (*Local Shape Factor Calibration Method* - LSFCM).

3.1 Introdução

Funções Gaussianas, multiquadráticas, logarítmicas e funções splines são usadas no método RPIM como função base que dependem de um paramêtro arbitrário [19]. Particularmente, bases Gaussianas dependem de um parâmetro livre *c*, conhecido como fator de forma, que afeta a precisão da interpolação de funções de interesse em um determinado domínio de suporte. Nos trabalhos anteriores [2–4], o parâmetro *c* é definido globalmente geralmente como $c = 0,01$. Entretanto, este valor não é adequado para todos domínios de suporte, devido à perda de precisão e especialmente devido à impossibilidades de inversão de matrizes [20]. Este problema tem sido tratado na literatura utilizando-se métodos como o algoritmo *Leave-One-Out-Cross-Validation* (LOOCV) [17,19,21], que calculam um parâmetro de forma global por meio de uma análise estatística.

Para melhorar a precisão do método RPIM, este capítulo apresenta uma formulação para computar *c* de uma forma automática e local para cada domínio de suporte, de forma que os

erros de interpolação são reduzidos e as dificuldades para as inversões de matrizes são minimizados. Isto é conseguido utilizando-se um sinal de alta frequência, aqui chamado de *função de calibração*. Deste modo, o método proposto é chamado Método para Calibração do Fator de Forma Local (*Local Shape Factor Calibration Method* - LSFCM).

3.2 O Método para Calibração do Fator de Forma Local

Trabalhos anteriores consideram c próximo de zero como um parâmetro global [1, 21] ($c = 0,01$ é frequentemente usado). Apesar de que pequenos valores de c poderem gerar valores altamente precisos, nem sempre é possível computar os valores interpolados da função devido a dificuldades de inversão de matrizes [19]. Isto ocorre porque quando um nó é colocado próximo de outro em Ω, como ilustrado na Fig. 3.1, e c é próximo de zero, as função de base gaussianas tendem a ser constantes (tendendo para a unidade) entre os nós mencionados e suas matrizes associadas tendem a ser não-inversíveis. Matematicamente, considerando-se que R_0 em forma forma expandida é dada por [19]

$$R_o = \begin{pmatrix} R_o^{1,1} & R_o^{1,2} & \dots & R_o^{1,k} \\ R_o^{2,1} & R_o^{2,2} & \dots & R_o^{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ R_o^{k,1} & R_o^{k,2} & \dots & R_o^{k,k} \end{pmatrix}, \tag{3.1}$$

em que $R_0^{i,j} = e^{-c(r_{i,j}/r_{max})^2}$ e $r_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$, é observável que para o caso da Fig. 3.1, as distâncias $r_{1,n}$ e $r_{2,n}$ (dos nós 1 e 2 para o nó n, respectivamente), com $n > 2$, são aproximadamente iguais. Desta forma, é evidente que $R_0^{1,n} \approx R_0^{2,n}$. Além disso, quando $c \approx 0, R_0^{2,1} \approx R_0^{1,2} \approx e^0 = 1$. Observando que $R_0^{1,1} = R_0^{2,2} = 1$, é fácil visualizar que a situação descrita pela Fig. 3.1 faz com que as linhas um e dois de (3.1) sejam quase linearmente dependentes, gerando dificuldades de inversão para R_0, tornando-se difícil calcular (2.5) e (2.6).

Nesses casos, a função base Gaussiana deve apresentar decaimentos exponenciais bem elevedos como uma função de $\overline{x}$. Por outro lado, o decaimento exponencial (com o seu c associado) deve ser espacialmente compatível com os menores comprimentos de onda presentes em $u(\overline{x})$ para se obter uma interpolação precisa. Mesmo que os fatores de forma possam ser ajustados por um método de tentativa e erro [19], um procedimento automático é requerido.

Este procedimento pode ser obtido se a Fig. 3.2 é cuidadosamente observada. Esta figura contém plotagens do erro de interpolação percentual ($\mathscr{E}_{\%}$) para um dado sinal $u(\overline{x})$ como uma

função de c para três arranjos espaciais de nós: arranjos regulares , ligeiramente irregulares e irregulares . Como pode-se observar, conforme c se aproxima de zero, a interpolação tende a produzir erros percentuais muito baixos para todos casos. Entretanto, a curva é descontínua em torno de $c = 0$ devido à impossibilidade de inversão da matriz neste intervalo. Além disso, é possível observar que, para cada caso, um segunda raiz C_o existe. A idéia central deste trabalho é usar C_o como um fator de forma para evitar problemas de inversão e, adicionalmente, melhorar a precisão de interpolação. Na prática, como muitas vezes $u(\overline{x})$ não é conhecida analiticamente, a função erro $\mathscr{E}$ não pode ser calculada *a priori*.

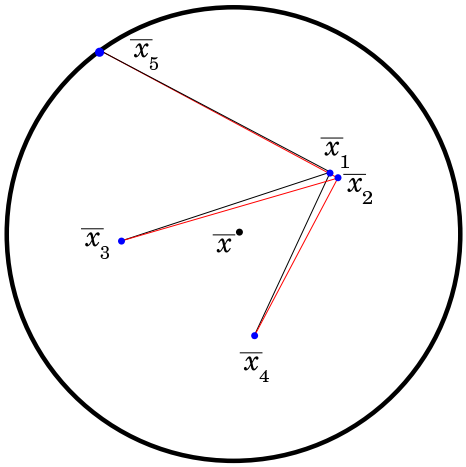


Figura 3.1: Exemplo de domínio de suporte com dois nós posicionados próximos entre si.

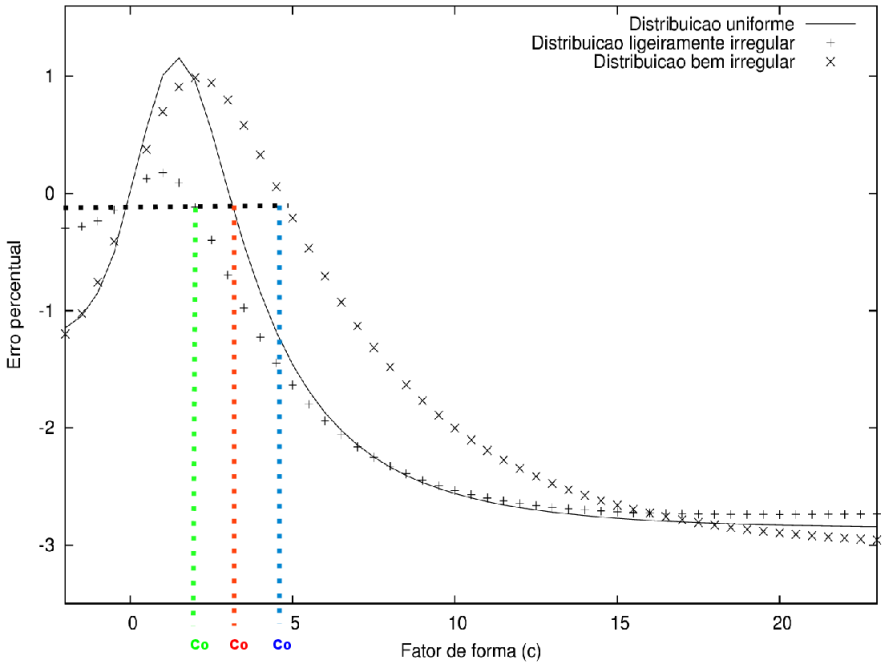


Figura 3.2: $\mathscr{E}_{\%}$ versus c para distribuições de pontos uniforme (Fig. 3.3), ligeiramente irregular (Fig. 3.4) e bem irregular (Fig. 3.5) de $k = 12$ nós de interpolação para $u(\overline{x}) = C(\overline{x})$ e definição gráfica de C_o.

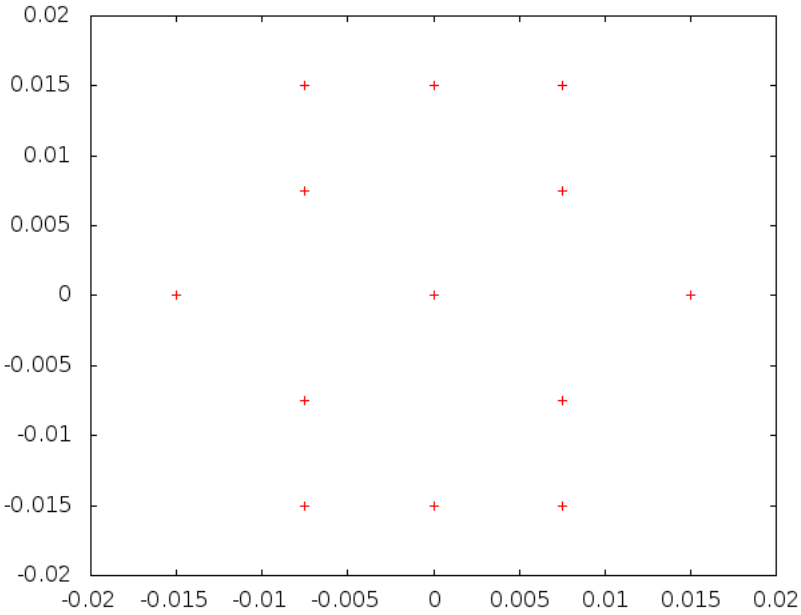


Figura 3.3: Distribuição uniforme de pontos.

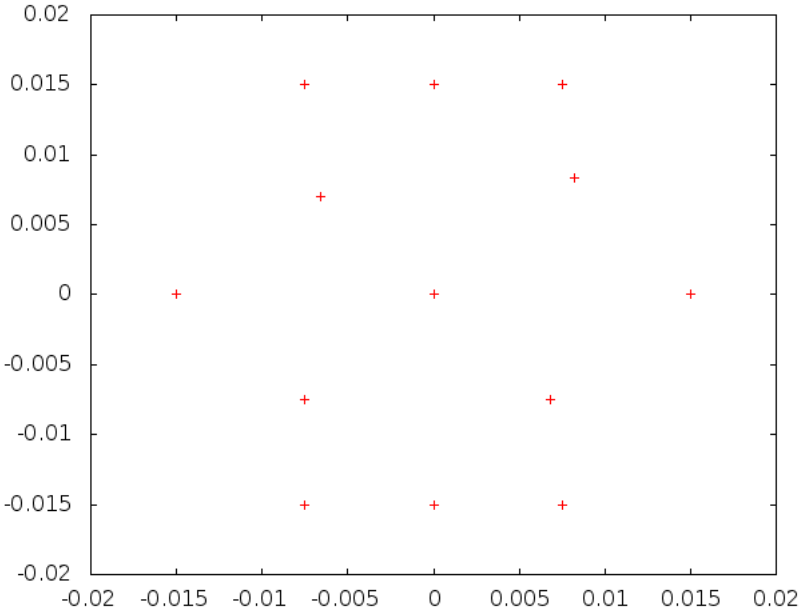


Figura 3.4: Distribuição ligeiramente irregular de pontos.

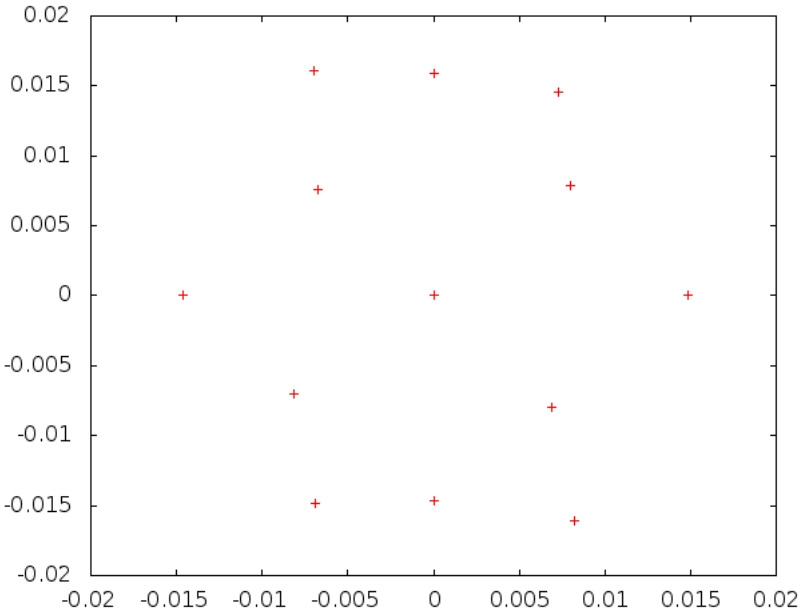


Figura 3.5: Distribuição irregular de pontos.

Baseando-se na discussão anterior, uma hipótese é formulada: dada uma função $u(\bar{x})$ e sua frequência máxima significativa f_{max}, a frequência f_{max} pode ser usada como um parâmetro de referência para determinar C_o para Ω, e, em tais casos, as menores componentes de frequência de $u(\bar{x})$ também podem ser apropriadamente interpoladas em Ω.

Esta hipótese pode ser verificada se, para dado domínio de suporte Ω, uma função de calibração $C(x_i, y_i)$, dada por

$$C(\bar{x}_i) = \cos(Kx_i) + \sin(Ky_i), \tag{3.2}$$

é calculada para todo $\bar{x}_i \in \Omega$. Em (3.2), $\bar{x}_i = (x_i, y_i)$ representa o i-ésimo nó em Ω e K é o número de onda associado, que é expressado por

$$K = \frac{2\pi f_{\max}}{v_0}. \tag{3.3}$$

Em (3.3), v_0 é a velocidade da luz no vácuo. Para se determinar C_o, o erro

$$\mathscr{E}(c) = C_i(c, \bar{x}) - C(\bar{x}) \tag{3.4}$$

é considerado neste trabalho. Em (3.4), $C_i(c, \bar{x})$, que é a versão interpolada de $C(\bar{x})$, é obtida usando-se (3.2) e a formulação do RPIM descrita no capítulo 2. De (3.4), o erro percentual pode ser calculado por $\mathscr{E}\%(c) = 100(C_i(c, \bar{x}) - C(\bar{x}))/C(\bar{x})$. Então, podemos dizer que C_o é a raiz da função erro percentual, quando a função de calibração é considerada, de tal modo que $C_o > 0$.

Neste trabalho, o método *regula falsi* modificado [22] é usado para determinar C_o de (3.4),

de tal forma que

$$C(\bar{x}) - \mathscr{E}_{max} \leq C_i(C_o, \bar{x}) \leq C(\bar{x}) + \mathscr{E}_{max}. \tag{3.5}$$

Aqui, o intervalo de busca considerado para c é $1 \leq c \leq 50$. Tipicamente, três a seis passos do *regula falsi* são necessários para obter um C_o que satisfaz (3.5) com $\mathscr{E}_{max} = 10^{-4}$. O algoritmo *regula falsi* usado neste trabalho é ilustrado pela Fig. 3.6.

Modified_Regula_Falsi($f(x)$,x_i,x_f,$\mathscr{E}_{max}$):
{$f(x)$ é a função cuja raiz será estimada e $[x_i, x_f]$ é o intervalo de busca para a raiz.}
$x \leftarrow a \leftarrow x_i$
$b \leftarrow x_f$
$fa \leftarrow f(a)$
$fb \leftarrow f(b)$
while $|f(x)| > \mathscr{E}_{max}$ **and** $b - a > \mathscr{E}_{max}$ **do**
  $x_0 \leftarrow x$
  $x \leftarrow (a \cdot fb - b \cdot fa)/(fb - fa)$
  if $f(x)f(a) > 0$ **then**
    $a \leftarrow x$
    $fa \leftarrow f(a)$
    if $f(x)f(x_0) > 0$ **then**
      $fb \leftarrow fb/2$
    end if
  else
    $b \leftarrow x$
    $fb \leftarrow f(b)$
    if $f(x)f(x_0) > 0$ **then**
      $fa \leftarrow fa/2$
    end if
  end if
end while
x é a aproximação final para a raiz de f.

Figura 3.6: O algoritmo *regula falsi* modificado.

Em alguns casos, não é possível determinar se C_o existe na faixa de 1 a 50 porque $\mathscr{E}(50) \times \mathscr{E}(1) > 0$. Para esses casos, uma algoritmo de minimização é executado para a função $|\mathscr{E}(c)|$ no intervalo referido. Se C_o não pode ser encontrado no intervalo de busca inicial, um novo intervalo é definido para investigação (por exemplo, $50 \leq c \leq 100$). Finalmente, é de fundamental importância observar que as derivadas espaciais da equação de Maxwell (2.10)-(2.12) são consideradas separadamente para a determinação dos valores assumidos por C_o.

3.3 Considerações Finais

Os resultados dos experimentos realizados (presentes no capítulo 4) confirmam a hipótese deste trabalho: uma função de calibração pode ser usada para calcular C_0, desde que contenha as componentes de mais alta frequência do sinal a ser propagado. A metodologia desenvolvida foi computacionalmente implementada e foi confirmado numericamente que o procedimento é apropriado para a obtenção automática dos fatores de forma locais das funções Gaussianas (para cada domínio de suporte). Permitindo que o RPIM seja mais autonômo e mais preciso para aplicações que envolvam técnicas *multi-scale*, a nova metodologia evita dificuldades envolvendo inversões de matriz associadas à pequenos valores de c.

4 *Experimentos de validação*

Neste capítulo, são apresentados casos de validação dos resultados obtidos pelo simulador implementado. O primeiro é um quadrado metálico com lado λ, cujo campo total calculado pelo RPIM é comparado com o obtido pelo FDTD. E o segundo é um triangulo isósceles metálico, onde o RCS obtido pelo RPIM é comparado com a solução analítica [23]. Por fim, é feito um experimento para validar a implementação das técnicas propostas nesse trabalho.

4.1 O cálculo do *Radar cross section* (RCS) para espalhadores 2-D

A formulação para o cálculo do *Radar cross section* (RCS) utilizada neste trabalho é a desenvolvida pela tese de Bavelis [23], cuja formulação é dada em (4.1)

$$\sigma_{2-D}(\theta) = \lim_{p\to\infty} \frac{2\pi\rho|\vec{E}|^2}{|E^{inc}|^2} = \frac{2\lambda}{\pi} \frac{\left|\sum_{n=-\infty}^{\infty} A_n j^n e^{jn\theta}\right|^2}{\left|\tilde{E}_z^{inc}\right|^2}, \tag{4.1}$$

onde A_n é dado por (4.2).

$$A_n = \frac{\int_0^{2\pi} \tilde{E}_z^{sc}(\rho_a,\phi) e^{-jn\phi} \rho_a d\phi}{2\pi\rho_a H_n^{(2)}(k_0\rho_a)}. \tag{4.2}$$

Em (4.2) e (4.1), $H_n^{(2)}$ é a função Hankel de tipo 2 [24], $\tilde{E}_z^{sc}$ é o campo elétrico espalhado, $\tilde{E}_z^{inc}$ é o campo elétrico incidente, θ é a posição angular e ρ_a é a distância para o centro do objeto.

Tal formulação basea-se na transformação de campos próximos para campos distantes. A integral em (4.2) é calculada em um circunferência que envolve o objeto espalhador em estudo [23].

4.2 Caso 1: O quadrado metálico

Neste caso, o campo elétrico total localizado a $1,5$ m do centro de um quadrado metálico com lado igual a 1 m (λ) foi calculado utilizando-se o FDTD e o RPIM com FLSFCM para ajustes do fator de forma, com o domínio de suporte tendo raio de $\lambda/40$. A Fig. 4.1 ilustra a estrutura simulada. No FDTD foram utilizadas células com mesma largura do raio utilizado nos domínios de suporte do RPIM. A fonte de excitação usada é uma onda plana descrita pelo pulso gaussiano modulado em seno, c om frequência central de $f_c = 300MHz$ dada por . (4.3).

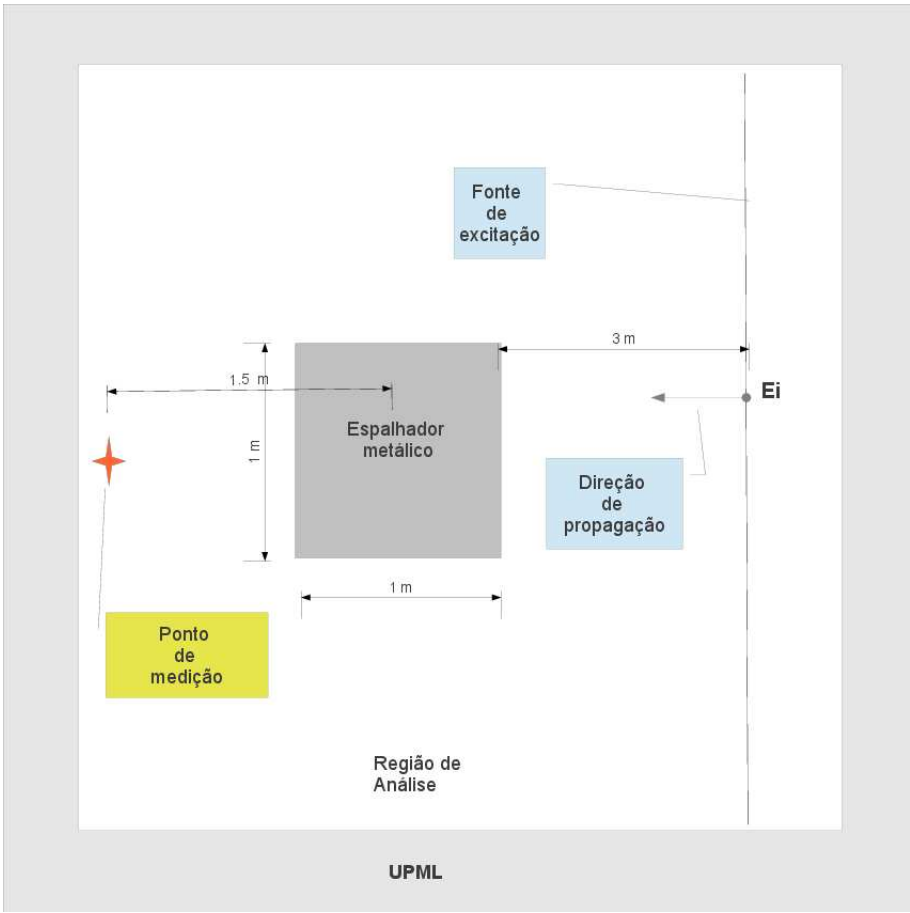


Figura 4.1: Diagrama representando a estrutura simulada, mostrando a posição do transmissor, do ponto de medição e do espalhador.

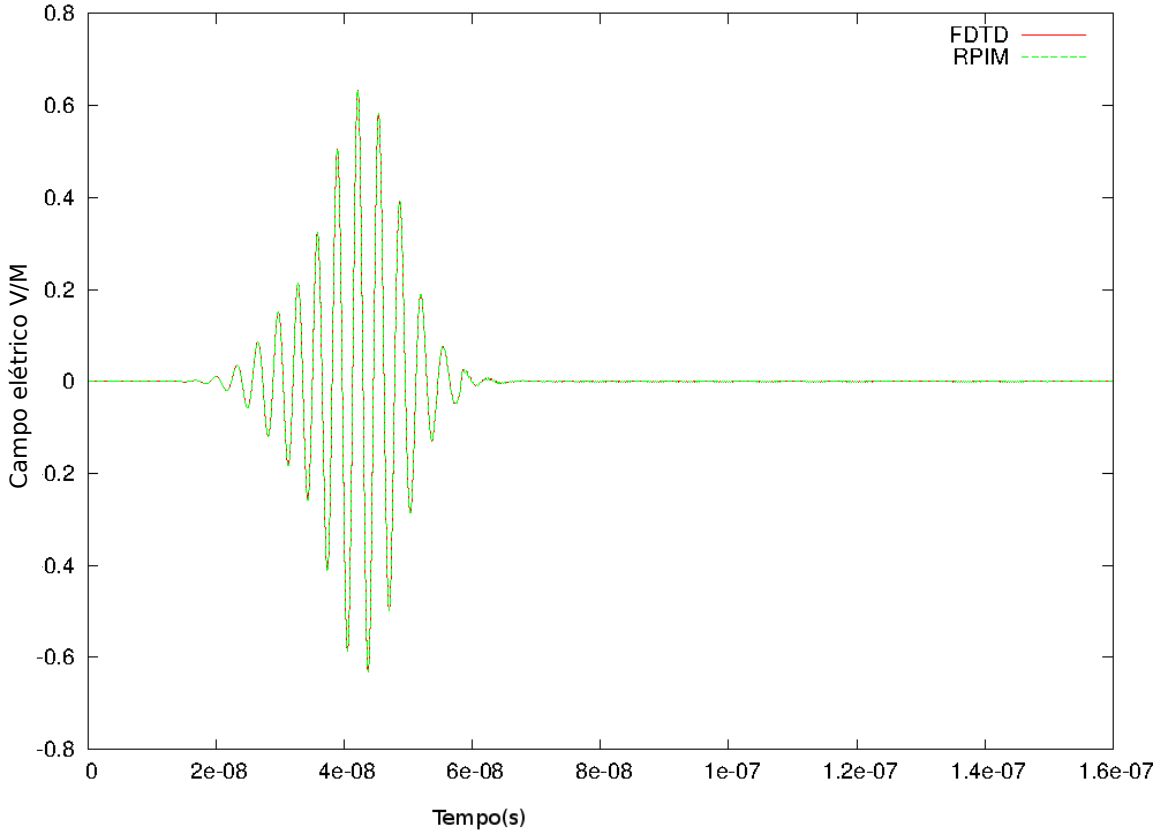


Figura 4.2: Campo total calculado utilizando-se o FDTD e o RPIM.

A fonte é dada por

$$E_z^{inc}(t) = exp\left[-\frac{(q-q_0)^2}{2W^2}\right] sin[2\pi f_c \Delta t(q-q_0+\frac{x}{c\Delta t})], \tag{4.3}$$

onde $q = \frac{t}{\Delta t}$, $q_0 = 700$, c é a velocidade da luz (299792458 m/s), $W = 150$ e $\Delta t = 40$ ps.

A Fig. 4.2 ilustra uma propriedade importante do RPIM: quando os raios dos domínios de suporte são iguais aos lados da célula utilizada no método FDTD, o RPIM gera resultados idênticos aos gerados pelo método FDTD [4].

4.3 Caso 2: O triângulo isósceles

O triângulo metálico isósceles descrito no trabalho de Bavelis [23] foi utilizado para a realização deste caso de validação. A discretização do domínio de análise é apresentada na Fig. 4.3. O espalhador possui base medindo $\lambda\sqrt{2}/2$ e altura medindo $1+\lambda\sqrt{2}/2$, onde $\lambda = 1$ m. A fonte utilizada é o mesmo pulso gaussiano utilizado no Caso 1 dada por (4.3), e é localizada

a 6 m do centro do triângulo. As medições foram feitas na região de "sombra", com pontos localizados a $1,5$ m do baricentro do triângulo, e com inclinação com relação ao eixo x variando de 0 a 30 graus (como ilustrado na Fig. 4.4).

Conforme pode-se observar na Fig. 4.5 o erro obtido pelo RPIM no intervalo de 0 a 30 graus é aceitável. O raio do domínio de suporte nessa simulação foi de $\lambda/40$ ($k \approx 12$).

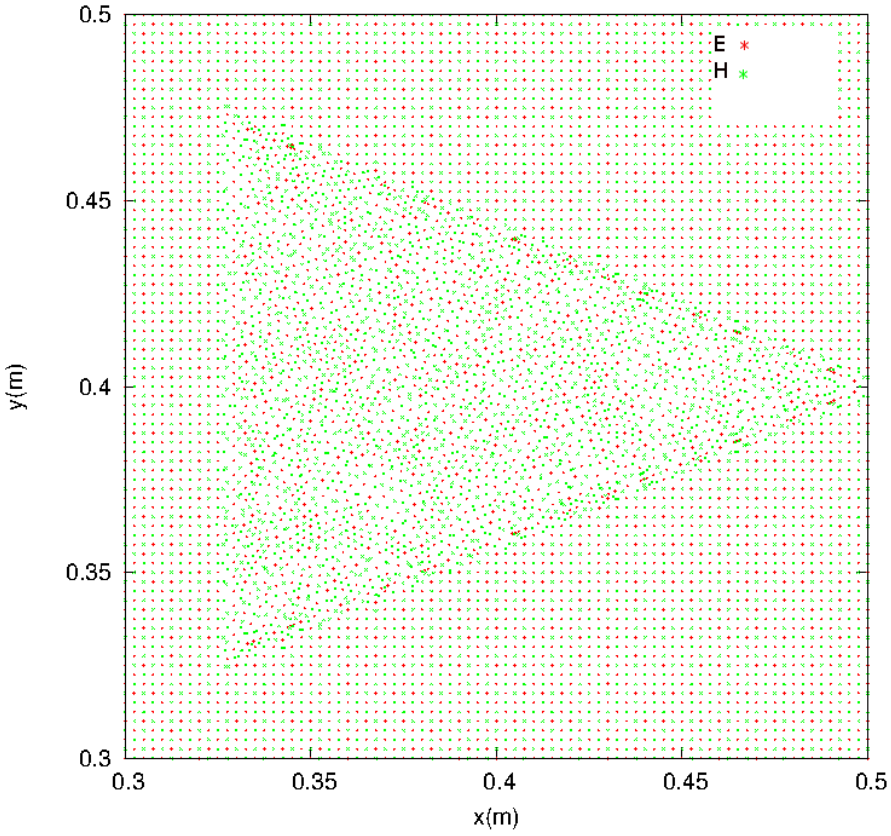


Figura 4.3: Distribuição de pontos utilizada.

O conjunto de pontos utilizados (Fig. 4.3) nessa simulação são os vértices da malha e do diagrama de Voronoi gerados pelo *software Triangles* [25] que gera malhas para o método dos elementos finitos. Os pontos de campo magnético foram escolhidos como sendo os vértices do diagrama de Voronoi [4] e os pontos de campo elétrico foram escolhidos como sendo os vértices dos elementos da malha.

Para calcular a integral de (4.2), foram utilizados pontos distribuídos em um círculo virtual que envolve o triângulo (Fig. 4.4). Em tais pontos, registrou-se o campo Ez transitório. Posteriormente, calculou-se a transformada de Fourier destes sinais para $f = 0.3GHz$ e estes valores foram usados para calcular (4.2) por integração numérica. Foram calculados 201 coeficientes $A_n(-100 \leq n \leq 100)$. Para calcular (4.1), utilizaram-se os coeficientes obtidos em (4.2) e a transformada de Fourier do campo incidente ($f = 0.3GHz$) para se determinar o denominador.

Tais resultados são apresentados na Fig. 4.3. Ótima concordância foi observada com a solução analítica do problema [23]. A Fig. 4.6 mostra a evolução temporal da propagação de campo elétrico.

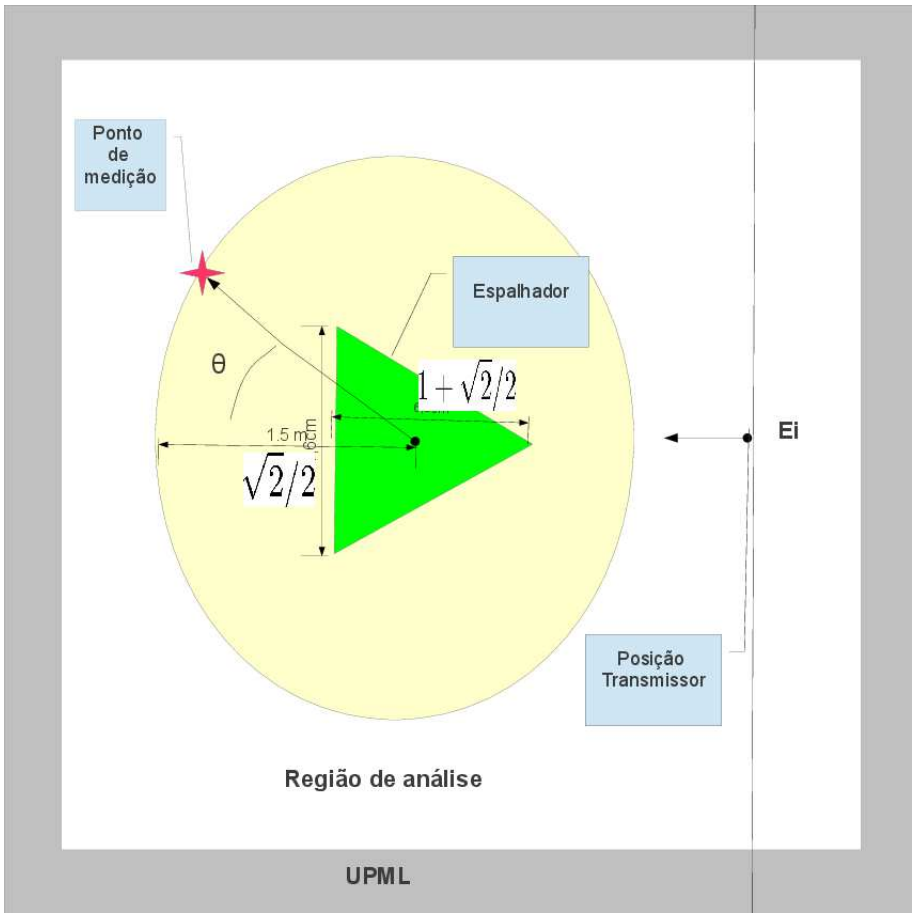


Figura 4.4: Diagrama ilustrando a estrutura utilizada para o cálculo do RCS do triângulo isósceles.

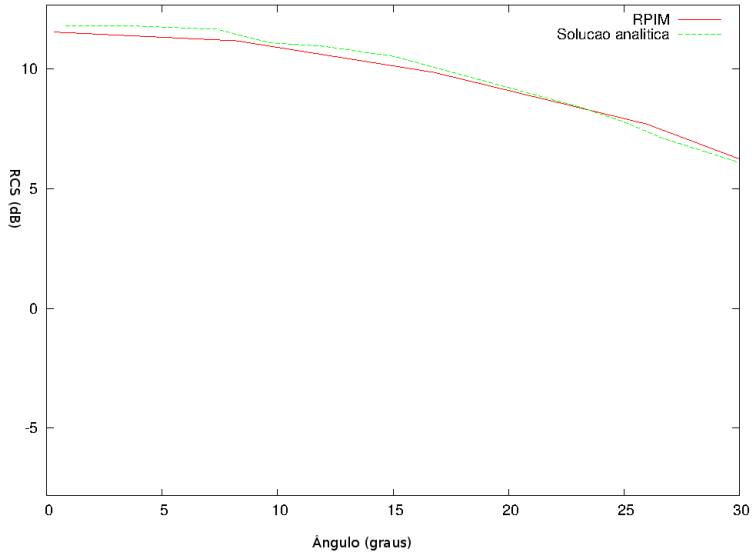
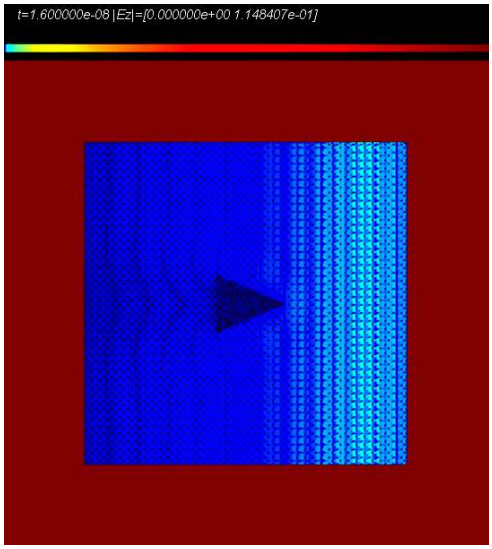
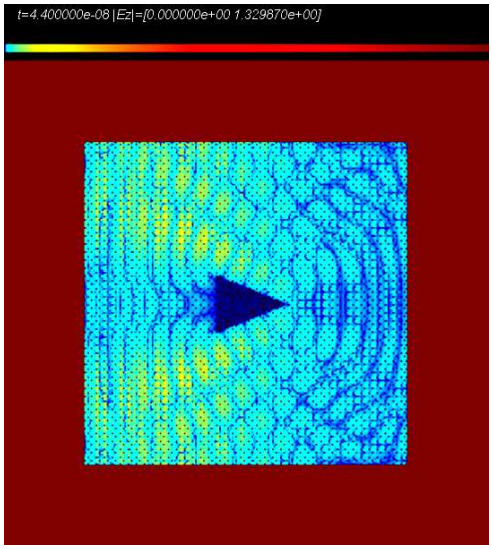


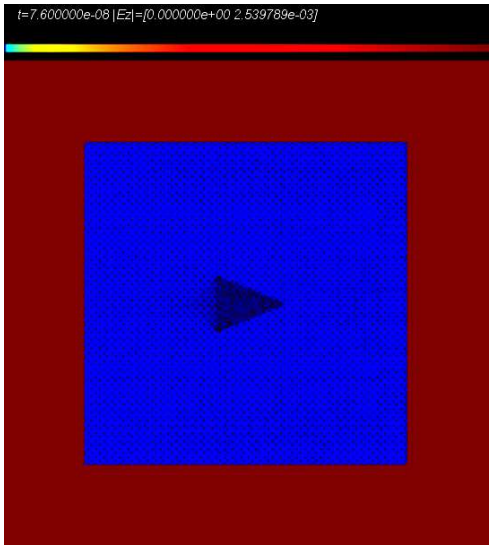
Figura 4.5: Comparação entre o RCS (em dB) calculado pelo RPIM com a solução analítica.



(a) Distribuição espacial de campo elétrico no instante 16 ns



(b) Distribuição espacial de campo elétrico no instante 44 ns



(c) Distribuição espacial de campo elétrico no instante 76 ns

Figura 4.6: Evolução temporal da propagação de campo elétrico.

4.4 Validação do LSFCM e o método FLSFCM

O método sem malha RPIM 2-D foi aplicado para simular o espalhamento eletromagnético de uma onda plana por um cilindro metálico imerso no espaço livre, conforme ilustrado pela Fig. 4.7. Aqui, o modo TMz foi empregado [1]. Os parâmetros do experimento são: diâmetro do cilindro metálico $a = 100$ mm; a onda eletromagnética propaga no vácuo e é excitada por um pulso monociclo banda larga (Fig. 4.8a) com frequência máxima significante $f_{max} = 1,5$ GHz (Fig. 4.8b); as dimensões da região de análise são $3m \times 7m$. A região de análise discreta é parcialmente mostrada pela Fig. 4.7b.

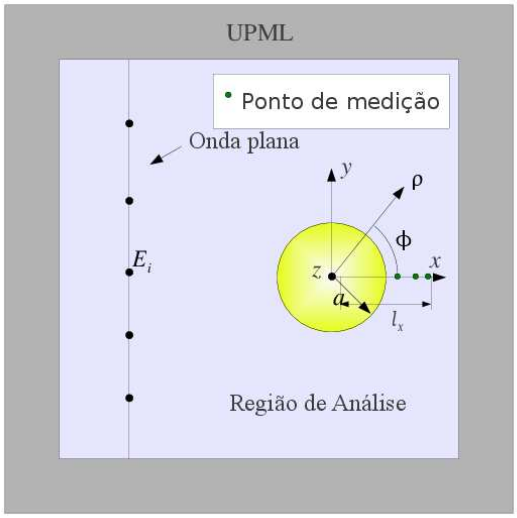
Para os experimentos realizados, inicialmente valores globais de c foram usados para propósitos de teste ($c = 0,1$ e $c = 0,01$) com $k = 12$. O espaçamento médio entre pontos é $\Delta_a = \frac{\lambda}{17}$ (Fig. 4.7b), onde $\lambda = v_0/f_{max}$. Então, c foi localmente calculado (especificamente para cada domínio de suporte) aplicando-se a metodologia apresentada neste trabalho, e uma nova simulação foi executada. Para este caso, os parâmetros Δ_a e k foram mantidos inalterados ($\Delta_a = \frac{\lambda}{17}$ e $k = 12$). A precisão e o critério de estabilidade do algoritmo RPIM seguem [1].

O problema foi também resolvido analiticamente usando a solução apresentada em [24], e dados numéricos adicionais foram gerados utilizando-se o método FDTD. A transformada de Fourier foi aplicada para os sinais transientes para se realizar as comparações com a solução analítica.

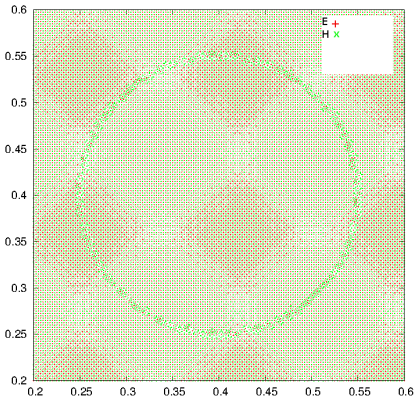
A figura 4.9 mostra a comparação gráfica entre as soluções numéricas e a análitica para o campo elétrico em $\ell_x = 10$ mm, com $\Delta_a = \frac{\lambda}{17}$, para parâmetros de forma global e local. Para o FDTD, devido o efeito *staircase*, foi necessário discretizar o espaço com $\Delta = \frac{\lambda}{80}$ de forma a se obter resultados mais próximos aos gerados com o RPIM ($\Delta_a = \frac{\lambda}{17}$). A Fig. 4.10 mostra resultados similares para $\ell_x = 38$ mm.

Nas Figs. 4.9 e 4.10, é possível ver que o uso de valores locais C_0 produzem curvas mais próximas da solução analítica para a banda inteira de frequências. Quando o método RPIM emprega $c = 0.1$, por exemplo, é possível ver erros $\mathscr{E}_{\%}$ de $7,32\%$ para as componentes de frequência mais altas. Entretanto. Com os parâmetros de forma locais C_0, o algoritmo RPIM produz um erro máximo de $1,24\%$ para $\Delta_a = \frac{\lambda}{10}$.

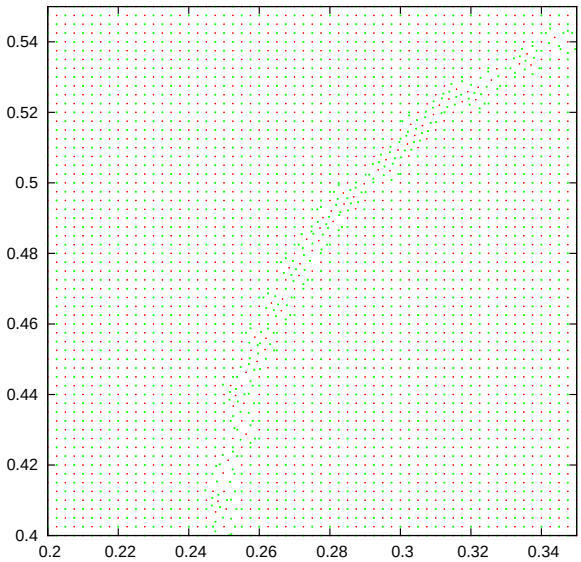
Para a discretização espacial ilustrada na Fig.4.7b, é possível usar $c = 0,1$ para o domínio inteiro, com nenhuma dificuldade para inverter as matrizes. É possível ver a partir das Figs.4.9 e 4.10 que o erro máximo produzido pelo algoritmo RPIM nesta situação é próximo de 2%. Entretanto, é fundamental enfatizar que nem sempre é possível usar tal valor global pequeno para c, especialmente se distribuições altamente heterogênas de pontos são necessárias para



(a) O cilindro espalhador.

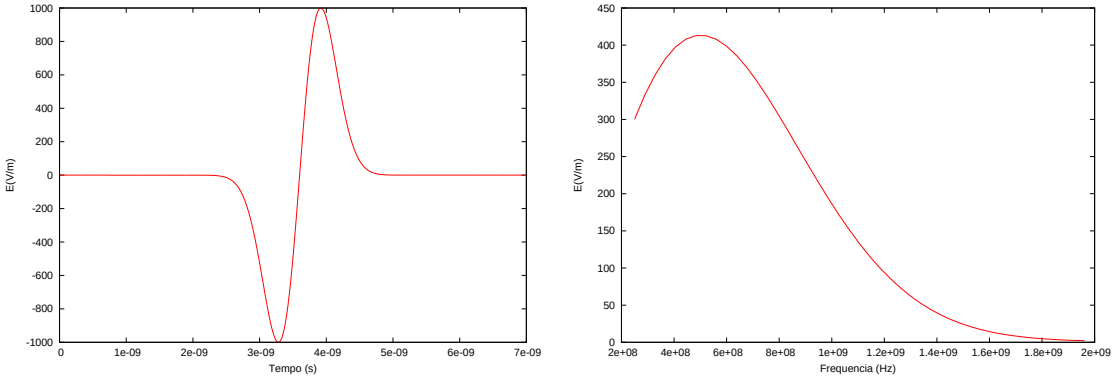


(b) Discretização utilizada.



(c) Ampliação da borda do cilindro.

Figura 4.7: (a) Configuração geométrica do problema e os pontos usados para calcular E_z, (b) parte do conjunto de pontos usados para representar a região de análise ($\vec{E}$ e $\vec{H}$ não são calculados nos mesmo pontos do espaço [1]) e (c) é uma ampliação da borda do cilindro.



(a) Domínio do tempo (b) Espectro de frequência

Figura 4.8: Pulso banda larga usado como fonte de excitação.

representar a região de análise.

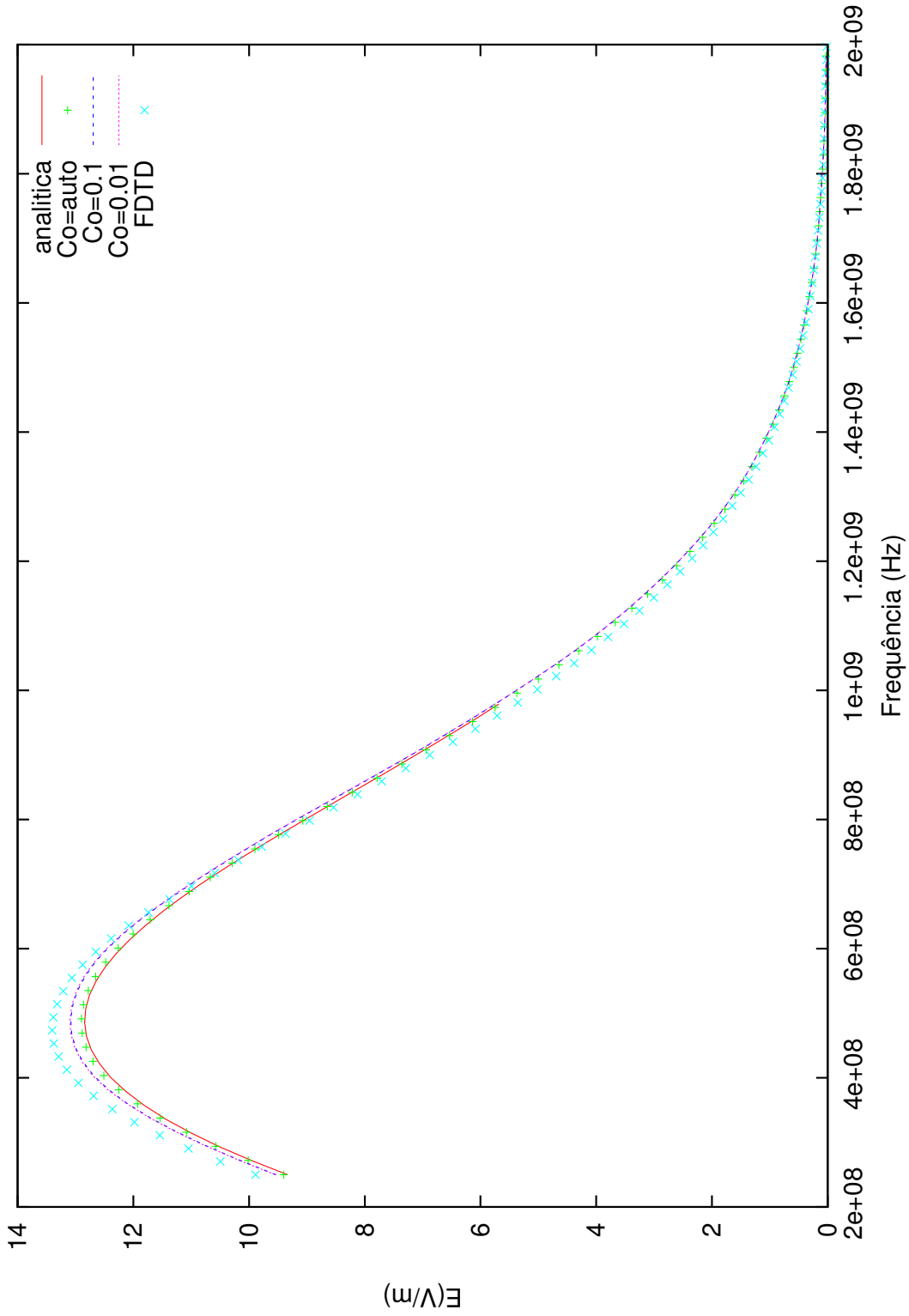


Figura 4.9: Soluções numéricas e analítica para $\vec{E}$ em $\ell_x = 10$ mm: FDTD ($\Delta = \frac{\lambda}{17}$); RPIM ($\Delta_a = \frac{\lambda}{17}$, c local e c global ($c = 0.1$ e $c = 0.01$)).

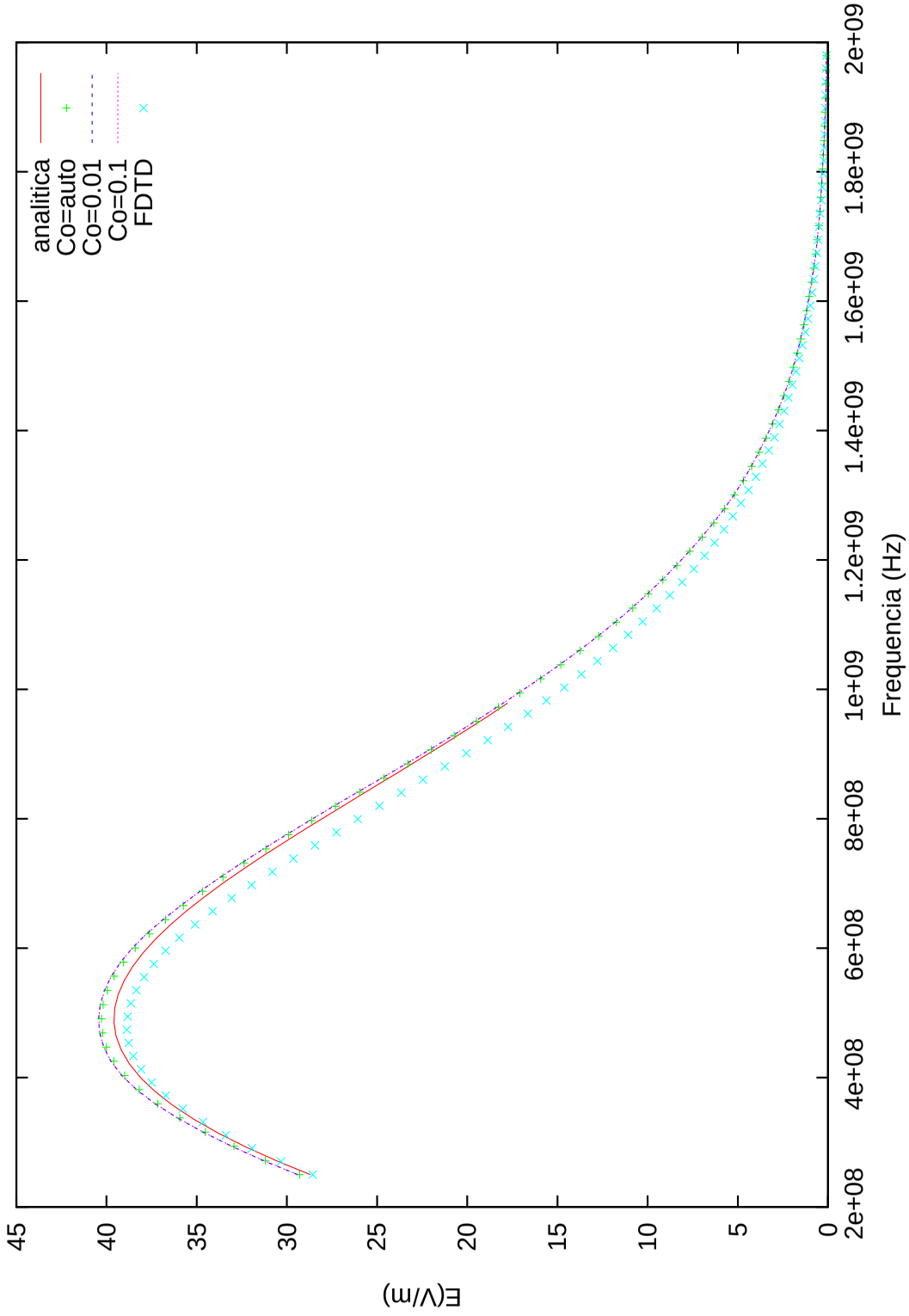


Figura 4.10: Soluções numéricas e analítica para $\vec{E}$ em $\ell_x = 38$ mm: FDTD ($\Delta = \frac{\lambda}{80}$); RPIM ($\Delta_a = \frac{\lambda}{17}$) com c local e c global ($c = 0.1$ e $c = 0.01$)).

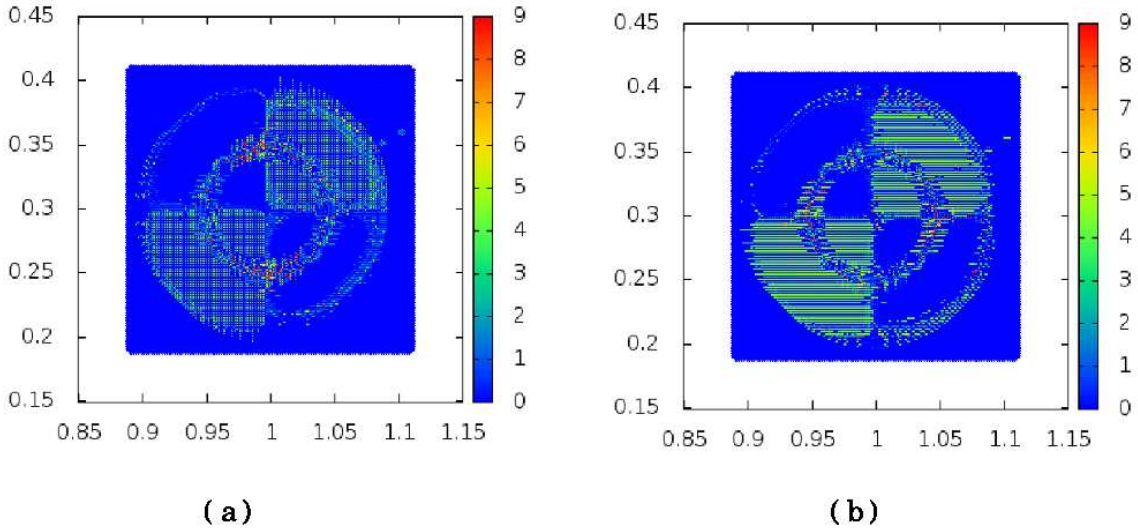


Figura 4.11: Distribuição espacial de C_o para o cálculo de (a) $\partial E_z/\partial x$ e (b) $\partial E_z/\partial y$ para o cilindro metálico.

A Fig. 4.7a define os pontos onde o campo elétrico foi registrado neste trabalho, onde ℓ_x é uma distância medida da arestra direita da superfície do cilindro (paralelamente à x).

Finalmente, a Fig. 4.11 mostra a distribuição espacial de C_0 para o presente problema. Eles foram obtidos a partir do cálculo dos coeficiente de interpolação para $\partial E_z/\partial x$ (Fig.4.11a) e $\partial E_z/\partial y$ (Fig.4.11b) em (2.11) e (2.10), respectivamente. Como é possível ver na Fig. 4.11, a maioria dos valores de C_0 está no intervalo de busca inical para c (de 1 a 5). Entretanto, em alguns casos (a maioria proxímo da borda do cilindro), onde os pontos são colocados mais proximos entre si, valores mais altos para C_0 foram obtidos (pontos vermelhos significam $C_0 \approx 9$, que é o máximo valor assumido por C_0 neste exemplo).

4.5 Um aprimoramento no LSFCM: Fast-LSFCM (FLSFCM)

O LSFCM apresenta bom desempenho quando é utilizado para encontrar o parâmetro de forma ótimo c para se interpolar uma função f. Mas, quando a técnica é utilizada para se otimizar o c para a interpolação de $\frac{\partial f}{\partial v}$, onde v é uma coordenada qualquer, o desempenho da técnica é limitado conforme pode ser observado na Fig. 4.12.

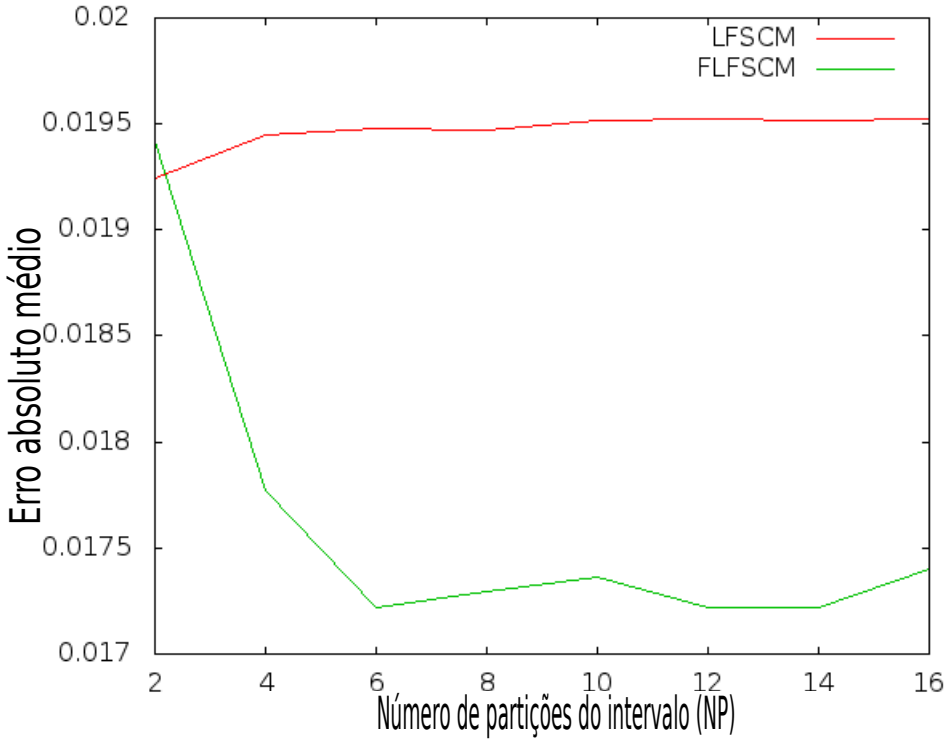


Figura 4.12: Comparação de desempenho entre o LSFCM e o FLSFCM na otimização de c para a interpolação da derivada em x da função $f(x,y) = sin(Kx) + cos(Ky)$, onde K é definido por (3.3).

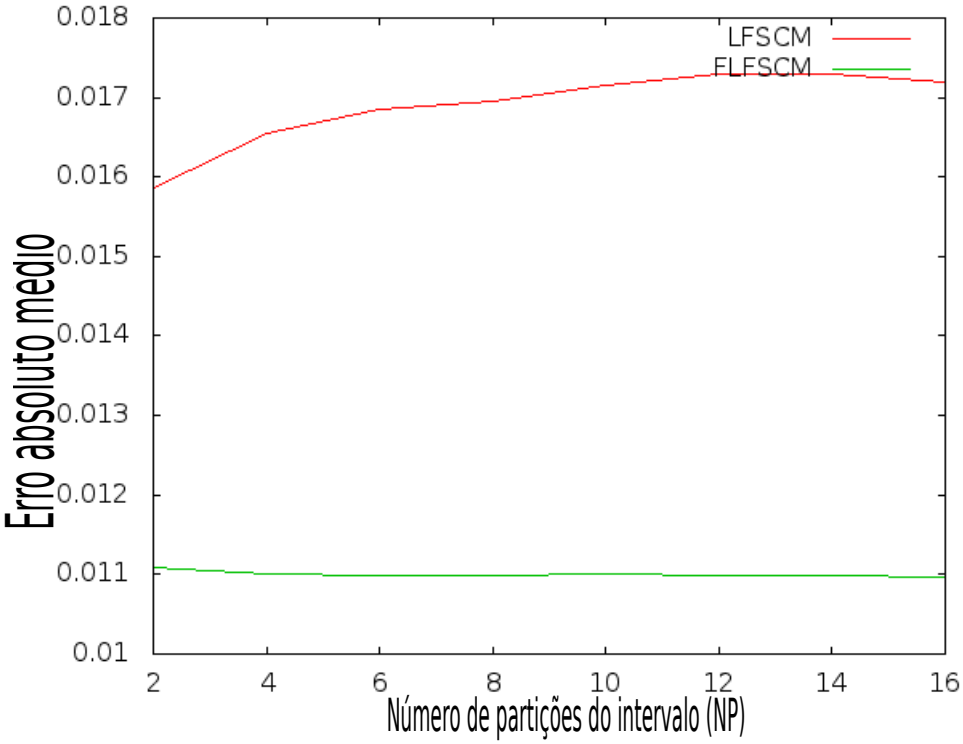


Figura 4.13: Comparação de desempenho entre o LSFCM e o FLSFCM na otimização de c para a interpolação da derivada em y da função $f(x,y) = sin(Kx) + cos(Ky)$, onde K é definido por (3.3).

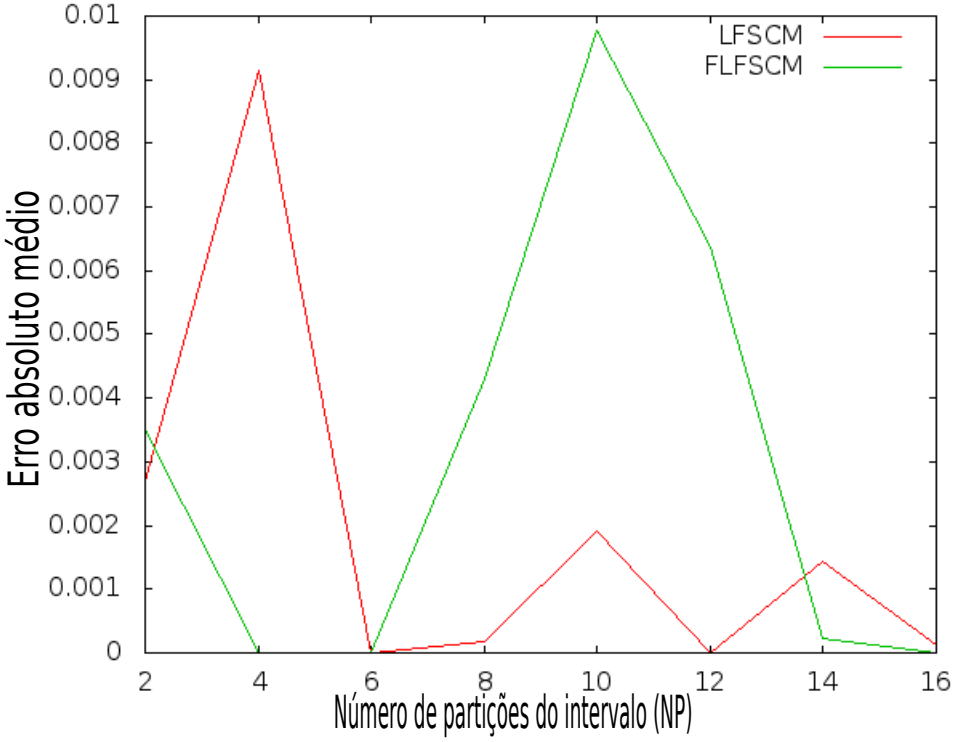


Figura 4.14: Comparação de desempenho entre o LSFCM e o FLSFCM na otimização do c para a interpolação da função $f(x,y) = sin(Kx) + cos(Ky)$, onde K é definido por (3.3).

Na Fig. 4.12, NP é o número de partições em que um intervalo de busca é quebrado para que o *regula falsi* modificado seja aplicado a cada partição para encontrar o c que minimiza o erro (desde que para o subintervalo a relação (4.4) se mantenha).

Uma partição é um subdivisão de um intervalo em intervalos menores para a realização da busca.

$$\mathscr{E}(x_1) \times \mathscr{E}(x_2) < 0, \tag{4.4}$$

onde $\mathscr{E}$ é a função erro utilizada e x_1 e x_2, com $x_1 < x_2$, são os limites de um subintervalo.

Ao se observar a equação da função radial de base utilizada neste trabalho, $f(r) = e^{-c(r/r_{max})^2}$ (onde r é a distância do ponto com relação ao centro do domínio), pode-se concluir que enquanto a busca por c é feita de forma linear, o comportamento da função f depende de um termo quadrático, $(r/r_{max})^2$. Ou seja, em muitas situações, variações lineares no c não têm impactos significativos sobre a função erro $\mathscr{E}$, conforme observa-se no comportamento do LSFCM (Fig. 4.12). Para contornar este problema, o LFSCM pode ser modificado para que a pesquisa se concentre na busca por c^2, permitindo dessa forma que a busca se extenda para um intervalo bem maior, mantendo-se o número de partições da técnica LSFCM. É por ter essa característica que o algoritmo modfiicado é chamado de Método para Calibração Rápida do Fator de Forma Local (*Fast Local Shape Factor Calibration Method* - FLSFCM).

As Figs. 4.12, 4.13 e 4.14 mostram o desempenho médio obtido pelo LSFCM e pelo FLSFCM. Para gerar esses gráficos foram gerados 30 conjuntos contendo deformações aleatórias da distribuição de pontos mostrada na Fig. 3.3. Essas deformações foram obtidas aplicando-se a transformação dada pela Eq. (4.5) a cada ponto do conjunto com excessão do centro. Assim, torna-se

$$(x,y) = (x_i(1+r_1), y_i(1+r_2)), \tag{4.5}$$

onde r_1 e r_2 são variáveis aleatórias com distribuição uniforme definidas no intervalo $[-0.1, 0.1]$ e (x_i, y_i) é o i-ésimo ponto do conjunto mostrado na Fig. 3.3.

Conforme pode se observar na Fig. 4.14, o desempenho do FLSFCM não é melhor que o LSFCM quando aplicado na otimização da interpolação da função, mas quando a técnica é aplicada para otimizar a interpolação de suas derivadas parciais, o desempenho do FLSFCM se evidencia. Mas, o ganho de desempenho e precisão observados nas Figs. 4.12 e 4.13 mostram que essa ligeira modificação - realizar a pesquisa por c^2 e determinar c específicos para cada derivada parcial - gera um importante aprimoramento na técnica, especialmente quando o obje-

tivo é resolver um sistema de equações diferenciais como as equações rotacionais de Maxwell, devido às derivadas espaciais de primeira ordem nelas presentes.

O script utilizado para gerar os dados mostrados nas Figs. 4.12, 4.13 e 4.14 é encontrado no Apêndice B.

4.6 Considerações finais

Neste capítulo, foram realizadas duas simulações que mostraram a correção e precisão do simulador implementado. O Caso 2 foi importante, pois foi possível utilizar pontos gerados por *softwares* disponíveis no mercado. Além disso, nessa simulação verificou-se desempenho satisfatório da técnica, já que com uma discretização relativamente grosseira foi possível obter um erro aceitável na região de sombra.

No caso 1, mostrou-se que o RPIM pode ser equivalente ao FDTD caso o raio do domínio de suporte seja equivalente a tamanho da aresta da grade do FDTD, para problemas retangulares.

E na validação do FLFSCM e do LFSCM, foi mostrada a eficiência da técnica para aumentar a precisão e flexibilidade da técnica.

Com isso, mostra-se a versatilidade e precisão do simulado implementado.

5 *Conclusão*

Neste trabalho apresentaram-se metodologias para contornar problemas de inversão de matrizes que costumam aparecer em certas distribuições de pontos quando se utiliza o RPIM ou outra técnica de simulação sem malha. Em certas situações, dependendo da distribuição dos pontos, a escolha do fator de forma local *c* se torna crucial, já que para algumas faixas de valores o erro da interpolação pode ser muito alto ou nem ser possível realizar a simulação devido aos problemas de inversão.

Um algoritmo eficiente, baseado no *Line Sweep*, para geração de domínios de suporte foi apresentado no capítulo 2. Diferente de técnicas tradicionais, baseadas em *kd-trees* [12], o algoritmo apresentado permite uma pré-computação de todos domínios de suporte em tempo $O(Nklog(N))$, dispensando as buscas de tempo $O(\sqrt{N})$ comuns no *kd-tree*.

O Capítulo 3 apresentou duas metodologias propostas. A primeira (o LSFCM), baseada no *regula falsi*, realiza uma busca pelo fator de forma local *c* que minimiza o erro de interpolação para a derivada de uma função de teste previamente definida. Aprimorando o LFSCM, ao invés de se realizar a busca por *c*, realiza-se a busca por $\beta = \sqrt{c}$, foi possível desenvolver uma metodologia, a FLSFCM, que reduz o tempo de pesquisa. A metodologia mostrou grande eficácia nos testes realizados, permitindo um erro de interpolação menor e, em alguns casos, foi possível até realizar simulações que eram impossíveis utilizando-se fatores de forma globais. Outra proposta apresentada nesse capítulo, foi a utilização de fatores de forma específicos para cada derivada parcial.

Apesar de que a metodologia (LSFCM) desenvolvida neste capítulo tenha sido desenvolvida numericamente, poderia ser melhorada se calculos analíticos de C_0 puderem ser realizados. Isto suprimiria a necessidade de se utilizar algoritmos de busca por raiz, tais como o método *regula falsi* usado (o cálculo do C_0 aumentou o tempo de processamento em aproximadamente 25% para os exemplos numéricos neste trabalho).

Deve-se observar que em muitas aplicações, como as que envolve a equação do calor, a frequência significante máxima não é conhecida imediatamente. Desta forma, mais investiga-

ções são necessárias nesta direção. É importante mencionar que a metodologia proposta pode ser extendida para problemas 3D.

Outro fato importante é que diferentemente de outros algoritmos de ajuste do fator de forma, como o LOOCV [17, 19, 21], o LSFCM consegue assegurar um erro percentual baixo para uma determinada banda de frequência. Isto torna a metodologia um importante aprimoramento para os algoritmos de interpolação sem malha.

Implementações para o RPIM, LSFCM e FLSFCM são encontradas no Apêndice A e no Apêndice B.

Por fim, no capítulo 4, foram apresentados testes que verificam a precisão e correção da técnica. O primeiro teste, um quadrado metálico, demonstra como o RPIM pode ser equivalente ao FDTD quando se ajusta o raio do domínio de suporte como sendo o mesmo da aresta da grade FDTD. O segundo teste, verifica a precisão do método comparando-o com a solução analítica apesentada na literatura [23]. Além dsso, é feito uma validação da implementação do LFSCM e do FLFSCM.

5.1 Trabalhos publicados

O desenvolvimento desse trabalho permitiu a publicação de 1 artigo em periódico:

- Machado, P. L. ; Oliveira, Rodrigo M. S. ; Souza, Washington C. B. ; Araújo, Ramon C. F. ; Tostes, Maria E. L. ; Gonçalves, Cláudio . *An automatic methodology for obtaining optimum shape factors for the radial point interpolation method. Journal of Microwaves, Optoelectronics and Electromagnetic Applications*, v. 10, p. 389-401, 2011.

E permitiu a publicação de 2 artigos em congresso:

- Gonçalves, C. ; Machado, P. L. ; Araújo, R. C. F. A. ; de Oliveira, R.M.S. . *Aplicação do Método RPIM-2D para Modelagem de Espalhamento Eletromagnético por Um Cilindro Metálico no Domínio do Tempo*. In: *Congresso de Métodos Numéricos em Engenharia*, 2011, Coimbra. CMNE, 2011.
- Gonçalves, C. ; Machado, P. L. ; Araújo, R. C. F. A. ; de Oliveira, R.M.S. . *Application of RPIM-2D Method for Modeling the Electromagnetic Scattering by a Dielectric Cylinder in Time-Domain* . In: *International Conference on Applied Simulation and Modelling*, 2011, Crete. IASTED, 2011.

APÊNDICE A – Implementação do RPIM em GNU-Octave

```
function [phi, dphi] = rpim(pts, c)
N = size(pts)(1);
M = size(pts)(2);

PxT(1) = 1;
for n = 1:M
    PxT(n+1) = pts(1, n);
end

rmax = 0;
for n = 2:N
    r = distE(pts(n,:), pts(1,:));
    if r > rmax
        rmax = r;
    end
end

rmax = rmax * rmax;

for n = 2:N
    r = distE(pts(n,:), pts(1,:));
    RxT(n-1) = exp(-c * r * r / rmax);
end


for n = 2:N
```

```
    for m = 2:N
        r = distE(pts(n,:), pts(m,:));
        Ro(n-1, m-1) = exp(-c * r * r / rmax);
    end
end


for n = 2:N
    Po(n-1,1) = 1;
    for m = 1:M
        Po(n-1, m+1) = pts(n, m);
    end
end

Sb = inv(Po' * inv(Ro) * Po) * Po' * inv(Ro);
Sa = (inv(Ro) - inv(Ro) * Po * Sb);

phi = RxT * Sa + PxT * Sb;

for m = 1:M
    for n = 2:N
        dphi(m, n-1) = 0;

        for k = 2:N
            r = distE(pts(k,:),pts(1,:));
            dRxT(k-1) = -2 * c *
            (pts(k, m) - pts(1, m)) * exp(-c * r * r / rmax);
            dphi(m, n-1) = dphi(m, n-1)  * Sa(k-1, n-1);
        end

        dphi(m, n-1) = dphi(m, n-1) + Sb(m+1, n-1);

    end
end
```

```
endfunction
```

Esta função depende de uma métrica, a função *distE*. Uma, que é utilizada neste trabalho, é fornecida a seguir.

```
function r=distE(x, y)
N = size(x)(2);
acc = 0;

for n = 1:N
    acc = acc + (x(n) - y(n)) * (x(n) - y(n));
end

r = sqrt(acc);
endfunction
```

A função *rpim* retorna dois vetores, *phi* e *dphi*, contendo, respectivamente, os coeficientes para interpolar a função e suas derivadas parciais.

APÊNDICE B – Implementação do LSFCM e do FLSFCM em GNU-Octave

```
function r=irpim(F, phi)
r= phi * F';
endfunction

function [c, errmin, ni] = lsfcm(pts, fmax,
                                 cmin, cmax, NP, errmax)

    N = size(pts)(1);
    M = size(pts)(2);

    pi = acos(-1);
    vo = 3e8;
    K = 2 * pi * fmax / vo;
    dbeta = (cmax - cmin) / NP;

    ni(1) = 0;

    for m = 1:M
        ni(m + 1) = 0;
    end

    F = [];
    dFr = [];

    for n = 2:N
        F(n-1) = 0;
```

```
        Fr = 0;

        acc = 0;
        for m = 1:M
            if mod(m,2) == 1
                F(n-1) = F(n-1) + sin(K*pts(n,m));
            else
                F(n-1) = F(n-1) + cos(K*pts(n,m));
                Fr = Fr + 1;
            end
            acc = acc + K*pts(n,m);
        end

        F(n-1) = cos(acc);
    end

    Fr = 1;


    for m = 1:M
        if mod(m,2) == 1
            dFr(m) = K;
        else
            dFr(m) = 0;
        end
         dFr(m) = 0;
    end

%
%Otimizacao de PHI
%
    ok = false;
    bestc = cmin;
    errmin(1) = 100;
```

```
a = 0;
b = 0;
fa = 0;
fb = 0;

fib = false;

for beta=cmin:dbeta:cmax
    xa = beta;
    xb = beta + 0.5 * dbeta;

    [phi, dphi] = rpim(pts, xa);

    Fi = irpim(phi, F);

    dfa = Fi - Fr;

    if !ok || abs(dfa) < errmin(1)
        ok = true;
        errmin(1) = abs(dfa);
        bestc = xa;
    end

    [phi, dphi] = rpim(pts, xb);

    Fi = irpim(phi, F);

    dfb = Fi - Fr;

    if !ok || abs(dfb) < errmin(1)
        ok = true;
        errmin(1) = abs(dfb);
        bestc = xb;
    end
```

```
    if dfa * dfb < 0
        fa = dfa;
        fb = dfb;
        a = xa;
        b = xb;
        fib = true;
    end

    ni(1) = ni(1) + 1;
end

k = 0;
fx = fa;
fib;

while k < NP && errmin(1) > errmax
    x = (a*fb - b*fa)/(fb-fa);

    [phi, dphi] = rpim(pts, x);

    Fi = irpim(phi, F);
    fxo = fx;
    fx = Fi - Fr;

    if  fx*fa > 0
        a =  x;
        fa = fx;
        if fx*fxo > 0
            fb = 0.5*fb;
        end
    else
        b = x;
        fb = fx;
        if fx*fxo > 0
            fa = 0.5*fa;
```

```
            end
        end

        if abs(fx) < errmin(1)
            errmin(1) = abs(fx);
            bestc = x;
        end

        ni(1) = ni(1) + 1;
        k = k + 1;
    end

    c(1) = bestc;

%
%Otimizacao de DPHI
%
    for m=1:M
        ok = false;
        bestc = cmin;
        errmin(m+1) = 100;

        a = 0;
        b = 0;
        fa = 0;
        fb = 0;
        ni(m+1) = 0;
        fib = false;
        for beta=cmin:dbeta:cmax
            xa = beta;
            xb = beta + 0.5 * dbeta;

            [phi, dphi] = rpim(pts, xa);

            Fi = irpim(dphi(m,:), F);
```

```
        dfa = Fi - dFr(m);

        if !ok || abs(dfa) < errmin(m+1)
            ok = true;
            errmin(m+1) = abs(dfa);
            bestc = xa;
        end

        [phi, dphi] = rpim(pts, xb);

        Fi = irpim(dphi(m,:), F);

        dfb = Fi - dFr(m);

        if !ok || abs(dfb) < errmin(m+1)
            ok = true;
            errmin(m+1) = abs(dfb);
            bestc = xb;
        end

        if dfa * dfb < 0
            fa = dfa;
            fb = dfb;
            a = xa;
            b = xb;
            fib = true;
        end

        ni(m+1) = ni(m+1) + 1;
    end

    k = 0;
    fx = fa;
    fib;
```

```
while k < NP && errmin(m+1) > errmax
    x = (a*fb - b*fa)/(fb-fa);

    [phi, dphi] = rpim(pts, x);

    Fi = irpim(dphi(m,:), F);
    fxo = fx;
    fx = Fi - dFr(m);

    if  fx*fa > 0
        a =  x;
        fa = fx;
        if fx*fxo > 0
            fb = 0.5*fb;
        end
    else
        b = x;
        fb = fx;
        if fx*fxo > 0
            fa = 0.5*fa;
        end
    end

    if abs(fx) < errmin(m+1)
        errmin(m+1) = abs(fx);
        bestc = x;
    end

    ni(m+1) = ni(m+1) + 1;
    k = k + 1;
end

c(m+1) = bestc;
```

```
    end
end



function [c, errmin, ni] = flsfcm(pts, fmax,
                                  cmin, cmax, NP, errmax)

    N = size(pts)(1);
    M = size(pts)(2);

    pi = acos(-1);
    vo = 3e8;
    K = 2 * pi * fmax / vo;
    dbeta = (cmax - cmin) / NP;

    ni(1) = 0;

    for m = 1:M
        ni(m + 1) = 0;
    end

    F = [];
    dFr = [];

    for n = 2:N
        F(n-1) = 0;
        Fr = 0;

        acc = 0;
        for m = 1:M
            if mod(m,2) == 1
                F(n-1) = F(n-1) + sin(K*pts(n,m));
            else
                F(n-1) = F(n-1) + cos(K*pts(n,m));
```

```
                Fr = Fr + 1;
            end
            acc = acc + K*pts(n,m);
        end

        F(n-1) = cos(acc);
    end

    Fr = 1;


    for m = 1:M
        if mod(m,2) == 1
            dFr(m) = K;
        else
            dFr(m) = 0;
        end
         dFr(m) = 0;
    end

%
%Otimizacao de PHI
%
    ok = false;
    bestc = cmin * cmin;
    errmin(1) = 100;

    a = 0;
    b = 0;
    fa = 0;
    fb = 0;

    fib = false;

    for beta=cmin:dbeta:cmax
```

```
xa = beta;
xb = beta + 0.5 * dbeta;

[phi, dphi] = rpim(pts, xa * xa);

Fi = irpim(phi, F);

dfa = Fi - Fr;

if !ok || abs(dfa) < errmin(1)
    ok = true;
    errmin(1) = abs(dfa);
    bestc = xa * xa;
end

[phi, dphi] = rpim(pts, xb * xb);

Fi = irpim(phi, F);

dfb = Fi - Fr;

if !ok || abs(dfb) < errmin(1)
    ok = true;
    errmin(1) = abs(dfb);
    bestc = xb * xb;
end

if dfa * dfb < 0
    fa = dfa;
    fb = dfb;
    a = xa;
    b = xb;
    fib = true;
end
```

```
    ni(1) = ni(1) + 1;
end

k = 0;
fx = fa;
fib;

while k < NP && errmin(1) > errmax
    x = (a*fb - b*fa)/(fb-fa);

    [phi, dphi] = rpim(pts, x * x);

    Fi = irpim(phi, F);
    fxo = fx;
    fx = Fi - Fr;

    if  fx*fa > 0
        a =  x;
        fa = fx;
        if fx*fxo > 0
            fb = 0.5*fb;
        end
    else
        b = x;
        fb = fx;
        if fx*fxo > 0
            fa = 0.5*fa;
        end
    end

    if abs(fx) < errmin(1)
        errmin(1) = abs(fx);
        bestc = x * x;
    end
```

```
        ni(1) = ni(1) + 1;
        k = k + 1;
    end

    c(1) = bestc;

%
%Otimizacao de DPHI
%
    for m=1:M
        ok = false;
        bestc = cmin;
        errmin(m+1) = 100;

        a = 0;
        b = 0;
        fa = 0;
        fb = 0;
        ni(m+1) = 0;
        fib = false;
        for beta=cmin:dbeta:cmax
            xa = beta;
            xb = beta + 0.5 * dbeta;

            [phi, dphi] = rpim(pts, xa * xa);

            Fi = irpim(dphi(m,:), F);

            dfa = Fi - dFr(m);

            if !ok || abs(dfa) < errmin(m+1)
                ok = true;
                errmin(m+1) = abs(dfa);
                bestc = xa * xa;
            end
```

```
        [phi, dphi] = rpim(pts, xb * xb);

        Fi = irpim(dphi(m,:), F);

        dfb = Fi - dFr(m);

        if !ok || abs(dfb) < errmin(m+1)
            ok = true;
            errmin(m+1) = abs(dfb);
            bestc = xb * xb;
        end

        if dfa * dfb < 0
            fa = dfa;
            fb = dfb;
            a = xa;
            b = xb;
            fib = true;
        end

        ni(m+1) = ni(m+1) + 1;
    end

    k = 0;
    fx = fa;
    fib;

    while k < NP && errmin(m+1) > errmax
        x = (a*fb - b*fa)/(fb-fa);

        [phi, dphi] = rpim(pts, x * x);

        Fi = irpim(dphi(m,:), F);
        fxo = fx;
```

```
            fx = Fi - dFr(m);

            if  fx*fa > 0
                a =  x;
                fa = fx;
                if fx*fxo > 0
                    fb = 0.5*fb;
                end
            else
                b = x;
                fb = fx;
                if fx*fxo > 0
                    fa = 0.5*fa;
                end
            end

            if abs(fx) < errmin(m+1)
                errmin(m+1) = abs(fx);
                bestc = x * x;
            end

            ni(m+1) = ni(m+1) + 1;
            k = k + 1;
        end

        c(m+1) = bestc;

    end
end
```

A figuras mostradas na seção 4.5 do capítulo 3, foram geradas utilizando o script a seguir.

```
function test_rpim(tipo)
    fmax = 1e9;
    lmb = 3e8 / fmax;
    dx = dy = lmb / 20;
```

```
if tipo == 1
    f = fopen("pontos_igual.dat");
    fp = fopen("dist_1.dat","w+");
end

if tipo == 2
    f = fopen("pontos_igual.dat");
    fp = fopen("dist_2.dat","w+");
end

if tipo == 3
    f = fopen("pontos_igual.dat");
    fp = fopen("dist_3.dat","w+");
end

m = 1;
NR = 3;

while (!feof(f))
    [px, py, count] = fscanf(f,"%lf%lf", '2');

    if count == 0
        break;
    end

    xo(m) = x(m) = px * dx;
    yo(m) = y(m) = py * dy;


    if tipo == 3 && m > 1
        x(m) = (1 + 0.2 * rand() - 0.1) * x(m);
        y(m) = (1 + 0.2 * rand() - 0.1) * y(m);
    end

    if tipo == 2 && m > 1  && NR > 0
```

```
            x(m) = (1 + 0.2 * rand() - 0.1) * x(m);
            y(m) = (1 + 0.2 * rand() - 0.1) * y(m);
            NR = NR - 1;
        end


        fprintf(fp,"%e %e\n", x(m), y(m));

        m = m+1;

    end

    fclose(fp);
    fclose(f);


    pts = [x' y'];

    N = size(pts)(1);
    M = size(pts)(2);
    pi = acos(-1);

    K = 2 * pi * fmax / 3e8;


    Fr = 1;


    dFr(1) = K;
    dFr(2) = 0;



    errmax = 1e-10;
    NP = 24;
```

```
cmin = 1;
cmax = 101;

if tipo == 3
    M = 7;
else
    M = 1;
end




if tipo == 1
    f11 = fopen("err_1lsfcm_1_1.dat","w+");
    f12 = fopen("err_1lsfcm_2_1.dat","w+");

    f21 = fopen("err_2lsfcm_1_1.dat","w+");
    f22 = fopen("err_2lsfcm_2_1.dat","w+");

    f31 = fopen("err_3lsfcm_1_1.dat","w+");
    f32 = fopen("err_3lsfcm_2_1.dat","w+");
end

if tipo == 2
    f11 = fopen("err_1lsfcm_1_2.dat","w+");
    f12 = fopen("err_1lsfcm_2_2.dat","w+");

    f21 = fopen("err_2lsfcm_1_2.dat","w+");
    f22 = fopen("err_2lsfcm_2_2.dat","w+");

    f31 = fopen("err_3lsfcm_1_2.dat","w+");
    f32 = fopen("err_3lsfcm_2_2.dat","w+");
end
```

```
if tipo == 3
    f11 = fopen("err_1lsfcm_1_3.dat","w+");
    f12 = fopen("err_1lsfcm_2_3.dat","w+");

    f21 = fopen("err_2lsfcm_1_3.dat","w+");
    f22 = fopen("err_2lsfcm_2_3.dat","w+");

    f31 = fopen("err_3lsfcm_1_3.dat","w+");
    f32 = fopen("err_3lsfcm_2_3.dat","w+");
end

printf("lsfcm\n");

for NP = 2:2:16
    errm(1, NP) = 0;
    errm2(1, NP) = 0;

    errm(2, NP) = 0;
    errm2(2, NP) = 0;

    errm(3, NP) = 0;
    errm2(3, NP) = 0;
end

for NI = 1:M

    if tipo == 3
        for m = 2:N
            x(m) = xo(m)+(0.02 * rand() - 0.01) * dx;
            y(m) = yo(m)+(0.02 * rand() - 0.01) * dy;
        end
    end
```

```
pts = [x' y'];

F = [];
dF = [];

for n = 2:N
    F(n-1) = 0;
    F(n-1) = F(n-1) + sin(K*pts(n,1)) +
        cos(K*pts(n, 2));
end

for  NP = 2:2:16

    printf("NP = %d\n", NP);

    cmin = 1;
    cmax = 10;
    [c, err, ni] =
        lsfcm(pts, fmax, cmin, cmax, NP, errmax);


    errm(1, NP) = errm(1, NP) + abs(err(1));
    errm(2, NP) = errm(2, NP) + abs(err(2));
    errm(3, NP) = errm(3, NP) + abs(err(3));

    cmin = 1;
    cmax = 10;
    [c, err, ni] =
        flsfcm(pts, fmax, cmin, cmax, NP, errmax);


    errm2(1, NP) = errm2(1, NP) + abs(err(1));
    errm2(2, NP) = errm2(2, NP) + abs(err(2));
    errm2(3, NP) = errm2(3, NP) + abs(err(3));
```

```
        end

         printf("M = %d\n", M);

    end

    for NP = 2:2:16
        fprintf(f11, "%d %e\n", NP, errm(1, NP)/M);
        fprintf(f12, "%d %e\n", NP, errm2(1, NP)/M);

        fprintf(f21, "%d %e\n", NP, errm(2, NP)/M);
        fprintf(f22, "%d %e\n", NP, errm2(2, NP)/M);

        fprintf(f31, "%d %e\n", NP, errm(3, NP)/M);
        fprintf(f32, "%d %e\n", NP, errm2(3, NP)/M);
    end

    fclose(f11); fclose(f12);
    fclose(f21); fclose(f22);
    fclose(f31); fclose(f32);
end
```

Referências Bibliográficas

[1] KAUFMANN, T.; FUMEAUX, C.; VAHLDIECK, R. The meshless radial point interpolation method for time-domain electromagnetics. In: *IEEE MTT-S International Microwave Symposium*. [S.l.: s.n.], 2008. p. 61–65.

[2] WANG, J. G.; LIU, G. R. A point interpolation meshless method based on radial basis functions. *Int. J. Numer. Method*, v. 54, p. 1623–1648, 2002.

[3] LAI, S. J.; WANG, B. Z.; DUAN, Y. Meshless radial basis function method for transient electromagnetic computations. *IEEE Trans. Magn.*, v. 44, p. 2288–2295, 2008.

[4] YU, Y.; CHEN, Z. D. A 3-d radial point interpolation method for meshless time-domain modeling. *IEEE Transactions on microwave theory and techniques*, v. 57, n. 8, p. 2015–2020, 2009.

[5] TAFLOVE, A.; HAGNESS, S. C. *Computational Electrodynamics, The Finite-Difference Time-Domain Method*. 3. ed. [S.l.]: Artech House Inc., 2005.

[6] GUIBAS, L. J.; STOLFI, J. On computing all north-east nearest neighbors in the l1 metric. *Information Processing Letters*, v. 17, p. 219–223, 1983.

[7] YEE, K. Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media. *IEEE Trans. Antennas and Propagation*, v. 14, p. 302–307, 1966.

[8] CINGOSKI, V.; MIYAMOTO, N.; YAMASHITA, H. Element-free galerkin method for electromagnetic field computations. *IEEE Trans. Magn.*, v. 34, p. 3236–3239, 1998.

[9] VIANA, S. A.; MESQUITA, R. C. Moving least square reproducing kernel method for electromagnetic field computation. *IEEE Trans. Magn.*, v. 35, n. 3, p. 1372–1375, 1999.

[10] ALA, G. et al. Smoothed particle electromagnetics: A mesh-free solver for transients. *J. Comput. Appl. Math.*, v. 191, n. 2, p. 194–205, 2006.

[11] J.L.BENTLEY. Multidimensional binary search trees used for associative searching. *Commun.ACM*, v. 18, n. 9, p. 509–517, 1975.

[12] PARREIRA, G. F. et al. Efficient algorithms and data structures for element-free galerkin method. *IEEE Transactions on Magnetics*, v. 42, n. 4, p. 659–662, 2006.

[13] LIMA, E. L. *Espaços Métricos*. [S.l.]: IMPA-Projeto Euclides, 1993.

[14] HöNIG, C. S. *Aplicações de Topologia à Analise*. [S.l.]: IMPA-Projeto Euclides, 1976.

[15] STROUSTRUP, B. *The C++ Programming Language*. [S.l.]: Addison-Wesley Professional, 2000.

[16] PREPARATA, F. P.; SHAMOS, M. I. *Computational Geometry:An Introduction*. [S.l.]: Springer-Verlag, 1985.

[17] FASSHAUER, G. E.; ZHANG, J. G. On choosing optimal shape parameters for rbf approximation. *Numerical Algorithms*, v. 45, n. 1-4, p. 345–368, 2007.

[18] GEDNEY, S. D. An anisotropic perfectly matched layer absorbing media for the truncation of fdtd latices. *IEEE Trans. Antennas and Propagation*, v. 44, p. 1630–1639, 1996.

[19] FASSHAUER, G. E. *Meshfree Approximation Methods with MatLab*. [S.l.]: World Scientific Publishing Company, 2007.

[20] KAUFMANN, T. *The meshless radial point interpolation method for electromagnetics*. Tese (Doctor of Sciences) — ETH ZURICH, 2011.

[21] KAUFMANN, T.; ENGSTROM, C.; FUMEAUX, C. Residual-based adaptive refinement for meshless eigenvalue solvers. In: *Electromagnetics in Advanced Applications (ICEAA)*. [S.l.: s.n.], 2010. p. 244–247.

[22] CHAPRA, S. C.; CANALE, R. P. *Numerical Methods for Engineers*. 6. ed. [S.l.]: McGraw-Hill, 2010.

[23] BAVELIS, K. *Finite-Element Time-Domain Modelling of Cylindrical Structures with a Modal Non-Reflecting Boundary Condition*. Tese (Doctor of Philosophy in Engineering) — University of Warwick, 2010.

[24] BALANIS, C. A. *Advanced Engineering Eletromagnetics*. [S.l.]: John Wiley and Sons, New York, 1989.

[25] SHEWCHUK, J. R. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. *Applied Computational Geometry: Towards Geometric Engineering*, v. 1148, p. 203–222, 1996.