Universidade Estadual de Campinas
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE MICROONDAS E ÓPTICA
UNICAMP- FEEC-DMO, CAMPINAS –SP, C.P. 6101, CEP 13083-070

Resolução de sistemas de equações lineares provenientes da simulação de estruturas fotônicas

DISSERTAÇÃO DE MESTRADO

AUTOR: Kleucio Claudio
ORIENTADOR: Prof. Dr. Hugo Enrique Hernández Figueroa
CO-ORIENTADORA: Profa. Dra. Marli de Freitas Gomes Hernández

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos necessários para a obtenção do título de Mestre em Engenharia Elétrica

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Kleucio Claudio
C571r Resolução de sistemas de equações lineares provenientes da simulação de estruturas fotônicas / Kleucio Claudio.-- Campinas, SP: [s.n.], 2003.

Orientadores: Hugo Enrique Hernández Figueroa
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Sistemas lineares. 2. Métodos iterativos (Matemática). 3. Método dos elementos finitos. I. Hernández Figueroa, Hugo Enrique. II. Hernández, Marli de Freitas Gomes. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Universidade Estadual de Campinas
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE MICROONDAS E ÓPTICA
UNICAMP- FEEC-DMO, CAMPINAS –SP, C.P. 6101, CEP 13083-070

Resolução de sistemas de equações lineares provenientes da simulação de estruturas fotônicas

DISSERTAÇÃO DE MESTRADO

Autor: Kleucio Claudio
Orientador: Prof. Dr. Hugo Enrique Hernández Figueroa
Co-Orientadora: Profa. Dra. Marli de Freitas Gomes Hernández

BANCA EXAMINADORA

Dr. Hugo E. Hernández Figueroa - presidente
FEEC/ UNICAMP

Dr. Francisco José Arnold
CESET/ UNICAMP

Dr. Cosme Rúbio Mercedes
FEEC/ UNICAMP

Dr. Rui Fragassi Souza
FEEC/ UNICAMP

AGOSTO - 2003

Resumo

Implementou-se uma subrotina (***LIRIOS***) para resolução de sistemas lineares através do método iterativo gradiente biconjugado estabilizado precondicionado, para simulação de dispositivos fotônicos, através do método dos elementos finitos. Ela foi implementada aterando-se subrotina de precondicionamento ILUT, do pacote SPARSKIT, de variáveis reais para complexas. Aplicou-se a LIRIOS à propagação de pulsos e feixes em guias com formulações vetorial e escalar. Os dispositivos utilizados foram o acoplador por fibras ópticas, o guia canal, e a junção em ângulo reto com estruturas de cristais fotônicos. Os resultados obtidos demonstram que o método iterativo implementado é uma alternativa muito eficiente em resoluções de problemas deste tipo, sobressaindo-se em relação à subrotina ME28, que é baseada na eliminação de Gauss. Solucionou-se sistemas de dimensão da ordem de 250.000, enquanto a ME28 está limitada a ordem de 75.000 variáveis. A LIRIOS foi desenvolvida em linguagem Fortran 77, no sistema operacional Linux RED HAT.

"Abstract"

Subroutine (***LIRIOS***) to solve linear systems based on iterative methods was implemented for the Finite Element simulation of photonic devices. It was implemented with the Preconditioned Biconjugate Gradiente Stabilized version, with the Incomplete LU Threshold(ILUT) subroutine of the package SPARSKIT, which was modified in order to account for complex numbers. Using scalar and vectorial finite element formulations, the implemented subroutines were applied to the propagation of optical pulses and beams in different structures like fiber couplers, channel waveguides and 90^{o} junction with photonic crystals. In such analysed structures, results here obtained showing that the iterative methods are a better choice when compared to linear direct solvers using gaussian elimination (strictly speaking, the ME28). In fact, this iterative scheme permits the use of about 250,000 unknowns, while the ME28 is limited to aproximately 75,000 unknowns. All subroutines were written and run in Fortran 77 under the operational system LINUX RED HAT.

Agradecimentos

Agradeço aos
meus pais , irmãos,
aos colegas do departamento e a todos que contribuíram para a realização deste trabalho direta ou indiretamente.
Agradeço ao órgão de fomento à pesquisa CNPq – Conselho Nacional de Desenvolvimento Científico e Tecnológico, pelo financiamento por dois anos deste trabalho.

Dedicatória

Dedico este trabalho aos meus antepassados e a Deus, dando força e clareza para avançar, nos momentos mais difíceis e decisivos, nesta caminhada.

Conteúdo

Lista de figuras

Lista de tabelas

Lista de símbolos

A	-	Matriz de coeficientes
x	-	vetor solução do sistema linear
b	-	vetor independente
x,y,z	-	coordenadas espaciais
h_x	-	campo magnético na direção x
h_y	-	campo magnético na direção y
Aa	-	amplitude do campo
w	-	largura do pulso
Δz	-	passo de propagação
N	-	dimensão da matriz quadrada
Nz	-	números de valores não-nulos
lfil	-	parâmetro de preenchimento
droptol	-	parâmetro de estabilidade
n_i	-	índice de refração do meio i
λ_0	-	comprimento de onda
iter	-	iterações
tbcgs	-	tempo de iteração.utilizando o biconjugado estabilizado precondicionado
tilu	-	tempo de decomposição da ILU
tbi	-	soma dos tempos tbcgs e tilu

Capítulo 1

Introdução

A resolução de grandes sistemas esparsos de equações lineares é utilizada para simular situações do mundo real e tem muita importância nas ciências aplicadas. São fundamentais em um grande número de áreas tais como, Mecânica, Construção Civil, Medicina, Aviação, Logística e Comunicações.

É uma etapa fundamental na resolução de vários problemas que envolvam, por exemplo, equações diferenciais parciais que são geralmente solucionadas usando métodos numéricos como elementos finitos e diferenças finitas.

A capacidade de solução de problemas numéricos está relacionada com a capacidade computacional disponível, porém esta capacidade é constantemente renovada com os avanços da indústria de computadores, pois à medida que temos equipamentos mais rápidos e com maior capacidade de armazenamento de dados, maiores problemas tornam-se objeto de estudo. Com isto, técnicas numéricas que eram consideradas caras computacionalmente passam a ser empregadas.

O método numérico chamado Elementos Finitos [1] é um bom exemplo, pois cada vez que a capacidade computacional se renova, problemas com maior número de variáveis podem ser resolvidos e novos problemas, que eram inviáveis, começam a ser pesquisados.

Neste trabalho, a solução dos sistemas de equações lineares terão aplicação em eletromagnetismo, com simulações numéricas através da discretização das equações de onda pelo método dos Elementos Finitos. Os sistemas oriundos da discretização por elementos finitos são, em geral esparsos. E foram obtidos de trabalhos do DMO/FEEC/UNNICAMP (Departamento de Microonda e Óptica) em propagações de ondas no domínio do tempo, bem como no domínio da freqüência.

A resolução de sistema esparso pode tornar-se mais eficiente quando aproveita os elementos nulos para reduzir o número de operações efetuadas. Para que isto ocorra é necessário uma adaptação dos algoritmos utilizados, bem como um tratamento de armazenamento diferenciado dos dados.

O aproveitamento da esparsidade da matriz teve início em diversas áreas de engenharia para "matrizes banda", porém as primeiras aplicações ocorreram em redes elétricas nos anos 60 [2]. A utilização das técnicas de aproveitamento da esparsidade foi importante, por apresentar um melhor desempenho, mas o mais relevante foi a possibilidade de solução de problemas que, até então, eram considerados inviáveis computacionalmente. No Capítulo 2 trataremos de uma maneira mais geral sobre a solução de sistemas esparsos: armazenamento de dados e ordenação das matriz de coeficiente.

Podemos definir os métodos de resoluções de sistemas lineares em dois grupos: diretos [3] e iterativos [2]; os métodos diretos são aqueles que trabalham com a decomposição gaussiana, chegando a soluções exatas e os métodos iterativos são métodos que refinam a solução a cada iteração; os métodos iterativos mais modernos trabalham com busca no espaço de Krylov [4].

O espaço de Krylov é gerado por um polinômio matricial, os vetores soluções em cada passo do método iterativo é gerados pela base de Krylov. Estes métodos serão aprofundados nos Capítulos 3.

Este trabalho tem como objetivo principal alavancar a capacidade de solucionar grandes sistemas lineares que atualmente se dispõe no DMO. Neste primeiro trabalho do DMO nesta área, não nos prenderemos ao código completo de propagação, mas apenas ao gargalo do problema que é a resolução de sistemas lineares oriundo de trabalhos anteriores realizados no DMO. As ferramentas de soluções utilizadas atualmente são a subrotina ME28 [5] e o Matlab. A ME28 soluciona o sistema baseando-se na eliminação gaussiana. No Matlab utilizamos funções pré-definidas do método biconjugado estabilizado precondionado e a decomposição LU incompleta, como precondicionador. Pretendemos apresentar uma alternativa a estas, pois para os problemas aqui estudados a ME28 está limitada a resolver sistemas lineares de, no máximo, 75.000 variáveis e O(n) valores não-nulos; no Matlab através do método biconjugado estabilizado é possível ir além, porém o fato de ser uma linguagem de mais alto nível não oferece bom desempenho.

Ambas ferramentas, ME28 e o Matlab, são externa ao grupo que caracteriza uma desvantagem, gerando encargos.

Como alternativa, implementaremos o método iterativo gradiente biconjugado estabilizado [6] pelo fato deste não possuir restrições em relação ao tipo de matriz, pois é aplicável a matrizes não-simétricas e não-hermitianas, o que permite uma maior aplicabilidade nos problemas tratados pelo grupo.

Para solução de sistemas esparsos de grande porte, além do método iterativo, são necessárias ferramentas de ordenamento das matrizes e de precondicionamento, utilizamos as disponíveis no pacote Sparskit [7]. A Sparskit é projetada para trabalhar com valores reais; como trabalhamos com sistemas de variáveis complexas; a rotina de precondicionamento foi alterada.

Para validação da subrotina implementada faremos vários testes, primeiro verificaremos a escalabilidade, depois sua concordância com resultados já conhecidos ou consagrados em teses de doutorado do DMO/FEEC. A subrotina desenvolvida neste trabalho será denominada por ***LIRIOS***.

Esperamos, com a construção do pacote implementado, poder contribuir para o refinamento de soluções e, com isto, observar com mais detalhes os fenômenos ópticos em um guia de onda, tornando-se mais uma ferramenta de trabalho disponível.

No Capítulo 4 apresentaremos mais detalhes sobre essa implementação, também apresentaremos os testes, nos quais poderemos avaliar a LIRIOS em relação à escalabilidade.

Todas as implementações foram realizadas com programação serial, embora os métodos poderiam ser programados para computadores paralelos ou uma rede de computadores. Os ambientes de troca de informações, podem ser MPI (Messaging Passing Interface) e o PVM (Parallel Virtual Machine). No Apêndice C apresentamos esses ambientes paralelos com mais detalhes. No Capítulo 5 apresentamos os resultados obtidos nas aplicações. No Apêndice A temos tabelas que complementam os testes apresentados no Capítulo 5. No Apêndice B apresentamos algumas dicas sobre a utilização da plataforma linux, tendo em vista a extensa utilização no, meio científico, desta versão similar ao UNIX, porém mais barata. No Apêndice D faremos uma descrição do método de elementos finitos [1].

Capítulo 2

Solução de sistemas esparsos

Um dos problemas principais, em soluções numéricas, que sugem em diversas áreas de engenharia e ciência básica, é a solução de sistemas de equações lineares do tipo:

$$Ax = b\,, \qquad (2.1)$$

onde A é uma matriz esparsa quadrada, de dimensão n e b, x *são* vetores colunas de dimensão n. Os sistemas lineares são encontrados quando o problema de solução de equações diferenciais são discretizados, como por exemplo, ao utilizamos elementos finitos ou diferenças finitas [8].

Classificamos os métodos de soluções em duas categorias métodos diretos e iterativos. Os métodos, iterativos ou diretos, utilizados ao resolvermos sistemas lineares esparsos, aproveitam a esparsidade das matrizes, objetivando sempre economizar memória computacional e operações com pontos flutuantes, que refletirá em tempo de execução.

Ao resolvermos sistemas lineares esparsos, os métodos utilizados aproveitam a esparsidade das matrizes para economizar memória computacional e operações com pontos flutuantes. A aplicação desses tratamentos de dados é fundamental, devido ao fato dos recursos computacionais serem finitos. Portanto quanto mais eficiente for o armazenamento de dados, maior a capacidade de solução dos problemas e maior será a precisão; os detalhes dessas considerações serão apresentados nas seções posteriores deste capítulo.

Um outro aspecto a ser levado em consideração, na solução de sistemas lineares esparsos, é a geração dos elementos não-nulos, chamados *fill-ins*, durante a eliminação gaussiana [9]; para amenizar este fator, é feito um ordenamento das linhas possibilitando, também, monitorar os pivôs, garantindo a estabilidade numérica.

Embora a primeira vista somente os métodos diretos sofreriam com os *fill-ins*, métodos iterativos também sofrem com esse problema ao utilizarmos métodos precondicionados baseados em decomposição LU incompleta. Na Seção 3.2.4 trataremos com mais detalhes sobre esses precondicionadores.

2.1 Armazenamento de dados

Os algoritmos de solução de sistemas lineares se sustentam em uma forma de armazenamento de dados diferenciada. O objetivo principal deste armazenamento diferenciado é não desperdiçar memória com os valores nulos da matriz; outro fator da escolha é a eficiência do código.

Considere uma matriz de dimensão N e com O(N) valores não-nulos, caso aplicássemos a forma convencional, gastaria-se um espaço de memória correspondente a O(N^2) valores, caso aplicássemos uma forma que armazenasse apenas os valores não-nulos, se ocuparia apenas um espaço de memória correspondente a O(N) valores.

Ao utilizarmos uma estrutura de dados eficiente, além de se ocupar um espaço reduzido de memória, também facilita uma economia de processamento, pois as operações triviais de multiplicação e soma com o valor nulo serão eliminadas do algoritmo.

Descreveremos exemplos de formas de armazenamento, considerando uma única matriz, com dimensão $N = 3$ e $Nz = 6$ elementos não-nulos, como na Figura 2.1

$$\begin{pmatrix} 1 & 0 & 1 & 4 \\ 2 & 2 & 3 & 1 \\ 0 & 0 & 5 & 6 \\ 0 & 0 & 2 & 7 \end{pmatrix}$$

Figura 2.1 - Matriz exemplo para formas de armazenamento

2.1.1 Coordenadas

O sistema de armazenamento em coordenadas é o mais simples e mais intuitivo, consiste em armazenar a matriz em três vetores:

1- valor

2- índices das linhas

3- índices das colunas

A matriz da Figura 2.1 é representada na Tabela 2.1:

Valor	1	1	4	2	2	3	1	5	6	2	7
Linha I	1	1	1	2	2	2	2	3	3	4	4
Coluna J	1	3	4	1	2	3	4	3	4	3	4

Tabela 2.1 - Representação da matriz esparsa da Figura 2.1, em coordenadas

O vetor *Valor* corresponde aos valores não-nulos da Figura 2.1, e os vetores *Linha I* e *Coluna J* correspondem a linhas e colunas correspondentes as componentes do vetor *Valor*.

Nesta representação, foram necessários O(NZ) armanezamentos, enquanto para o armanezamento convencional ocupa O(N^2). Este fato nos diz que, para $NZ << N^2$, há uma economia significativa de espaço de memória computacional.

2.1.2 Comprimido por colunas

Neste caso consideramos que o vetor de valores esteja ordenado por colunas, economizando espaço no armazenamento do vetor coluna:

1- valor

2- índices das linhas

3- posição inicial das colunas em 1

A matriz da Figura 2.1 fica representada na Tabela 2.2:

Valor	1	2	2	1	3	5	2	4	1	6	7
Linha I	1	2	2	1	2	3	4	1	2	3	4
Colun. Compr.	1	3	4	8	12	-					

Tabela 2.2- Representação da matriz esparsa da Figura 2.1 comprimida por colunas

Neste caso temos uma diferenciação em relação ao armazenamento por coordenadas, neste caso o vetor *Valor* está em ordem crescente de coluna e o vetor *Coluna Comprimida* indicará onde inicia cada coluna no vetor *Valor*.

Detalhando um pouco mais: a primeira componente do vetor *Coluna Comprimida* indica em que posição inicia a *coluna 1* no vetor *Valor* que é igual a *1*. A segunda componente indica que a *coluna 2* inicia-se na posição 3 do vetor *Valor* na n+1 posição temos a indicação do final do vetor de *Valor*.

Com esta representação o espaço de memória ocupada é referente a *2*NZ + N+1* que é menor que os 3**NZ* obtidos em 2.2.1.

2.1.3 Comprimido por linhas

Consideramos, neste caso, que o vetor de valores esteja ordenado por linhas, economizando espaço no armazenamento do vetor linha:

1- valor

2- índices das colunas

3- posição inicial das linhas em 1

A matriz da Figura 2.1 fica representada da seguinte maneira (Tabela 2.3):

Valor	1	1	4	2	2	3	1	5	6	2	7
Coluna J	1	3	4	1	2	3	4	3	4	3	4
Linha Compr.	1	4	8	10	12	-					

Tabela 2.3 - Representação da matriz esparsa da Figura 2.1, comprimida por linhas

Este caso é similar ao Comprimido por Colunas, porém temos o vetor *Valor* ordenado por linhas e a componente *i,* do vetor *Linha Comprimida, indica* em que posição do vetor do *Valor* a linha *i* está iniciando.

Com esta representação o espaço de memória ocupada é dado por a *2*NZ + N +1*, que é menor que os 3**NZ* obtidos em 2.2.1.

2.2 Ordenamento

Os *fill-ins* são as alterações realizadas na matriz durante o processo de fatoração, que mudam os valores nulos para valores não-nulos, tornando a matriz menos esparsa e, por conseqüência, elevando o custo para a solução do sistema linear.

Mostraremos, abaixo, exemplos do aparecimento dos *fill-ins* nas Figura 2.2 e 2.3. Este exemplo foi retirado da referência [3]. Na Figura 2.2 temos os *fill-ins*, gerados durante a eliminação gaussiana, representados pelos quadrados negros; na Figura 2.3 temos a matriz ordenada pelo método mínimo grau, o qual iremos detalhar mais adiante. Notamos na Figura 2.3 que não há geração de nenhum *fill-in*.

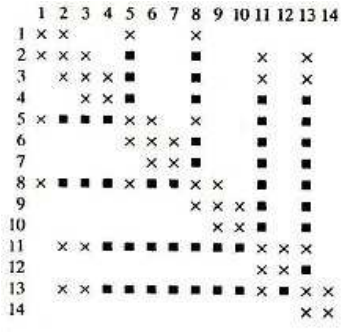


Figura 2.2 - Visualizando os "fill ins" [3]

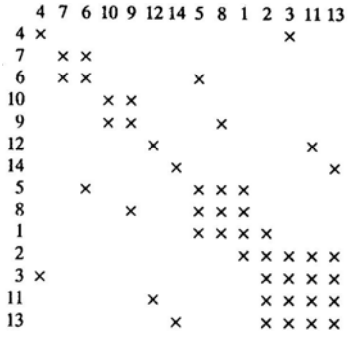


Figura 2.3 - Matriz ordenada por grau mínimo [3]

Apresentaremos um breve histórico sobre ordenamento. Em 1957, Markowitz [10], publicou a primeira estratégia de ordenação/pivoteamento aplicada em matrizes assimétricas, para qualquer problema. Outras técnicas foram introduzidas por Sato e Tinney [11], em 1963.

Em 1967, Tinney e Walker, preocupados com a ordenação ótima, publicaram outras técnicas de ordenação, chamadas Critérios de Ordenação 1,2 e 3 e divididos de acordo com sua complexidade. O Critério de Ordenação 2 é aplicado a sistemas lineares simétricos definido positivos e trabalha através da ordenação da matriz de modo a encontrar um grafo equivalente de grau mínimo [12].

Pesquisas comparativas dos métodos de ordenação e pivoteamento foram publicados por Tinney, Powell e Peterson (1974) [13], Duff e Reid (1974) [14], Erisman e Tinney (1975) [15].

O maior problema, ao ordenar a matriz, é também garantir um bom pivoteamento. Mostraremos, neste trabalho, duas técnicas de ordenamento: Markowitz e Tinney2 (grau mínimo) [12]

Ao desenvolver ou implementar um método, é sempre desejável uma economia de memória computacional e o menor número possível de operações com pontos flutuantes. Estes objetivos são as bases que norteiam o desenvolvimento dos pacotes para soluções de sistemas lineares de grande porte, pois os recursos computacionais são finitos, e também, devido ao fato dos computadores possuírem uma precisão finita. Quando relacionamos os métodos ou pacotes aos recursos disponíveis, outros aspectos a serem levados em consideração são:

- sistema operacional a ser utilizado (linux, windows, sun, etc);
- aplicação do método;
- paradigma de programação: estruturada ou orientada a objeto;
- tipo de processamento, paralelo ou serial;
- disponibilidade financeira;
- país onde é realizada a pesquisa, pois muitos equipamentos são vistos como estratégicos e suas vendas são restritas por leis protecionistas;

2.2.1 Ordenamento de Markowitz

O ordenamento de Markowitz (1957) [10] é aplicado a uma matriz geral qualquer; ele consiste em escolher, na *k*-ésima Eliminação de Gauss, o pivô a_{ij} da submatriz de dimensão *(n-k)x(n-k)* :

$$|a_{ij}| \geq \alpha^* max |a_{lj}| \qquad (2.5a)$$

ou

$$|a_{ij}| \geq \alpha^* max |a_{il}|, \qquad (2.5b)$$

para todo $l \geq k$, $0 \leq \alpha \leq 1$.

O valor "alfa" acima permite a possibilidade de escolha de um pivô que não precisa ser, necessariamente, o valor máximo; em contrapartida, teremos a liberdade da escolha daquele que minimiza o preenchimento da matriz esparsa por valores não-nulos.

Escolhidos os possíveis pivôs, escolhemos a linha a ser ordenada tal que minimize o número de entradas r_t não-nulas, referentes às linhas (r_i) e colunas: (c_j). Desta forma, temos esta regra na Equação (2.6).

$$r_t = Min\ (r_i - 1)\ (c_j - 1) \qquad (2.6)$$

Nas Figuras 2.4 e 2.5, temos a ordenação de Markowitz.

x	x	x	x
x	x		
x		x	
x			x

Figura 2.4 - Matriz original

x			x
	x		x
		x	x
x	x	x	x

Figura 2.5 - Matriz ordenada

2.2.2 Ordenamento de grau mínimo

O ordenamento de grau mínimo [12] também é conhecido como Tinney2 ou ainda Esquema B; o nome de Grau Mínimo é atribuído porque escolhe-se os pivôs de modo que, no grafo correspondente, escolhe-se aquele nó que possui o menor número de arestas a ele conectadas. Um nó de grau mínimo é o nó, dentre todos os nós do grafo, que possui o menor número de arestas ligadas a ele, muito embora isto não garantirá que os números de *fill-ins* seja mínimo.

Nas Figuras 2.6 e 2.7 temos uma matriz exemplo e o grafo correspondente.

	1	2	3	4	5	6
1	x	x		x		
2	x	x	x	x	x	
3		x	x		x	
4	x	x		x	x	x
5		x	x	x	x	
6				x		x

Figura 2.6 - Matriz original – grau mínimo

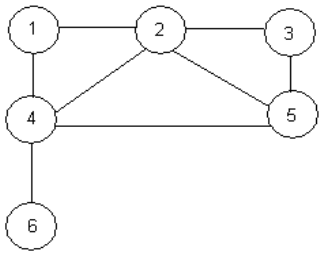


Figura 2.7 - Grafo correspondente a Figura 2.6

Escolhendo os nós que possuem grau mínimo, para serem os primeiros pivôs no processo de fatoração LU, temos a seguinte matriz ordenada (Figura 2.8):

	6	*1*	*3*	*5*	*2*	*4*
6	x					x
1		x			x	x
3			x	x	x	
5			x	x	x	x
2		x	x	x	x	x
4	x	x		x	x	x

Figura 2.8 - Matriz ordenada por grau mínimo

Formalizaremos, adiante, o método de mínimo grau em termos de elementos não-nulos da matriz que trará uma outra abordagem, facilitando o tratamento sistemático da ordenação.

Este método de ordenação pode ser visto como um caso particular do método de Markowitz. O método de grau mínimo é aplicado às matrizes simétricas definidas positivas; em cada iteração *k* da eliminação gaussiana escolhemos como linha pivotal *i* aquela que possui o menor número r_t *de elementos não-nulos.*

$$r_i = Min\ (r_t),\ \ t = 1 \ldots n\ . \qquad (2.7)$$

O ordenamento do exemplo da Figura 2.2 será detalhado logo a seguir nas Figuras 2.9, 2.10 e 2.11. Nas representações deste exemplo as linhas e colunas dos pivôs estão em negrito e as submatrizes ativas (sujeitas a decomposição) são aquelas que estão em fundo branco, as linhas e colunas já ordenadas estão em itálico.

Passo 1:

O elemento (4,4) é escolhido como pivô, ilustrado na Figura 2.9 em negrito, segundo a regra expressa em (2.7).

	1	2	3	**4**	5	6	7	8	9	10	11	12	13	14
1	x	x			x			x						
2	x	x	x								x		x	
3		x	x	x							x		x	
4			x	x										
5	x				x	x		x						
6					x	x	x							
7						x	x							
8	x				x			x	x					
9								x	x	x				
10									x	x				
11		x	x								x	x	x	
12											x	x		
13		x	x								x		x	x
14													x	x

Figura 2.9 -Passo 1 do ordenamento mínimo grau

Passo 2:

Troca-se de linha 4 e coluna 4 com linha 1 e coluna 1, respectivamente, escolhe-se o novo pivô segundo a regra (2.7), em negrito, lembrando que é aplicada à submatriz ativa; neste caso, o pivô será o elemento da posição (7,7). A variável já ordenada está em cinza, e a submatriz com fundo branco é a submatriz ativa.

	4	2	3	1	5	6	**7**	8	9	10	11	12	13	14
4	x		x											
2		x	x	x							x		x	
3	x	x	x								x		x	
1		x		x	x			x						
5				x	x	x		x						
6					x	x	x							
7						x	x							
8				x	x			x	x					
9								x	x	x				
10									x	x				
11		x	x								x	x	x	
12											x	x		
13		x	x								x		x	x
14													x	x

Figura 2.10 - Passo 2 do ordenamento de mínimo grau

Repetindo os passos 1 e 2 teremos, no passo 14:

Passo 14:

	4	***7***	***6***	***10***	***9***	***12***	***14***	***8***	***5***	***1***	***11***	***3***	***13***	**2**
4	x											x		
7		x	x											
6		x	x						x					
10				x	x									
9				x	x			x						
12						x					x			
14							x						x	
8					x			x	x	x				
5			x					x	x	x				
1								x	x	x				x
11						x					x	x	x	x
3	x										x	x	x	x
13							x				x	x	x	x
2										x	x	x	x	x

Figura 2.11 - Passo 14 do ordenamento de mínimo grau

Notamos na Figura 2.11 que o resultado final é ótimo, pois não aparece nenhum *fill-in* quando executa-se a decomposição LU, embora este método não seja um método ótimo. Notamos, também, que a solução não é única, conforme constata-se ao comparar o resultado obtido em nosso exemplo e o apresentado na Figura 2.3, retirada de [3].

Quando fazemos a Eliminação de Gauss devemos ter a preocupação com a estabilidade numérica; esta estabilidade fica comprometida quando o elemento pivotal possui um valor numérico pequeno. Desta forma quando aplica-se o método de grau mínimo, pode-se incluir uma condição adicional para que a estabilidade seja garantida: que o elemento pivotal seja maior, em módulo, que um dado valor ε real positivo .

Capítulo 3

Métodos diretos e Métodos iterativos

3.1 Métodos diretos

Chamamos de métodos diretos aqueles que baseiam-se na decomposição gaussiana [9] para a solução do sistema linear.

Um aspecto a ser levado em consideração, na solução de sistemas lineares esparsos, é a geração dos elementos não-nulos, chamados "fill ins", durante a eliminação gaussiana; para amenizar este fator, é feito o ordenamento das linhas possibilitando, também, monitorar os pivôs, garantindo a estabilidade numérica.

3.1.1 Histórico

As técnicas de resolução de sistemas lineares esparsos inicia-se na década de 50. Porém o princípio básico da maioria destas técnicas é a eliminação gaussiana introduzida por Carl Friedrich Gauss em 1743. Os chamados Métodos Diretos são todos aqueles que se baseiam na Eliminação de Gauss.

Nos métodos diretos, referentes a fatoração dos sistemas lineares, temos algumas variações da eliminação gaussiana:

- em 1970, Irons [16] introduziu o Método Frontal que mais tarde evolui-se para o Método Multifrontal, ambos inspirados na solução de sistemas gerados pela discretização utilizando elementos finitos;
- em 1971, Zollenkopf [17] apresentou o método da Bi-Fatoração, baseado em dois outros métodos Decomposição Triangular, e os Produtos das Inversas;

- em 1981, Eisenstat, Schultz e Sherman [18] desenvolveram o método da Eliminação Simétrica e Esparsa de Gaus. Esse método é aplicado a sistemas cujas matrizes são quadradas, simétricas, definidas positivas e esparsas.

A partir da década de 80, iniciaram-se vários trabalhos em solução de sistemas lineares esparsos via processamento paralelo, pois haviam sido estudadas, até então, as soluções via processamento serial. O processamento paralelo pode ser implementado em ambientes de trabalhos, tais como MPI e PVM, tratados no Apêndice C; nele é apresentado um primeiro estudo, preparando-se para trabalhos futuros com computação paralela.

3.1.2 O Método

São considerados métodos diretos para a solução de sistemas de equações lineares, os métodos que têm,como base, a Eliminação de Gauss. Ao implementar a eliminação de Gauss, a forma mais utilizada é a fatoração da matriz *A,* dos coeficientes, no produto de duas matrizes triangulares, inferior (*L) e* superior (*U*):

$$Ax = b \qquad (3.1)$$

$$A = L * U \qquad (3.2)$$

Substituindo (3.2) em (3.1) temos:

$$L * U * x = b \qquad (3.3)$$

De (3.3) temos o sistema matricial abaixo:

$$L * y = b \qquad (3.4a)$$

$$U * x = y \qquad (3.4b)$$

Ao resolvermos um sistema de equações lineares esparso, é conveniente separarmos os métodos em três etapas, pois isto garantirá uma maior eficiência ao resolver problemas que exigem a solução de vários sistemas com a mesma estrutura esparsa; estas etapas são: Análise, Fatoração e Solução [3]. Este tipo de fragmentação é feito na subrotina ME28 [5].

1. Análise:

A análise é a etapa que avalia a estrutura esparsa da matriz, faz o ordenamento, escolhendo o pivô, prepara a estrutura de dados para uma execução eficiente nas etapas futuras e fornece saídas de estatísticas.

Dizemos que um sistema linear é estável quando os erros, acumulados nas diversas operações, não interferem significativamente nos resultados. Estes erros são causados devido ao fato dos computadores possuírem uma precisão finita; assim arredondamentos poderão causar grandes erros. Quando trabalhamos com solução de sistemas lineares, dizemos que a estabilidade está relacionada com a boa escolha dos pivôs, durante a eliminação gaussiana.

Um bom ordenamento da matriz é aquele que, além de satisfazer as exigências de minimizar os *fill-ins* que aparecem durante o processo da fatoração da matriz de coeficientes do sistema, também irá proporcionar uma estabilidade numérica.

2. Fatoração:

A Fatoração é a etapa onde a matriz dos coeficientes é fatorada de maneira a evitar o cálculo da inversa da matriz, cujos cálculos são demasiadamente custosos do ponto de vista computacional.

Para solução de sistemas lineares esparsos, a fatoração mais utilizada é a LU, embora existam variações na forma como estas fatorações são manipuladas.

Apresentaremos dois métodos chamados frontal [16] e multi-frontal [20], ambos inspirados em problemas que discretizam o domínio através do método dos elementos finitos.

3. Solução:

Após realizada a fatoração, temos a matriz A como o produto entre $L * U$; a solução corresponde a resolver o sistema matricial exposto na Equação (3.3). Esta fase é a mais rápida e simples de ser implementada, mas é importante uma implementação eficiente, pelo fato de se precisar executá-la várias vezes, em algumas aplicações.

3.1.2.1 Método frontal

O método frontal tem sua origem na solução de problemas de elementos finitos, mais especificamente em problemas de estrutura mecânica [16]. Porém o método não se restringe à solução de problemas de elementos finitos, podendo ser aplicados a outros problemas [3], [21].

No problema de elemento finitos temos que a matriz global A é formada pela contribuições das matrizes locais referentes a cada elemento:

$$A = \sum A^{(l)}, \qquad (3.8)$$

onde $A^{(l)}$ são as matrizes locais de dimensão igual à dimensão da matriz A, porém seus valores são nulos nas componentes referente aos nós que não fazem parte do elemento em questão. Chamaremos à expressão (3.8) de "montagem" e temos a seguinte operação elementar desta expressão:

$$a_{ij} = a_{ij} + a_{ij}^{(l)}. \qquad (3.9)$$

Quando a componente a_{ij} da matriz global não tiver mais nenhuma contribuição, conforme (3.9), dos elementos da malha, dizemos que a_{ij} está "completamente somada". Quando todos os termos envolvidos na operação da Eliminação Gaussiana estiverem completamente somados, poderemos efetuar a eliminação gaussiana:

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - (a_{ik}^{(k)} / a_{kk}^{(k)})\, a_{kj}^{(k)} \qquad (3.10)$$

Cada variável pode ser eliminada quando a linha e coluna estiverem completamente somadas. Chamamos de variáveis ativas aquelas que ainda não foram eliminadas, mas estão envolvidas com elementos que não estão completamente somados.

As variáveis ativas formam uma frente que caminhará sobre a malha à medida que as variáveis são eliminadas. A formação desta frente é o fato que origina o nome Método Frontal.

Consideremos a malha formada por elementos triangulares da Figura 3.1 , retirada de [3]:

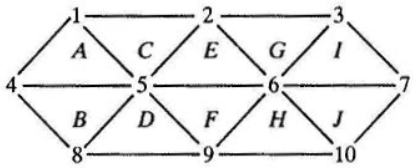


Figura 3.1 - Malha de elememtos [3]

Temos os seguintes passos, durante a fatoração LU, esboçado pelas Figuras 3.2 e 3.3:

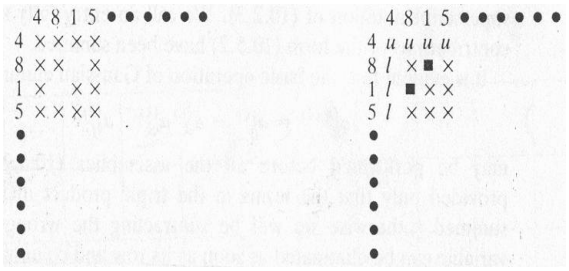


Figura 3.2 - Fatoração LU pelo método frontal – passos 0 e 1[3]

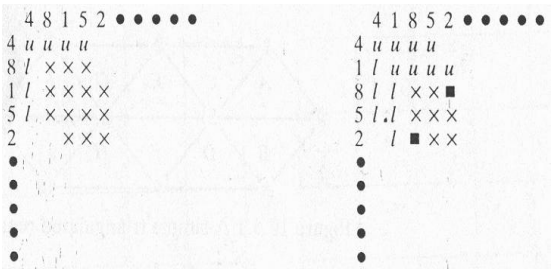


Figura 3.3 - Fatoração LU pelo método frontal – passos 2 e 3 [3]

Iniciando a fatoração da esquerda para a direita, na malha apresentada as variáveis ativas são 4, 8, 1 e 5 e o nó 4 não possui mais contribuição de nenhum outro nó, portanto está completamente somado e poderá ser eliminado, como mostra a Figura 3.2.

Continuando a fatoração, o próximo elemento a contribuir é o elemento C; logo o nó 1 passa a ser completamente somado e poderá ser eliminado, como mostra a Figura 3.3.

No *k*-ésimo passo temos a configuração da Figura 3.4.

$L_{11} \diagup U_{11}$	U_{12}	0
L_{21}	F	
0		

Figura 3.4 - Representação da matriz no *k*-ésimo passo

Na Figura 3.4, L_{11} , U_{11}, L_{21} e U_{12} são os blocos de submatrizes já completamente somadas , F é a submatriz frontal (variáveis ativas), nos blocos (3,1) e (1,3) as variáveis já estão completamente somadas e eliminadas e, nos blocos restantes, as variáveis ainda podem receber contribuições dos elementos.

3.1.2.2 Método multi-frontal

O método Multi-frontal [3], [20], consiste em uma generalização do método frontal; agora teremos mais de uma frente de variáveis ativas, desta forma teremos uma árvore que representará as várias frentes. Estas frentes são formadas pelo rearranjo do somatório da montagem (3.8)

Consideremos o problema de elementos finitos da Figura (3.5).

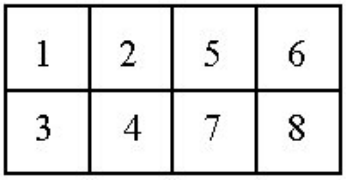


Figura 3.5 - Elementos finitos - método multi-frontal

A árvore referente à Equação (3.11), é dada pela Figura 3.6, a montagem da árvore é feita iniciando-se pelo colchete mais interno de (3.11), o resultado do fechamento do colchete gera um nó pai das somas internas, cujas componentes são nós filhos, esse nó pai será um nó filho no próximo colchete, somando temos novamente um nó pai, assim por diante até o colchete mais externo.

$$A = [\ldots [[[A^{(1)} + A^{(2)}] + A^{(3)}] + A^{(4)}] + \ldots + A^{(8)}]. \qquad (3.11)$$

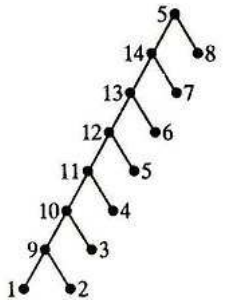


Figura 3.6 - Árvore correspondente a montagem [3]

Rearranjando a soma de outra maneira, temos:

$$A = [A^{(1)} + A^{(2)}] + [A^{(3)} + A^{(4)}] + \ldots + [A^{(7)} + A^{(8)}] \qquad (3.12)$$

Na Figura 3.7 temos o domínio dividido conforme a Equação (3.12), que representam as quatro frentes.

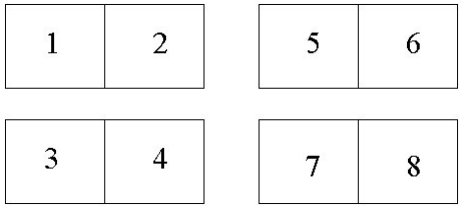


Figura 3.7 - Representação do método multi-frontal

No final da recursão teremos apenas os valores da interface das frentes, que na Figura 3.8, representam os nós artificiais 13, 14 e 15.

Temos a respectiva representação gráfica, para $l = 8$, na Figura 3.8:

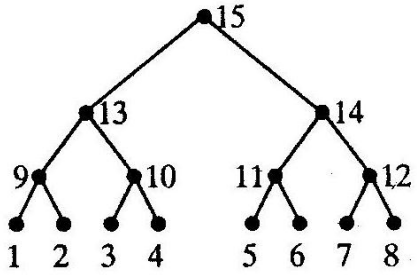


Figura 3.8 - Árvore correspondente a segunda montagem [3]

Neste caso, aplica-se o mesmo procedimento para entender a representação; devemos considerar que os nós filhos representam os elementos e, para cada nó pai, temos o fechamento de um colchete da respectiva soma, sendo os nós pais elementos artificiais, pois não participam, de fato, da malha.

3.2 Métodos iterativos

Os métodos iterativos são métodos de solução de sistemas de equações lineares que trabalham com o refinamento do vetor solução até que se obtenha uma precisão aceitável. Podemos classificar esses métodos em dois tipos: estacionários e não-estacionários [19].

Os métodos estacionários possuem critérios de busca fixos durante as iterações. Os métodos estacionários são os mais tradicionais (Jacobi, Gauss-Seidel e SOR) com grande valor histórico, porém são muitos restritivos, aplicando-se a sistemas esparsos diagonalmente dominante.

Os métodos iterativos não-estacionários (Gradiente Conjugado e suas variações) possuem parâmetros de busca que variam a cada iteração; esta flexibilidade justifica o seu melhor desempenho em relação ao estacionário. Os métodos não-estacionários trabalham com busca de soluções em espaços específicos (Krylov), encontradas através de seqüências de projeções ortogonais ou biortogonais [2].

Geralmente eles são melhores e mais aplicáveis a problemas práticos de grande porte em 2D [20]. A convergência dos métodos iterativos dependem significativamente do condicionamento da matriz, que, por sua vez, depende dos autovalores da matriz de coeficientes do sistema linear [19]. O mal-condicionamento pode ser interpretado como um reflexo da uniformidade dos autovalores; uma matriz com um autovalor muito pequeno e um muito grande será mal-condicionada, na seção 5.2 o condicionamento é definido com mais precisão.

Para resolver os sistemas cujas matrizes são mal-condicionadas, utilizam-se pré-condicionadores; os pré-condicionadores adicionam um custo adicional aos métodos, porém este é o preço pago para que tenhamos convergência e, ainda, para termos uma solução confiável.

Os pré-condicionadores são a parte chave destes métodos, pois para cada tipo de matriz, teremos pré-condicionadores que serão mais eficientes ou menos eficientes. Desta forma, a eficiência deste tipo de método esta intrinsecamente ligada à boa escolha do pré-condicionador.

O papel desempenhado pelos pré-condicionadores será melhorar o espectro da matriz, de modo a proporcionar a convergência do método, trabalhando sobre matrizes bem-condicionadas, ou seja, que tenham pouca variação entre os autovalores próximos da unidade, ou ainda um espectro mais uniforme.

Os métodos iterativos são geralmente utilizados para grandes sistemas de equações, para serem tratados através dos métodos diretos [21]. Apesar dos métodos iterativos serem aplicáveis a grandes sistemas de equações, a convergência não é garantida.

Os métodos iterativos estacionários podem ser descritos da seguinte maneira:

$$x^k = T * x^{k-1} + c \qquad (3.13)$$

Nos métodos estacionários, os valores de T e c permanecem constantes durante as iterações k, enquanto que, nos métodos não-estacionários, há uma flexibilização durante as iterações e c passa a variar.

Relatamos na próxima seção, um breve histórico sobre os métodos iterativos [21]. Em seguida faremos, uma exposição de cada método e de seus respectivos algoritmos, iniciando pelos métodos estacionários e depois com os métodos não-estacionários.

3.2.1 Histórico

Os métodos estacionários foram os primeiros métodos iterativos, que surgiram no início do século 19. Naquela época já havia uma preocupação com o número de cálculos necessários para a solução de sistemas lineares, o que levou Jacobi a formular, em 1846, o método conhecido como Gauss-Jacobi, ou apenas, Método de Jacobi; nesta época Jacobi resolveu um sistema com 7 equações. O Método Gauss-Seidel fora, primeiramente, apresentado por Gauss, em 1823, segundo [22], e mais tarde, em 1873, por Seidel. No século XX, o primeiro método a surgir foi o método de Richardson em 1910 [23]; em meados do século com a intenção de acelerar o método Gauss-Seidel, surgiu os métodos de relaxações sucessivas (SOR – "Succesive Over-Relaxation"), introduzidos por Frankel [24] e Young [25]. O método SOR e suas variações tornaram-se muitos populares e com aplicações de grandes dimensões, que duraram até os anos 80, quando técnicas mais eficientes começaram a ser aplicadas, embora ainda sejam usados em muitas aplicações, como precondicionadores.

Em 1952, temos um marco para os métodos iterativos,com o surgimento do Método do Gradiente Conjugado com a publicação de Hesteness e Stieffel [26] e o Método de Lanczos [27], iniciando a época dos métodos iterativos baseados nos espaços de Krylov. Em 1976, Flecher [28] apresenta o método gradiente biconjugado, aplicável a sistemas não-simétricos e indefinidos.

Em 1984, surge o método biconjugado quadrático, apresentado por Sonneveld [29], que acelera o método biconjugado, mas ainda possui instabilidades. Em 1992, Vorst [6] apresenta o método mais estável, que corrige as instabilidades do biconjugado quadrático; este método pode ser colocado como um segundo marco nos métodos iterativos aumentando significativamente as aplicações, principalmente para os problemas grandes.

3.2.2 Algoritmos estacionários

Neste tópico apresentaremos os métodos estacionários: Jacobi, Gauss-Seidel e SOR.

3.2.2.1 Método Jacobi

Para entendermos o método de Jacobi, considere o sistema da Equação (3.1), decompomos a matriz *A* conforme a Equação (3.14):

$$A = (L + D + U), \qquad (3.14)$$

onde *L* é a submatriz triangular inferior da matriz *A*, *D* a diagonal e *U* a triangular inferior. Substiuindo a Equação (3.14) na Equação (3.1), podemos escrever o método iterativo matricial na forma:

$$D\,x^{(k)} = b - (L+U)\;x^{(k-1)} \qquad (3.15)$$

A Equação (3.15) pode ser escrita como:

$$x_i^{(k)} = (b_i - \Sigma_{j \neq i}\, a_{ij}\, x_j^{(k-1)}) / a_{ii} \quad i=1...n,\ j = 1...n \qquad (3.16)$$

Segue, abaixo, o algoritmo deste método:

```
Escolha uma solução inicial para  x.
for k=1,2,...
    for  i=1,2,...,n
          x_i = 0
        for j=1,2...,i-1,i+1, ...n
              x_i = x_i + a_ij x_j^(k-1)
        end
        x_i = (b_i - x_i )/ a_ii
    end
    x^(k) = x
    verifique a convergência e continue se necessário.
end
```

A convergência do método é garantida quando a matriz de coeficientes é diagonalmente dominante.

O método de Jacobi executa, por iteração, *N*m* operações, onde *m* é o número médio de valores não-nulos por linha [30].

3.2.2.2 Método Gauss-Seidel

Este método é uma variação do método de Jacobi, pois é utilizado o cálculo da variável que fora calculada na mesma iteração. Na expressão de iteração temos a seguinte alteração correspondente:

$$x_i^{(k)} = (b_i - \Sigma_{j<i}\, a_{ij}\, x_j^{(k)} - \Sigma_{j>i}\, a_{ij}\, x_j^{(k-1)}) / a_{ii} \qquad (3.17)$$

Neste caso podemos, também, montar o método matricialmente, porém neste caso de aproximação, não é a diagonal, mas uma matriz triangular inferior. Reescrevendo a Equação (3.15), temos:

$$D\, x^{(k)} = b - L\, x^{(k)} - U x^{(k-1)} \qquad (3.18)$$

O algoritmo é mostrado a seguir:

```
Escolha uma solução inicial para x.
for k=1,2,...
    for i=1,2,...,n
        s = 0
        for j=1,2...,i-1
            s = s + a_ij x_j^(k)
        end
        for j=i+1, ...n
            s = s + a_ij x_j^(k-1)
        end

        x_i^(k) = (b_i - s )/ a_ii
    end
    verifique a convergência e continue se necessário.
end
```

A convergência do método Gauss-Seidel também é assegurada quando temos uma matriz diagonalmente dominante, embora esta seja uma condição apenas suficiente [31]. O número de operações é idêntico ao método iterativo de Jacobi.

Algo importante, que deve ser notado. é que este método é serial, ou seja, os cálculos dependem dos já efetuados na mesma iteração, algo contrário do método de Jacobi, onde todos seus cálculos dependem exclusivamente dos já efetuados na iteração anterior.

3.2.2.3 Método SOR ("Successive Overrelaxation")

Este método é uma variação do método de Gauss-Seidel; este método tem a idéia de uma média ponderada entre as iterações anteriores e as calculadas pelo método de Gauss-Seidel. Temos a seguinte expressão para a média ponderada:

$$x_i^{(k)} = \omega \, \underline{x}_i^{(k)} + (1 - \omega) \, x_i^{(k-1)}, \qquad (3.19)$$

onde $\underline{x}$ denota o resultado da iteração pelo método de Gauss-Jacobi e ω o fator de extrapolação. Em [32] é demonstrado que este método falha para valores de ω fora do intervalo (0,2).

O algoritmo do método SOR é dado a seguir:

```
Escolha uma solução inicial para x.
for k=1,2,...
    for i=1,2,...,n
        s = 0
        for j=1,2...,i-1
            s = s + a_ij x_j^(k)
        end
        for j=i+1, ...n
            s = s + a_ij x_j^(k-1)
        end
        s = (b_i - s )/ a_ii
        x_i^(k) = x_i^(k-1) + ω (s - x_i^(k-1) )
    end
    verifique a convergência e continue se necessário.
end
```

O número de operações do método SOR, por iteração, é de $N*(m+1)$.

O método SOR tem a finalidade de acelerar o método Gauss-Seidel, porém a dificuldade está na determinação do ω; é possível utilizar algoritmos adaptativos para minimizar esta dificuldade.

3.2.3 Algoritmos não-estacionários

Alguns métodos não-estacionários, baseados no espaço de Krylov, são: Gradiente Conjugado, Gradiente Biconjugado, Gradiente Biconjugado Estabilizado

Chamamos de espaço de Krylov o espaço gerado por:

$$K(r_0,A) = [r_0, A\,r_0, A^2\,r_0, A^3\,r_0, ..., A^{k-1}\,r_0] \qquad k = 1, 2, 3 ..., \qquad (3.20)$$

onde A é a matriz de coeficientes e r_0 é o vetor de resíduo, dado por: $r_0 = b - A*x_0$.

Os métodos baseados no espaço de Krylov são aqueles que possuem, o espaço de busca é gerado pelos vetores de (3.20).

Utilizando projeções ortogonais ou biortogonais, os métodos minimizam o resíduo, conforme Figuras 3.9 e 3.10.

Os métodos não-estacionários, baseados em espaços de Krylov, variam em relação aos espaços L, onde o resíduo é projetado, enquanto o vetor incremento solução δ pertence sempre ao espaço de Krylov usual $K(r_0,A)$.

Para o Gradiente Conjugado, temos $L = K(r_0,A)$, onde K é o espaço do vetor incremento δ e, nos métodos biconjugado e biconjugado estabilizado, L passa a ser dado por $L = K(r_0,A^t)$.

Se assumirmos $x_0 = 0$, para cada iteração k temos a solução x^k pertencendo ao subespaço gerado por $K(b,A)$ [33].

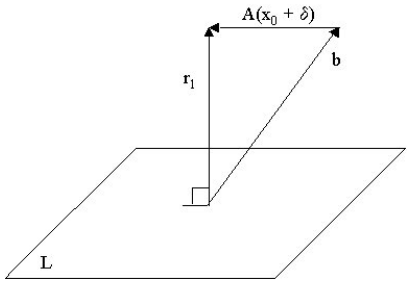


Figura 3. 9- Projeção ortogonal

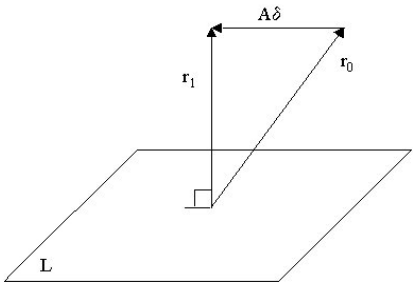


Figura 3.10 - Projeção ortogonal equivalente a (3.1)

Quando este espaço não é suficiente, para que ocorra convergência, utilizam-se os precondicionadores que o tornam mais favorável à convergência. Mais adiante trataremos os precondicionadores com mais detalhes.

3.2.3.1 Método do Gradiente Conjugado

Este método recebe este nome por gerar uma seqüência de vetores de direção de descida *A*-ortogonais ou conjugados. O resíduo das iterações são, também, os gradientes da uma função quadrática (3.21).

$$x^T Ax = x^T b. \qquad (3.21)$$

A minimização de (3.21), derivando e igualando a zero, corresponde a resolver o sistema linear. O resíduo das iterações é dado por:

$$r^{(i)} = b - Ax^{(i)}. \qquad (3.22)$$

O método do Gradiente Conjugado é aplicado, com sucesso, a sistema linear cuja matriz de coeficiente seja simétrica positiva definida. Definimos que uma matriz é definidas positiva quando $x^T Ax$ for um valor positivo, diferente de zero, para todo *x*. Deste fato é possível afirmar que a forma quadrática possui um único ponto de mínimo e todos os autovalores são positivos.

A seqüência de resíduos pode se tornar muito longa, porém apenas um pequeno número de vetores são colocados na memória.

Os valores de x são alterados em cada iteração, da forma:

$$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}. \qquad (3.23)$$

Os resíduos são alterados da seguinte maneira:

$$r^{(i)} = r^{(i-1)} + \alpha_i q^{(i)}, \qquad (3.24)$$

onde

$$q^{(i)} = Ap^{(i)}. \qquad (3.25)$$

O fator multiplicativo α_i dado por (3.26):

$$\alpha_i = r^{(i)t} r^{(i)} / p^{(i)t} Ap^{(i)}. \qquad (3.26)$$

O vetor $p^{(i)}$ é a direção na qual os resíduos são alterados:

$$p^{(i)} = r^{(i)} + \beta_{i-1} p^{(i-1)}, \qquad (3.27)$$

onde o escalar β_i é dado por:

$$\beta_i = r^{(i)t} r^{(i)} / r^{(i-1)t} r^{(i-1)}. \qquad (3.28)$$

O algoritmo precondicionado deste método é exposto a seguir:

Calcule $r^{(0)} = b - Ax^{(0)}$ *para algum vetor inicial* x^0
for $i=1,2,...$
  resolver $Mz^{(i-1)} = r^{(i-1)}$
  $\rho_{(i-1)} = r^{(i-1)t} z^{(i-1)}$
  if $i = 1$
    $p^{(1)} = z^{(0)}$
  else
    $\beta_{i-1} = \rho_{(i-1)} / \rho_{(i-2)}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
  endif
  $q^{(i)} = A p^{(i)}$
  $\alpha_i = \rho_{(i-1)} / p^{(i)t} q^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)} = r^{(i-1)} + \alpha_i q^{(i)}$

verifique a convergência e continue se necessário
end

No algoritmo, o precondiconador é dado pela matriz *M*, que ao assumir a matriz identidade e substituindo o vetor *z* pelo vetor *r*, torna-se o algoritmo simples sem precondicionamento.

Para a matriz de dimensão *N*, o método do gradiente conjugado converge, teoricamente, em *N* iterações. Porém devido a erros de arredondamentos aritméticos isto não ocorre, conforme fora mostrado em [34].

A taxa de convergência nos métodos iterativos é algo difícil de estimar [19], porém para o caso do gradiente conjugado, é possível estabelecer limites superiores:

$$\left\| x^i - x \right\|_A \leq 2\alpha^i \left\| x^0 - x \right\|_A, \qquad (3.29)$$

onde α é dado pela relação do número de condição espectral $\kappa = \lambda max / \lambda min$, onde λmax e λmin são os autovalores máximo e mínimos em módulo, da matriz $M^{-1}A$:

$$\alpha = (\sqrt{\kappa} - 1) \Big/ (\sqrt{\kappa} + 1). \qquad (3.30)$$

O gradiente conjugado precondicionado é formado por 2 produtos internos, 3 relações recorrências e 2 soluções de sistemas triangulares .

3.2.3.2 Método do Gradiente Biconjugado

No método Gradiente Biconjugado [28], [19] utiliza-se de duas seqüências de vetores ortogonais, uma com base na matriz de coeficientes A e outra com base na A^t.

Assim, teremos dois resíduos:

$$r^{(i)} = r^{(i-1)} + \alpha_i A p^{(i)} \quad \underline{r}^{(i)} = \underline{r}^{(i-1)} + \alpha_i A^t \underline{p}^{(i)}, \qquad (3.31)$$

e duas seqüências de direções de busca:

$$p^{(i)} = r^{(i)} + \beta_{i-1} p^{(i-1)} \quad \underline{p}^{(i)} = \underline{r}^{(i)} + \beta_{i-1} \underline{p}^{(i-1)} \qquad (3.32)$$

Os dois valores constantes, α_i e β_i permitem estabelecer a relação de biortogonalidade destes método:

$$\alpha_i = \underline{r}^{(i-1)t} r^{(i-1)} / \underline{p}^{(i)t} A p^{(i)} \quad \beta_i = \underline{r}^{(i)t} r^{(i)} / \underline{r}^{(i-1)t} r^{(i-1)} \qquad (3.33)$$

A relação de biortogonalidade é dada por:

$$\underline{r}^{(i)t} r^{(j)} = \underline{p}^{(i)t} A p^{(j)} = 0 \text{ se } i \neq j \qquad (3.34)$$

O algoritmo precondicionado é apresentado logo a seguir; o precondiconador é dado pela matriz *M*, que ao assumir a matriz identidade e substituindo o vetor *z* pelo vetor *r,* torna-se o algoritmo simples sem precondicionamento, da mesma maneira que fora apresentado o algoritmo do gradiente conjugado.

O algoritmo é apresentado abaixo:

Calcule $r^{(0)} = b - Ax^{(0)}$ *para algum vetor inicial* x^0

for $i=1,2,...$
  resolver $Mz^{(i-1)} = r^{(i-1)}$
  resolver $M^t\underline{z}^{(i-1)} = \underline{r}^{(i-1)}$

  $\rho_{(i-1)} = \underline{r}^{(i-1)t} z^{(i-1)}$
  if $\rho_{(i-1)} \tilde{=} 0$, *o método falhou*
  if $i = 1$
    $p^{(1)}=z^{(0)}$
    $\underline{p}^{(1)}=\underline{z}^{(0)}$
  else
    $\beta_{i-1}= \rho_{(i-1)} / \rho_{(i-2)}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1}p^{(i-1)}$
    $\underline{p}^{(i)} = \underline{z}^{(i-1)} + \beta_{i-1}\underline{p}^{(i-1)}$
  endif
  $q^{(i)} = Ap^{(i)}$
  $\underline{q}^{(i)} = A^t\underline{p}^{(i)}$
  $\alpha_i = \rho_{(i-1)} / \underline{p}^{(i)t} q^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)} = r^{(i-1)} + \alpha_i q^{(i)}$
  $\underline{r}^{(i)} = \underline{r}^{(i-1)} + \alpha_i \underline{q}^{(i)}$
verifique a convergência e continue se necessário
end

Este método é aplicável a matrizes não-simétricas; porém, a convergência deste método pode ficar comprometida quando o valor de $\underline{p}^{(i)t} q^{(i)}$ for próximo de zero e também, quando $\underline{r}^{(i-1)t} z^{(i-1)}$ for próximo de zero.

O custo computacional do método é, praticamente, o dobro do custo associado ao método do gradiente conjugado.

3.2.3.3 Método do Gradiente Biconjugado Estabilizado

O método Biconjugado estabilizado [6] é aplicado a sistemas cujas matrizes de coeficientes sejam não-simétricas, ele surge como uma variante do método do Gradiente Biconjugado Quadrático [29], [19], [6], evitando-se uma convergência irregular. O algoritmo do Gradiente Biconjugado Quadrático é dado logo abaixo.

Calcule $r^{(0)} = b - Ax^{(0)}$ *para algum vetor inicial* x^0
Escolha $\underline{r}$ *(por exemplo* $\underline{r} = r^{(0)}$*)*
for *i=1,2,...*
  $\rho_{(i-1)} = \underline{r}^t r^{(i-1)}$
  if $\rho_{(i-1)} \approx 0$*, o método falhou*
  if *i = 1*
    $u^{(1)}=r^{(0)}$
    $p^{(1)}=z^{(0)}$
  else
    $\beta_{i-1}= (\rho_{(i-1)} / \rho_{(i-2)})$
    $u^{(i)} = r^{(i-1)} + \beta_{i-1} q^{(i-1)}$
    $p^{(i)} = u^{(i)} + \beta_{i-1} (q^{(i-1)} - \beta_{(i-1)} p^{(i-1)}$
  endif
  resolver $M\underline{p} = p^{(i)}$
  $v^{(i)} = A\underline{p}^{(i)}$
  $\alpha_i = \rho_{(i-1)} / \underline{r}^t v^{(i)}$
  $q^{(i)} = u^{(i)} + \alpha_i v^{(i)}$
  resolver $M\underline{u} = u^{(i)} + q^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha_i \underline{u}$
  $\underline{q} = A\underline{u}$
  $r^{(i)} = r^{(i-1)} + \alpha_i \underline{q}$
  Verifique a convergência continue se necessário;
end

O método do gradiente biconjugado estabilizado freqüentemente possui uma convergência mais rápida que o gradiente conjugado quadrático, além da correção de instabilidades.

O algoritmo do gradiente biconjugado estabilizado é dado logo abaixo:

Calcule $r^{(0)} = b - Ax^{(0)}$ *para algum vetor inicial* x^0
Escolha $\underline{r}$ *(por exemplo* $\underline{r} = r^{(0)}$*)*
for *i=1,2,...*
  $\rho_{(i-1)} = \underline{r}^t r^{(i-1)}$
  if $\rho_{(i-1)} \approx 0$*, o método falhou*
  if *i = 1*
    $p^{(1)} = z^{(0)}$
  else
    $\beta_{i-1} = (\rho_{(i-1)} / \rho_{(i-2)})(\alpha_{(i-1)} / \omega_{(i-1)})$
    $p^{(i)} = r^{(i-1)} + \beta_{i-1}(p^{(i-1)} - \omega_{(i-1)} v^{(i-1)})$
  endif
  resolver $M\underline{p} = p^{(i)}$
  $v^{(i)} = A\underline{p}^{(i)}$
  $\alpha_i = \rho_{(i-1)} / \underline{r}^t v^{(i)}$
  $s = r^{(i-1)} + \alpha_i v^{(i)}$
verifique a norma de s; se for pequena suficiente faça
  $x^{(i)} = x^{(i-1)} + \alpha_i \underline{p}^{(i)}$ ***pare***
  resolver $M\underline{s} = s$
  $t = A\,\underline{s}$
  $\omega_i = t^t s / \underline{t}^t t$
  $x^{(i)} = x^{(i-1)} + \alpha_i \underline{p}^{(i)} \omega_i \underline{s}$
  $r^{(i)} = s + \omega_i \underline{t}$
  Verifique a convergência continue se necessário;
  Se ω_i *for* $\neq 0$ *então continue*
end

Como mostra os algoritmos (Quadrático e Estabilizado), estes métodos não necessitam da matriz transposta e possui um custo computacional semelhante. O bicomjugado estabilizado possui quatro produtos internos (dois a mais que o gradiente biconjugado e o gradiente biconjugado quadrático).

Implementaremos o algoritmo para matrizes esparsas e complexas, com precondionamento, conforme apresentaremos no Capítulo 4. Escolheu-se este algoritmo por ser uma das evoluções mais estáveis e versáteis dos métodos de Krylov, o que permitirá sua maior aplicabilidade aos problemas de propagação de ondas ópticas, onde é preciso resolver sistemas com matrizes hermitianas, matrizes não-simétricas e não-hermitianas.

3.2.4 Precondicionamento

Chamamos de precondionadores a matriz de coeficientes M tal que, quando multiplicada pela matriz A, resulta em uma matriz com melhor propriedade espectral que a matriz original A:

$$M \,.\, A = C \qquad (3.35)$$

As propriedades espectrais são fundamentais para que tenhamos métodos iterativos eficientes. É desejável que a matriz C tenha seus autovalores próximos de *1(um)* ou próximos de um valor constante.

Caso M seja exatamente a inversa de A, então C será a matriz identidade e todos autovalores serão iguais a *1(um)*. Porém o cálculo da inversa explicitamente é muito caro computacionalmente, para uma matriz esparsa; podemos ter uma inversa com muitos *fill-ins* e além do custo de processamento, a quantidade de memória exigida tornará o problema computacionalmente insolúvel.

O precondicionador M pode ser encontrado de forma fatorada, ou explícita. Como exemplo de forma fatorada, temos a decomposição LU incompleta [36] e, para a forma explícita, temos a SPAI ("Sparse Aproximate Inverse") [37]. No próximo tópico tratemos da decomposição incompleta lu: ILU(0) ("Incomplete LU 0"), MILU("Modified Incomplete LU") e ILUT("Incomplete LU Threshold").

3.2.4.1 Decomposição LU incompleta - ILU

A decomposição ILU ("Incomplete LU") consiste em introduzir modificações na eliminação de Gauss de modo que a decomposição matricial exija menor número de operações com pontos flutuantes, diminuindo, desta forma, o esforço computacional em relação à eliminação de Gauss simples.

Com controle sobre os *fill-ins*, gerados durante a eliminação de Gauss, as modificações, além de diminuir o número de operações, diminui significativamente a quantidade de memória em relação à eliminação de Gauss completa [36].

Apresentaremos um breve histórico da decomposição incompleta neste parágrafo, obtido do trabalho básico deste capítulo, Chan e Van der Vorst - Aproximate and Incomplete Factorizations [36]. A referência mais antiga, em relação a este método, foi feita por Buleev [38] e Oliphant [39], [40], O primeiro trabalho a introduzir o termo precondicionamento foi apresentado por Evans [41] que também utilizou matrizes LU esparsas como precondicionadores. Estes métodos tiveram grande interesse para pesquisadores que trabalhavam com reservas de petróleo e, em particular, Stone [42] e Dupont-Kendall-Rachford [43] propuseram métodos de fatoração para problemas elípticos e conseguiram razões de convergência semelhantes aos trabalhos anteriores. Beauwens mostrou que o trabalho de Dupont-Keldall-Rachford é exatamente o proposto por Buleev [38]. Estes métodos foram considerados como métodos de relaxação generalizados por Axelson. Meijenrich e Van der Vorst [44] consideraram esses métodos como uma Fatoração Incompleta e provaram a existência destes precondicionadores para matrizes M [9]. Kershall, em [45], apresentou testes numéricos que avaliaram o método.

Considerando que os índices (linha e coluna) permitidos pertencem ao conjunto *S*, então, definimos de maneira geral a decomposição incompleta como:

$l_{i,j} = 0$ se $i>j$ ou se a coordenada (i,j) não petencer ao conjunto S ;

$u_{i,j} = 0$ se $i<j$ ou se a coordenada (i,j) não petencer ao conjunto S.

Os índices permitidos no conjunto *S* determinarão o tipo de decomposição incompleta, bem como a eficiência do precondicionador na aplicação a cada problema específico.

3.2.4.1.1 Decomposição ILU(0)

Chamamos de decomposição ILU(0) [2], quando o conjunto *S*, anteriormente exposto, é formado pelos mesmos índices dos elementos não-nulos da matriz *A*. Assim, *S* pode ser definido como:

Se $A_{ij} \neq 0$ então o índice *(i,j)* pertence ao conjunto *S*.

Seja a matriz $M = L*U$, então temos mais uma condição:

$$m_{ij} = a_{ij}\,. \qquad (3.36)$$

```
for r = 1 :n-1
  d = 1/a_rr
  for i = r +1:n
     if (i,r) ε S then
        e = da_ir ; a_ir = e;
        for j = r+1:n
           if (i,r) ε S  and (r,j) ε S then
              a_i,j = a_i,j - e a_r,j
           end if
        end (fim de j)
     end if
  end (fim de i)
end (fim de r)
```

A decomposição ILU(0), apesar de possuir um custo computacional baixo não é muito eficiente, portanto, do ponto de vista de aplicações, não a torna atrativa.

3.2.4.1.2 Decomposição LU modificada (MLU):

A decomposição LU modificada [2] é muito semelhante à decomposição ILU(0); procedemos à decomposição com o mesmo conjunto S :

Se $A_{ij} \neq 0$ então o índice (i,j) pertence ao conjunto S;

A segunda condição é flexibilizada de maneira que não se exigirá uma igualdade elemento a elemento entre o precondicionador M e a matriz A, mas uma igualdade da soma dos elementos linha a linha:

$$\sum m_{ij} = \sum a_{ij} \qquad (3.37)$$

onde j = 1...n, para todo i

Este recurso possui justificativa em aplicações de mecânica de fluídos; mais dados sobre esses estudos estão nos trabalhos de Appleyard, Chesshire [46] e Dupont, Kendall e Rachford [43].

```
for r = 1 :n-1
  d = 1/a_{r,r}
  for i = r + 1 : n
     if (i,r) ε S then
       e = da_{ir} ; a_{ir} = e;
       for j = r + 1 : n
          if (r,j) ε S then
             if (i,r) ε S then
               a_{i, j} = a_{i, j} - e a_{r, j}
             else
               a_{i, i} = a_{i, i} - e a_{r, j}
             end if
          end if
       end (fim do laço j)
    end if
  end (fim do laço i)
end (fim do laço r)
```

3.2.4.1.3 Decomposição ILUT

Esta decomposição consiste em descartar valores muito pequenos, encontrados durante a Eliminação Gaussiana, ao mesmo tempo há um controle sobre o preenchimento da matriz. O parâmetro de referência *droptol* é definido em relação à norma da linha correspondente [2].

```
for  i = 1 : n
  w = a_{i,*}
  for k = 1, ..., i-1 and w_k ≠ 0
     w_k = w_k / a_{k,k}
     aplique regra droptol para w_k
     if w_k ≠ 0 then
        w = w - w_k * u_{k,*}
     end if
  end
  aplique a regra lfil à linha w
  l_{i,j} =  w_j    j =1, ..., i-1
  u_{i,j} = w_j     j =i, ..., n
  w = 0
end
```

A regra para descartar valores pode ser escrita como:

$$|val| < norma(a_{i,:}) * droptol, \qquad (3.38)$$

onde *droptol* é um valor constante real positivo e *val* é w_k no primeiro descarte.

No segundo descarte controla-se o preenchimento na linha *i* das matrizes triangulares L e U, tomando como *lfil* os maiores valores de cada linha *i*.

Desta forma, o parâmetro *droptol* controla a estabilidade da decomposição. E o parâmetro *lfil controla* a quantidade de memória consumida pelas matrizes *L* e *U*.

No pacote implementado neste trabalho, utilizaremos esta decomposição para auxiliar o método iterativo.

Capítulo 4

Implementação e testes

Neste capítulo apresentaremos o fluxograma que ilustra a resolução do sistema linear através da ***LIRIOS*** (subrotina implementada), os testes realizados, a estrutura do propagador utilizado para as aplicações do Capítulo 5 e nos testes.

O propagador é a ferramenta computacional que simula o movimento das ondas fotônicas em um meio; nos problemas tratados no Capítulo 5 e nos testes foram utilizados dois tipos de propagadores, um no domínio do tempo, e outro, no domínio da freqüência.

Chamamos de propagadores no domínio da freqüência quando a variação temporal é harmônica. Neste caso a propagação ocorre em uma dada coordenada espacial, por exemplo z e a discretização é realizada no plano transversal xy. A distribuição do campo sobre o plano transversal (seção reta do guia) é calculada a cada passo Δz; esta propagação é chamada de BPM ("Beam Propagation Method"). Para propagadores no domínio do tempo, a discretização é realizada na região espacial xyz e a propagação ocorre no tempo t, e a cada intervalo Δt, é verificado o estado espacial da simulação. Quando propaga-se no domínio da freqüência, temos que a geometria (seção reta) do guia pode-se alterar, enquanto, no domínio do tempo, a geometria não se altera no decorrer da propagação. Nos meios não-lineares, o índice de refração depende da intensidade do campo elétrico e, portanto, as características físicas do domínio variam, tanto no domínio do tempo como no domínio da freqüência.

Definimos como caso ***invariante*** quando lidamos com meios lineares e gometria fixa. E como caso ***variante*** quando os meios são não-lineares ou a geometria é variável. Porém, neste trabalho, o caso variante foi considerado apenas para meios não-lineares com geometria fixa.

Na Figura 4.1 temos um fluxograma esquemático de um propagador mostrando os recálculos das matrizes e vetores; o método de elementos finitos é aplicado nos recálculos das matrizes A e B; note que, para o caso invariante, não é alterada a matriz A a cada passo de propagação.

O propagador, na Figura 4.1, tem como entradas um pulso inicial e as características do meio. As matrizes A e B são montadas pelo método dos elementos finitos. Então resolve-se o sistema linear. Em seguida, no caso variante, recalcula-se as matrizes A e B para um novo passo,. Para o caso invariante recalcula-se apenas o vetor b; por fim verifica-se o critério de parada $\delta < L$ é satisfeito. A solução no passo n é dado por x_n.

A propagação em δ e com passo $\Delta\delta$, na Figura 4.1, assumem os valores z e Δz, respectivamente, quando ocorre no domínio da freqüência, enquanto que, no domínio do tempo, assumem os valores t e Δt, respectivamente. Onde z representa uma dimensão espacial e t uma dimensão temporal; L assume, da mesma forma, uma dimensão espacial máxima ou um tempo de propagação, servindo como critério de parada para o propagador.

A LIRIOS terá a finalidade de resolver o sistema linear proveniente do método de elementos finitos; utilizamos, para tal, o método iterativo biconjugado estabilizado precondicionado. Esta subrotina será inserida nos propagadores já desenvolvidos em trabalhos anteriores, realizados no DMO, cujas matrizes são fornecidas por esses propagadores.

Os tipos de armazenamentos utilizados foram: comprimidos por linhas, por coordenadas e o msr (“modified sparse row”). Todos os códigos foram implementados em Fortran 77; a escolha desta linguagem se deve ao fato de todos os propagadores disponíveis no DMO estarem nesta linguagem e, como a intenção é disponibilizar uma ferramenta mais eficiente para ser utilizada por esses propagadores, mantivemos a mesma linha de trabalho.

O sistema operacional utilizado na LIRIOS foi Linux RED HAT; o Windows 2000 foi utilizado nos testes adaptando-se aos propagadores do departamento, que são desenvolvidos no Windows. Utilizou-se nos testes uma máquina Xeon 2Ghz, biprocessada, 512 Kb de Cachê e 2Gb de memória RAM.

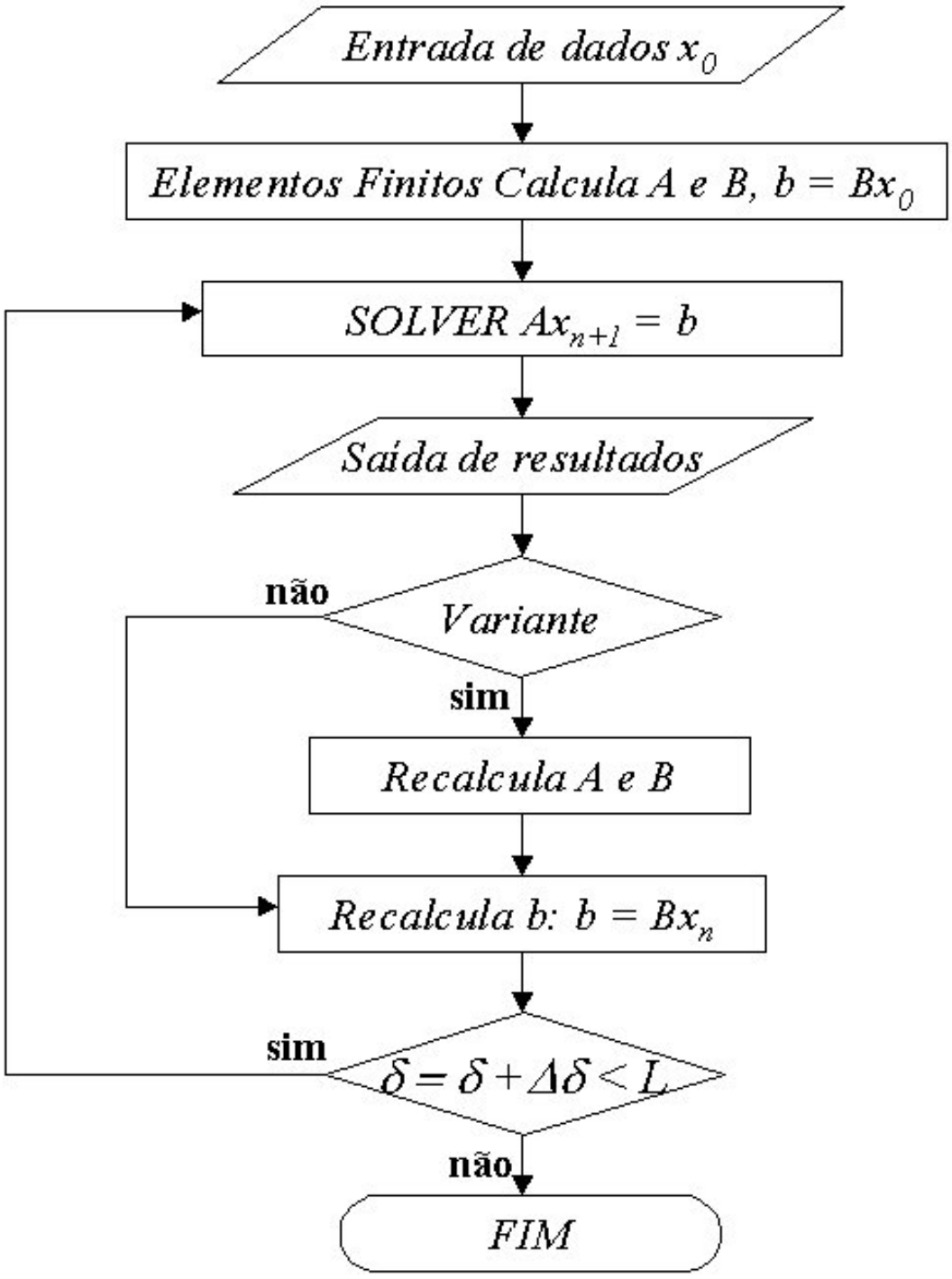


Figura 4.1 - Fluxograma do propagador

4.1 A LIRIOS

A LIRIOS (subrotina implementada) é a ferramenta proposta para solução de sistemas lineares, ela baseia-se no método do gradiente biconjugado estabilizado [6]. Para o implementarmos, de maneira eficiente, foi necessário a utilização de um precondicionador. O escolhido foi a decomposição incompleta ILUT [2], da subrotina Sparskit, disponível em [7], que foi desenvolvida por Youssef Saad. Esta subrotina, foi originalmente, desenvolvida para variáveis reais; como trabalhamos, na maioria dos casos, com variáveis complexas, a alteramos. Também adicionamos um ordenamento disponível na Sparskit, através da subrotina indset1.f; embora isto seja menos eficiente que o de grau mínimo, foi possível economizar memória nos sistemas de grandes dimensões. Este ordenamento utiliza o algoritmo de Greedy, auxiliado pela seqüência crescente dos graus do grafo correspondente; maiores detalhes deste algoritmo podem ser obtidos em [2]. Usamos esta subrotina de ordenamento como uma caixa preta e implementamos as ferramentas auxiliares de reordenamento a partir do vetor de permutação, fornecido pela indset1.f.

Como entrada de dados, na Figura 4.2, temos a matriz *A, b,* x_0, número máximo de iterações (*maxit*), resíduo relativo (*relres*), *droptol* e *lfil.*

O *relres* e *maxit* são valores utilizados nos critérios de parada de *Bicgstab*. *Droptol* e *lfil* são os parâmetros utilizados pela decomposição ILUT.

O critério de parada, do fluxograma da Figura 4.2, é dado na Figura 4.1, pelo valor de δ; onde $\delta = 0$ significa que o propagador está no primeiro passo.

O ordenamento, na Figura 4.2, é realizado apenas uma vez; esta ordenação fornece o vetor de permutação entre as linhas da matriz; nas iterações posteriores reordenamos, no início, apenas o vetor solução *x*, previamente calculado, e o vetor *b*; no final reordenamos *b* e o vetor solução *x*.

O reordenamento é a utilização do vetor de permutação para reordenar a matriz; ao reordenar as linhas e colunas, é preciso manter os vetores em ordem crescente, conforme o armazenamento de coordenadas.

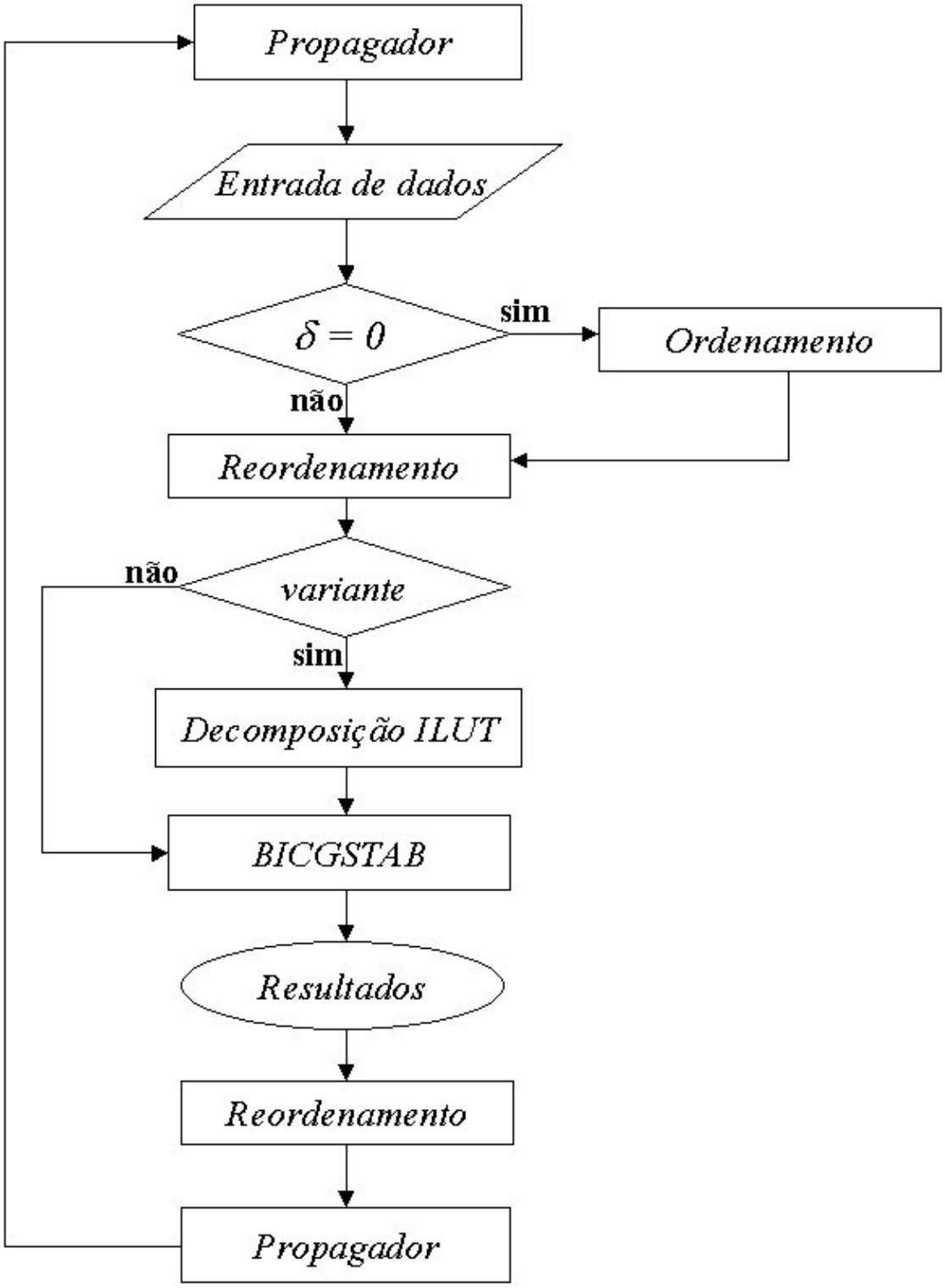


Figura 4.2 - Fluxograma da LIRIOS - geometria fixa

Na Figura 4.2 temos um fluxograma da LIRIOS. A propagação no caso invariante, conforme definimos anteriormente, ocorre quando o feixe ou pulso propagado não interage com o meio de propagação; desta forma a matriz de coeficiente se mantém a mesma, ou, ainda a geometria permanece constante, portanto é necessário fazer a decomposição ILUT apenas uma vez. A LIRIOS, apresentada no fluxograma da Figura 4.2, é inserida no estágio dado por *Solver Ax = b*, da Figura 4.1.

Para as aplicações cujas matrizes são complexas, os bons desempenhos, apresentados nos resultados do Capítulo 5, foram obtidos quando aplicamos as operações com o chamado Produto Simétrico, i.e. um produto sem aplicar o complexo conjugado do vetor transposto. Assim o produto entre dois vetores complexos, *a* e *b,* foi definido da seguinte maneira:

$$prod = \sum_{i=1}^{n} a_i b_i \ . \qquad (4.1)$$

Desta forma, para o caso complexo, não estamos mais trabalhando com um produto interno convencional, com o qual a convergência não era obtida ou, ainda, o desempenho era muito ineficiente. Este fato é determinante na implementação, para o sucesso da LIRIOS. As justificativas desta observação serão objeto de estudo em nossos possíveis trabalhos futuros, pois, no momento, estamos mais interessados nas aplicações do método.

O critério de parada escolhido, no método iterativo, foi a norma do resíduo relativo dado por:

$$resrel = | b - Ax^i | / |b|, \qquad (4.2)$$

onde x^i é a solução obtida na iteração *i*, no método iterativo.

4.2 Testes

Inicialmente, faremos alguns testes com matrizes provenientes de problemas práticos, sem nos prendermos a resultados, pois pretende-se, neste primeiro momento, fazer apenas uma avaliação da escalabilidade e confrontar a LIRIOS com a ME28.

A escalabilidade é uma maneira muito importante, para podermos afirmar quando um método numérico é aplicável a problemas grandes e qual é o custo desta aplicação.

Fazendo o gráfico de tempo de solução por dimensão da matriz, podemos ter uma idéia geral do desempenho do método de solução que chamamos de escalabilidade. Um processo de solução ideal é aquele que possui uma escalabilidade linear, isto é o tempo de solução é proporcional à dimensão da matriz.

Assim comparando a escalabilidade entre a subrotina ME28 e a LIRIOS, teremos uma idéia do comportamento destas ferramentas, para os diversos tipos de aplicações.

Nos testes que se seguem, chamamos de *caso invariante* quando a propagação da onda não altera as propriedades físicas do meio. Desta forma, quando utiliza-se a ME28, não será preciso executar a decomposição LU, a cada passo da propagação, para o método iterativo com decomposição LU incompleta (ILU). O tempo considerado, no caso invariante, na rotina implementada é apenas o tempo de solução do sistema em cada passo da propagação, pois o tempo da decomposição incompleta se diluiria, caso estivéssemos em uma propagação.

Para o caso variante, executa-se a decomposição LU com a subrotina ME28, ou a decomposição LU incompleta, a cada passo de propagação, pois a influência da onda, no meio, implica em um novo sistema de equações a ser resolvido.

Os testes realizados na LIRIOS foram utilizados a combinação dos valores dos parâmetros *lfil* e *droptol*, expostos na Tabela 5.1.

lfil	5	10	15	20	50	100	150	N
droptol	1e-1	1e-2	1e-3	1e-4	1e-5	1e-6	1e-7	1e-8

Tabela 4.2 - Parâmetros de ILUT

Assim teremos 64 resultados para cada matriz; como avaliamos, para cada, problema em torno de 10 matrizes de dimensões distintas, então os resultados apresentados serão apenas daqueles parâmetros que obtiveram melhor desempenho. No Apêndice A temos todas as tabelas, com os melhores desempenhos referentes aos gráficos da LIRIOS.

4.1.1 Caixa condutora 2D

A matriz trabalhada nesta seção foi obtida da ferramenta *Meftool*, desenvolvida como uma "toolbox" de elementos finitos, do Matlab, em uma tese de mestrado [50], desenvolvida no DMO. Utilizamos um problema de eletrostática para o cálculo de potencial em uma caixa condutora conforme Figura 4.3. Este problema é regido pela equação de Laplace [51] que, quando aplica-se elementos finitos para sua solução, gera-se uma matriz de coeficientes reais. Os valores dos potenciais são dados ao redor da caixa ma Figura 4.3.

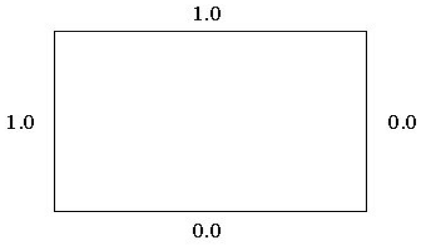


Figura 4.3 -Caixa oca condutora e condições de contorno

A malha deste problema foi feita no software comercial Grid and Data (GID) [52]. A matriz de coeficientes obtida é real e simétrica definida positiva.

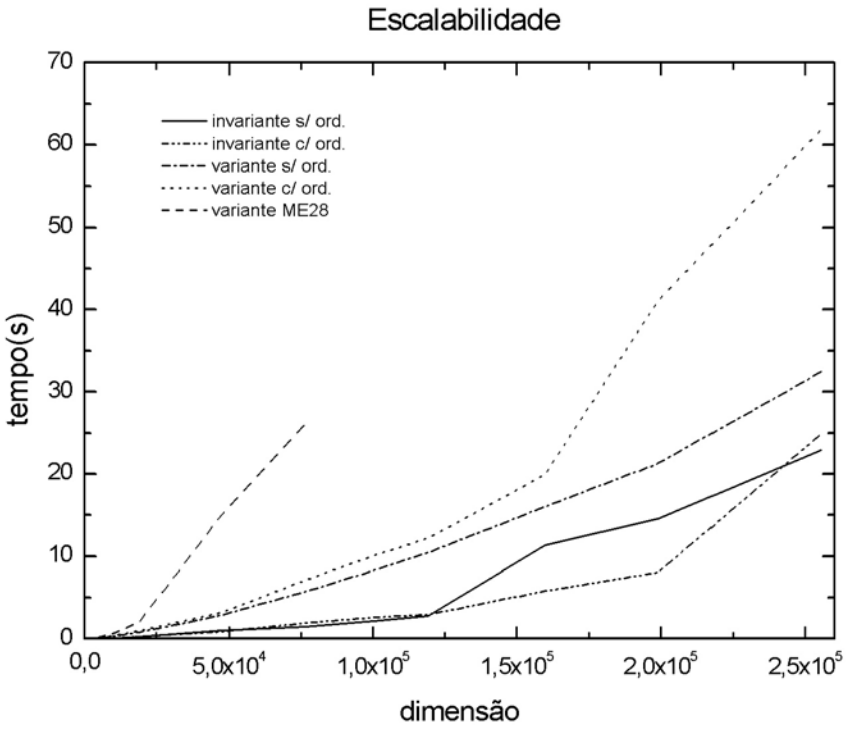


Figura 4.4 - Escalabilidade das matrizes – caixa condutora 2D

Na Figura 4.4 vemos que a ME28 não conseguiu resolver sistemas de dimensão além de 75.000 variáveis, enquanto a LIRIOS chegou a 250.000, com uma boa vantagem em relação ao desempenho. Para as matrizes ordenadas, temos uma perda no desempenho, porém ocorre um ganho em relação ao espaço de memória ocupada, pois gera menos preenchimento na decomposição ILU.

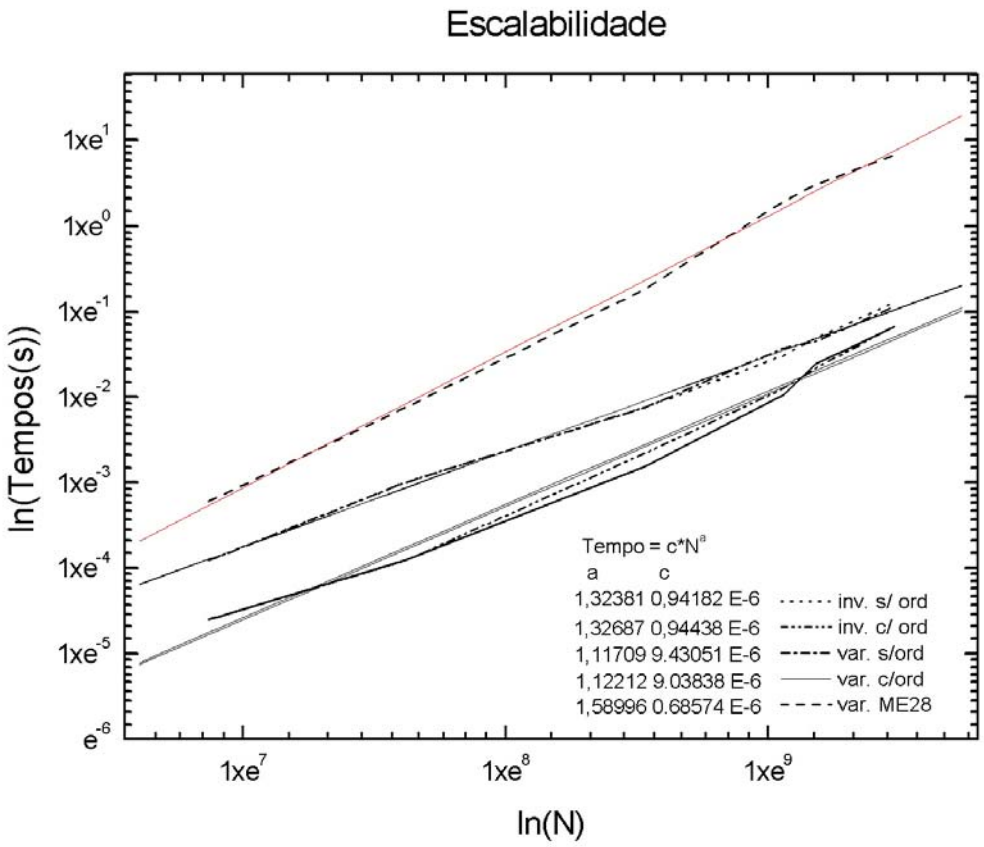


Figura 4.5 - Escalabilidade, linearizando a Figura 4.4

Neste caso, a vantagem da LIRIOS ficou mais evidente pela sua capacidade de solução de sistemas maiores e, também, por possuir uma boa escalabilidade, conforme mostram os coeficientes da Figura 4.5.

Foi supoosto que as curvas da Figura 4.5 são regidas pela expressão (4.3).

$$tempo = c*N^a \,, \qquad (4.3)$$

onde N é a dimensão da matriz , a e c são constantes a serem determinadas.

Encontramos as constantes e as retas expostas na Figura 4.5 traçando as curvas da Figura 4.4 em escala logarítmica e ajustando-as às retas correspondentes, utilizando o método dos mínimos quadrados, chamamos este processo de *linearização*. Notamos que a ME28 possui um coeficiente *a* maior que a rotina implementada, o que explica o melhor desempenho da LIRIOS.

Para o caso invariante, faremos uma outra análise, pois a decomposição LU da ME28 é mais lenta; porém a solução do sistema triangular, em cada passo, é mais rápida que na LIRIOS.

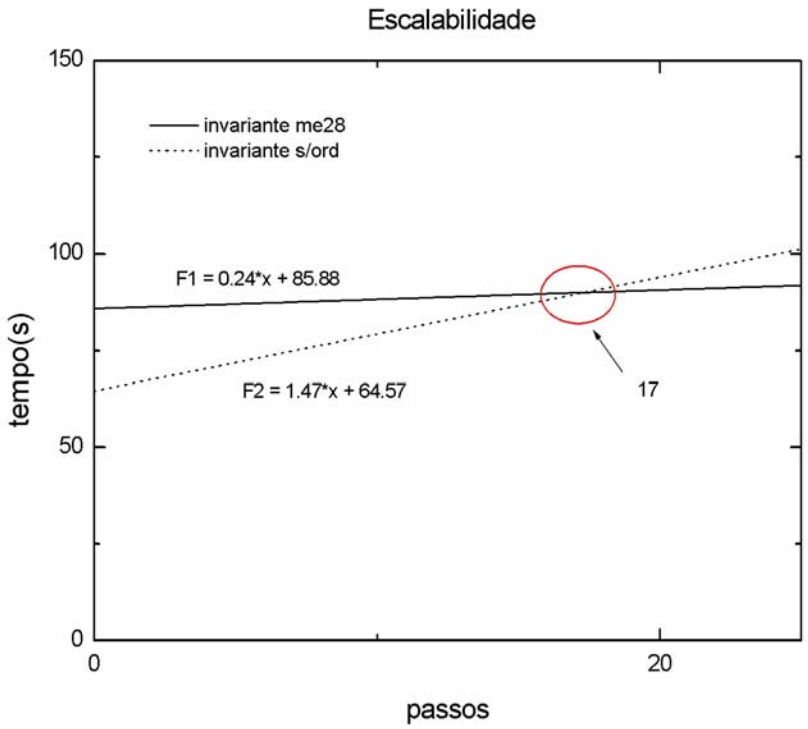


Figura 4.6 - Escalabilidade, em função do passo, para o melhor caso – caixa condutora 2D

Consideramos, na Figura 4.6, apenas o desempenho na matriz de maior dimensão (75.000) que a ME28 foi capaz de resolver. Para a ME28 obtivemos os tempos de 0,24 s, na solução do sistema triangular, e 85,88 s na decomposição LU; na LIRIOS o tempo de solução foi de 1,47 s e a decomposição incompleta (ILUT) demorou 64,57 s. A partir destes dados, fez-se o gráfico da Figura 4.6.

A seguir, montamos o sistema (4.4), onde x o número de passos e $F1$ e $F2$ as expressões de desempenho da ME28 e da LIRIOS, respectivamente. Resolvendo o sistema, podemos estimar que a partir de 17 passos é mais vantajoso utilizar a ME28, em comparação com o BICGSTAB sem ordenamento.

$$\begin{cases} F1 = 0{,}24 * x + 85{,}88 \\ F2 = 1{,}47 * x + 64{,}57, \end{cases} \qquad (4.4)$$

onde x representa o número de passos.

Na Figura 4.7 apresentamos a esparsidade da matriz, nesta figura podemos vizualizar como os valores não-nulos estão distribuídos. A matriz considerada é a de maior dimensão(250.000).

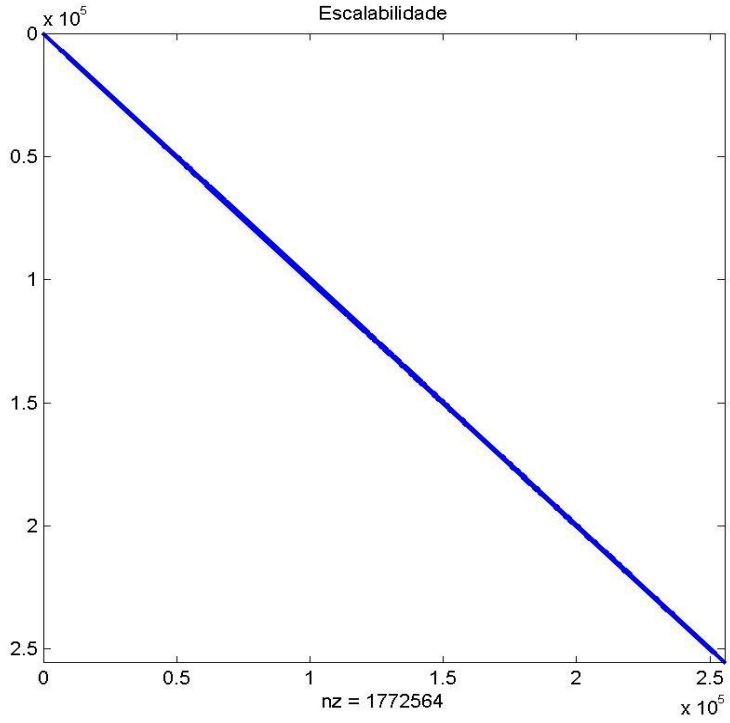


Figura 4.7 - Esparsidade da matriz – caixa condutora 2D

4.1.2 Guia de onda com formulação vetorial e PML

O guia de onda canal, da Figura 4.8, possui 2μm de lado e índice de refração de 1,5, o domínio computacional é um quadrado de 20 μm com uma PML("Perfect Matched Layer") [48] de 1μm de espessura ao redor. PML é um recurso para absorção da onda, que permite uma análise apenas da região de interesse, diminuindo a janela computacional do problema.

Na Figura 4.8 temos a seção reta do guia no plano xy, a direção de propagação está na direção z.

O BPM, utilizado neste problema, foi inicialmente tratado em uma tese de doutorado [49] trabalhando com formulação vetorial; porém, no caso aqui apresentado, possui a camada adicional de PML. Neste problema, a matriz utilizada é não-simétrica e não-hermitiana.

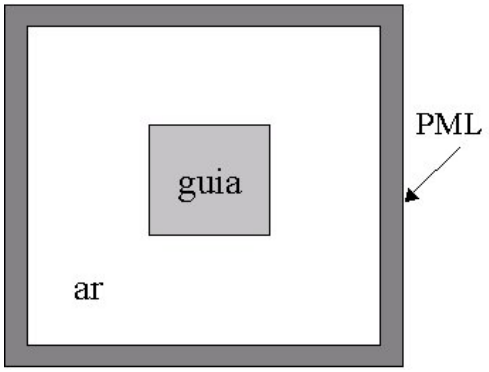


Figura 4.8 - Seção reta do guia canal, rodeado por PML

Na Figura 4.9 apresentamos a escalabilidade para este problema.

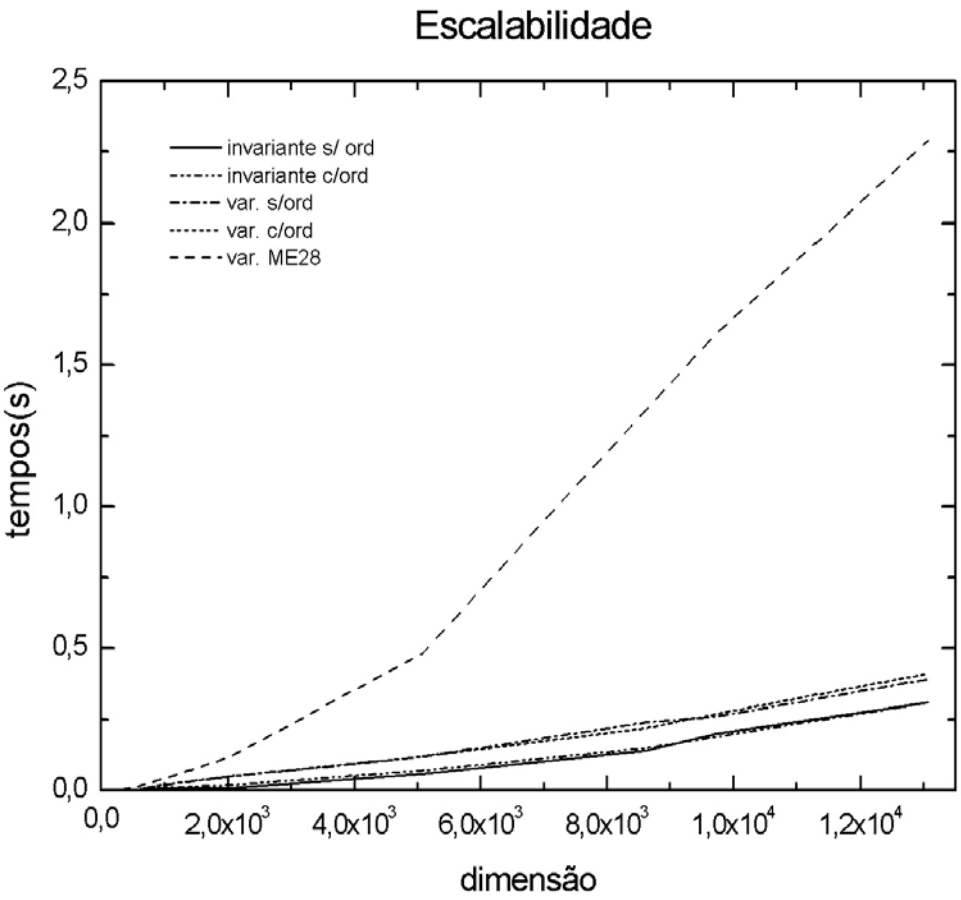


Figura 4.9 - Escalabilidade - guia canal

Notamos, na Figura 4.9, que a LIRIOS é muito eficiente possuindo uma escalabilidade praticamente linear, enquanto a subrotina ME28 possui uma escalabilidade bem maior.

Na Figura 4.9, as matrizes ordenadas e não-ordenadas possuem pouca diferença de desempenho. Isto pode ser melhor visto na Figura 4.10.

Aplicamos, na Figura 4.9, o mesmo processo de linearização realizado para a Figura 4.5. O resultado desta linearização é mostrado na Figura 4.10.

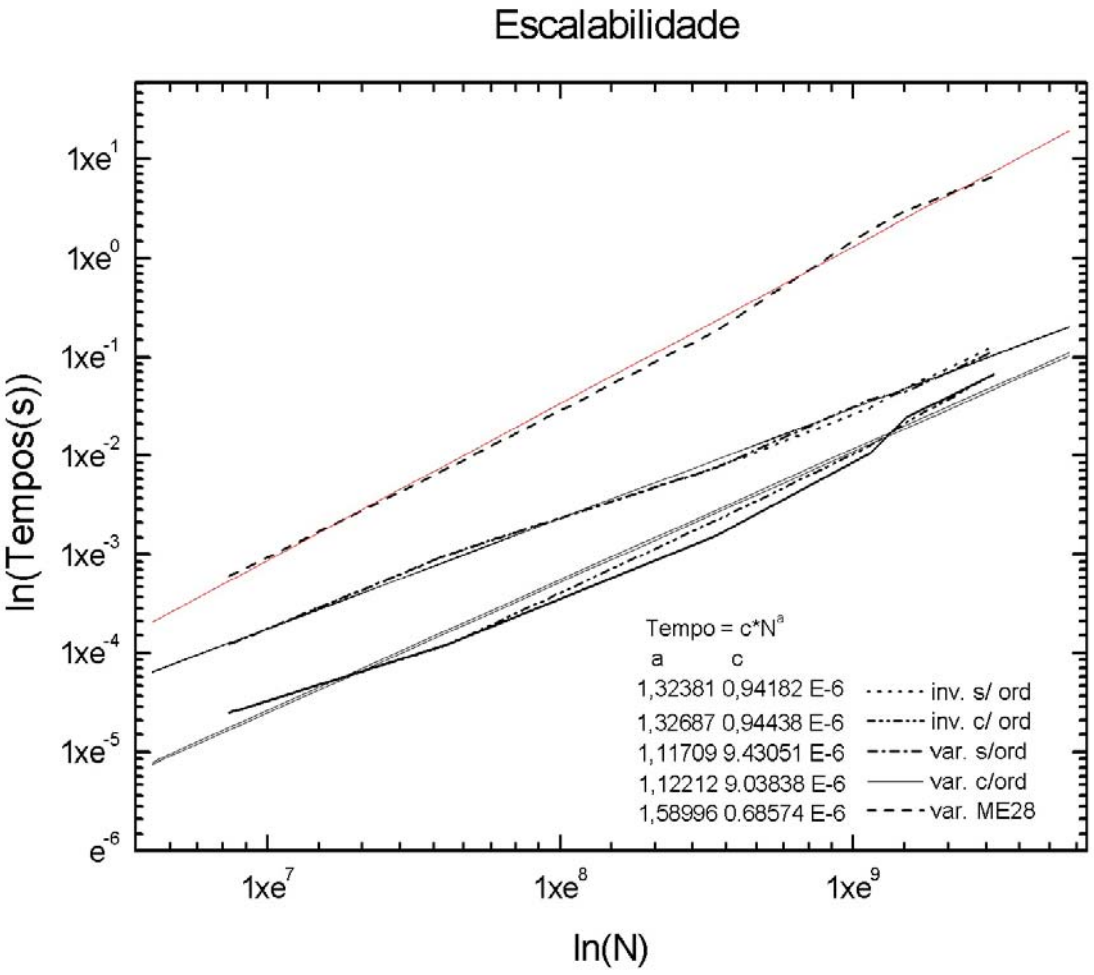


Figura 4.10 - Escalabilidade, linearizando a Figura 4.9

Notamos que o coeficiente angular da ME28, na Figura 4.10, é maior que o da LIRIOS, para o caso variante. Assim a LIRIOS demonstra ter uma melhor eficiência em relação à ME28.

Para o caso invariante, da ME28, utilizaremos a matriz de maior dimensão; obtivemos, os tempos de 0,04 s, na solução do sistema triangular, e de 5,88 s na decomposição LU; na LIRIOS, o tempo de solução foi de 0,31s e decomposição incompleta de 0,08 s. O sistema do desempenho, é dado por (4.5); resolvendo-o, verificamos que será mais vantajoso o uso da ME28 a partir de 21 passos. No sistema (4.5), *F1* e *F2* são as expressões de desempenho da ME28 e da LIRIOS, respectivamente.

$$\begin{cases} F1 = 0{,}04 * x + 5{,}88 \\ F2 = 0{,}31 * x + 0{,}08. \end{cases} \qquad (4.5)$$

Podemos vizualizar na Figura 4.11 o sistema (4.5), avaliando de maneira mais qualitativamente o comportamento das subrotinas avaliadas.

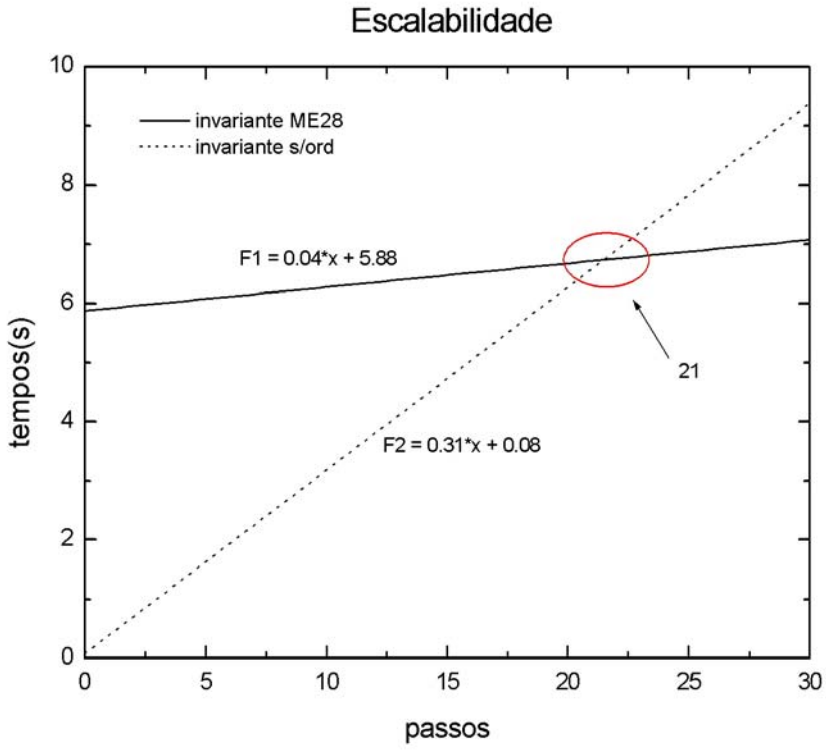


Figura 4.11 - Escalabilidade, em função do passo, para o melhor caso - guia canal

Na Figura 4.12 temos a esparsidade da matriz, podemos notar que os valores não-nulos distanciam bastante da diagonal.

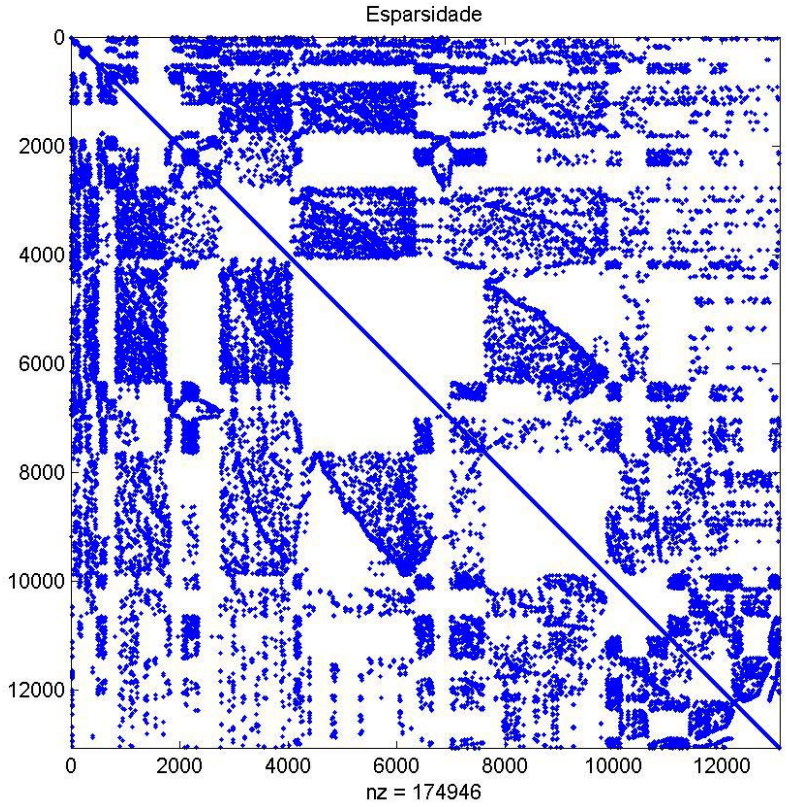


Figura 4.12 - Esparsidade da matriz – guia canal

O malhador utilizado, neste problema, foi o malhador chamado HM (Hugo Mesh), desenvolvido pelo DMO.

4.1.3 Guia PBG

Este problema consiste em propagar uma onda no ar através de um guia formado por uma estrutura PBG ("Photonic Band Gap"). A propagação foi feita no domínio do tempo [53], [54]. Esses guias são formados por cristais fotônicos.

Na Figura 4.13 temos o guia PBG, construído pela retirada dos cristais fotônicos no sentido longitudinal da figura; o guia foi rodeado por PML; os índices de refração são apresentados ao lado da figura.

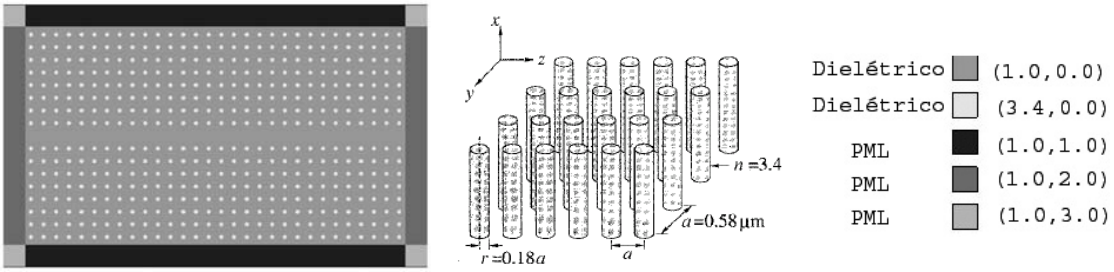


Figura 4.13 - Guia PBG e detalhes

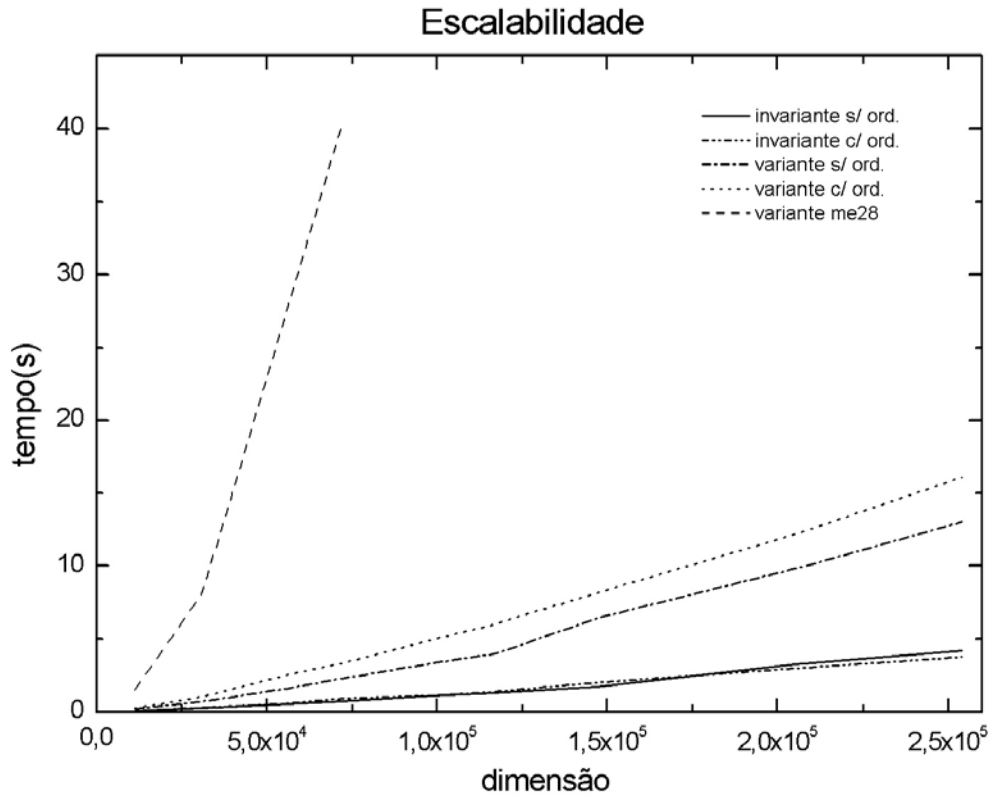


Figura 4.14 - Escalabilidade das matrizes – Guia PBG

Na Figura 4.14 temos a escalabilidade do problema, podemos observar que o método implementado é muito mais eficiente, em se tratando de sistemas lineares grandes, tanto no caso invariante como no caso variante.

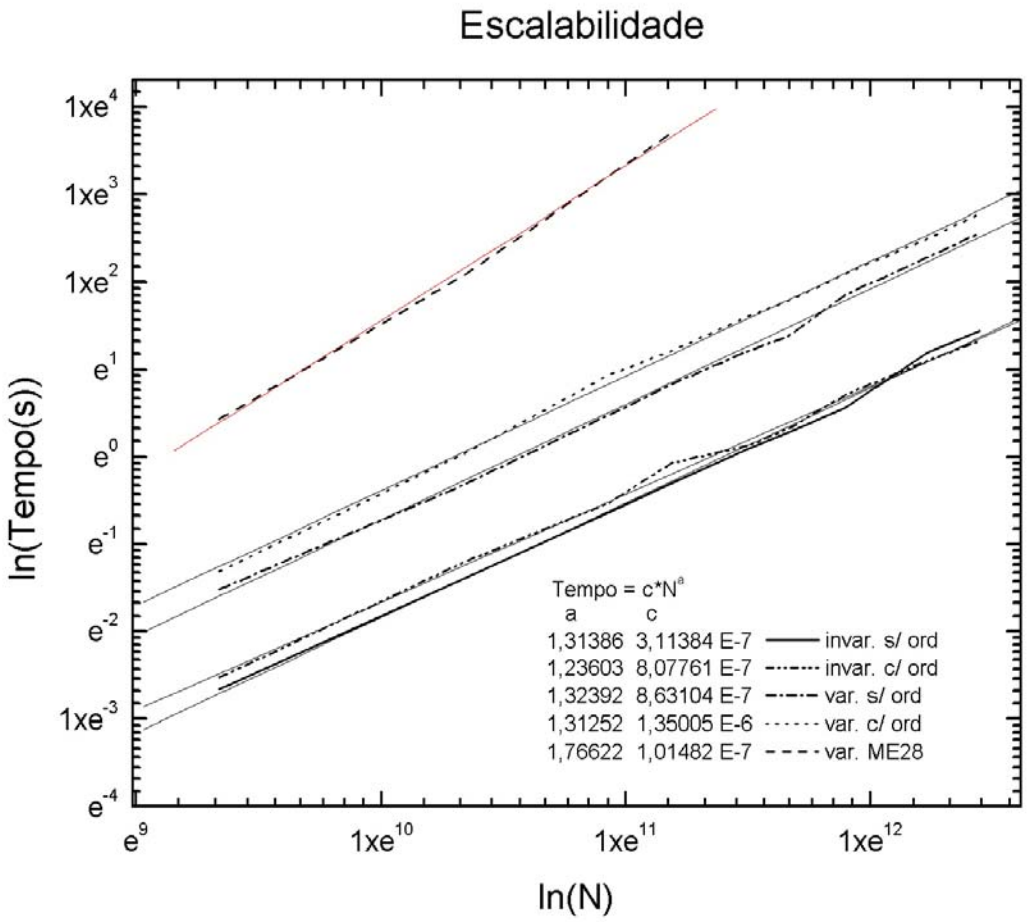


Figura 4.15 - Escalabilidade, linearizando a Figura 4.14

Os coeficientes, mostrados na Figura 4.15, mostram que a ME28 possui um valor de *a* maior que na LIRIOS, justificando a eficiência vista na Figura 4.14, notamos, também, que houve pouca variação dos coeficientes, para os casos invariante e variante. Algo importante a destacar é que o ordenamento adicionou, em alguns casos um custo computacional sensivelmente maior, mas este custo permaneceu constante, como notado pelo paralelismo das retas da Figura 4.15

Para o caso invariante, podemos estimar, resolvendo o sistema de desempenho (4.6), quando um método é melhor que outro, de forma semelhante a que apresentamos na Figura 4.6. No sistema (4.6), *F1* e *F2* são as expressões de desempenho da ME28 e da LIRIOS, respectivamente.

$$\begin{cases} F1 = 0{,}21 * x + 397{,}54 \\ F2 = 14{,}15 * x + 0{,}74, \end{cases} \qquad (4.6)$$

onde x é o número de passos.

Na Figura 4.16, verifica-se que a subrotina ME28 é mais eficiente acima de 723 passos. Novamente consideramos, para este cálculo, o maior sistema de equações que a ME28 foi capaz de resolver: 75.000 variáveis.

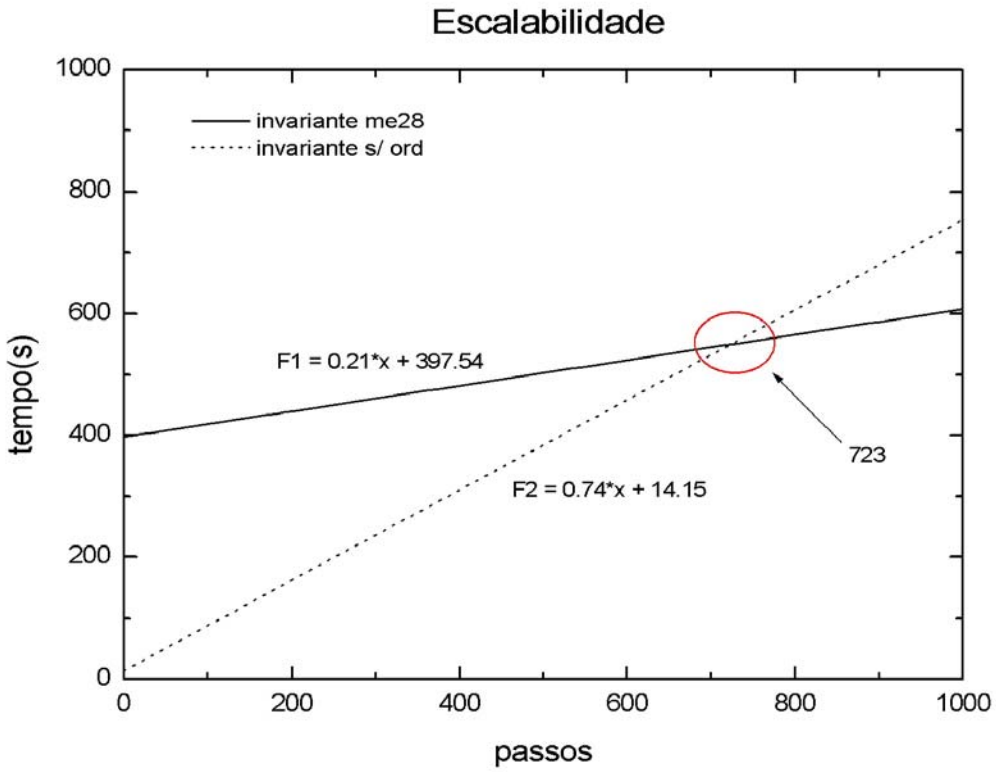


Figura 4.16 - Escalabilidade, em função do passo, para o melhor caso – guia PBG

A matriz de coeficientes, deste tipo de problema, é hermitiana. A malha foi feita utilizando o GID.

Na Figura 4.17 temos a esparsidade deste tipo de problema, para a matriz de dimensão 250.000, notamos novamente que temos uma esparsidade concentrada ao redor da diagonal. Este fato é uma característica proporcionada pelo malhador GID.

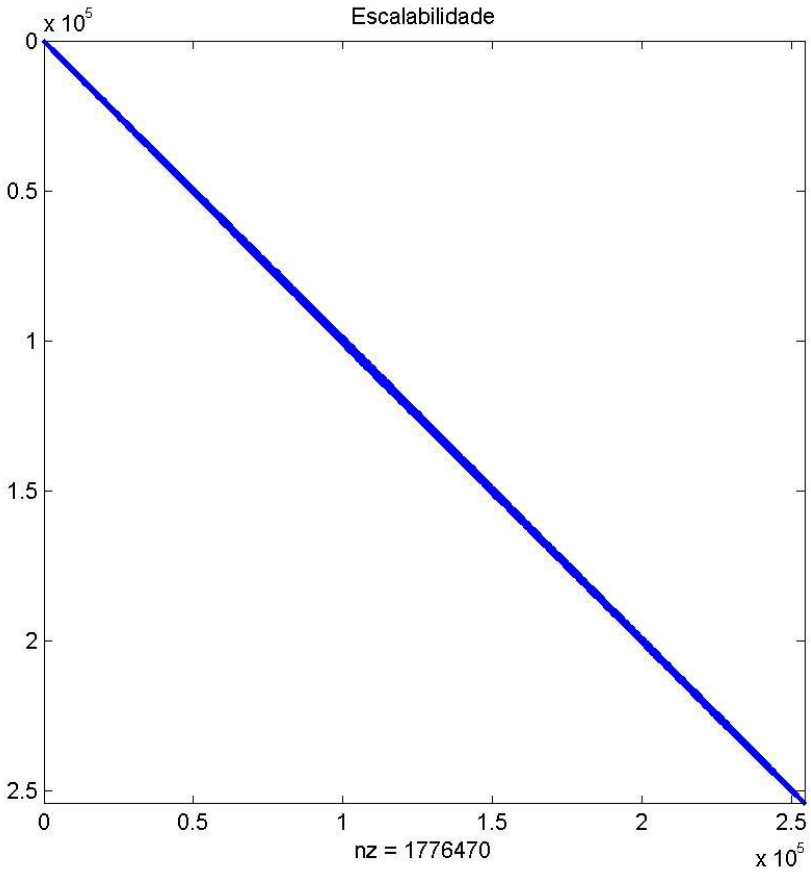


Figura 4.17 - Esparsidade da matriz - guia canal

Capítulo 5

Resultados

Neste capítulo mostraremos os resultados obtidos com a ***LIRIOS*** (subrotina implementada) comparando-a com a ME28 [5] e ao Matlab [47]. Tanto a ME28, como o Matlab, foram escolhidos por serem as ferramentas de solução de sistemas lineares utilizadas, atualmente, no DMO/FEEC.

O software Matlab dispõe de duas funções, que serão utilizadas: LUINC, que executa a decomposição LU incompleta [2] e a BICGSTAB [6], que soluciona o sistema pelo método do gradiente biconjugado estabilizado.

Nas aplicações, pretende-se validar o método e verificar o comportamento da LIRIOS, em problemas em que emprega-se a ME28 ou o Matlab.

Também apresentaremos um estudo sobre condicionamento da matriz, com o qual pretende-se estabelecer relações que permitam verificar a evolução do condicionamento em função da dimensão das matriz. Esta análise será útil para sistemas de grande porte.

Utilizou-se nas aplicações e nas análise para sistemas de grande porte uma máquina Xeon 2Ghz, biprocessada, 512 Kb de Cache e 2Gb de memória RAM.

5.1 Aplicações

Apresentamos nas aplicações, inicialmente, a propagação de um feixe gaussiano propagando no espaço livre. Neste caso é possível encontrar a solução analítica do problema; por esse motivo utilizaremos este problema para validar a LIRIOS.

Em seguida, trataremos de três aplicações que foram objeto de estudo em teses de doutorado no DMO [53], [55]. Os exemplos de [55] referem-se a um acoplador direcional, formado por duas fibras cilíndricas, e um guia de onda anisotrópico. Nestas aplicações, poderemos verificar, novamente, a convergência do método iterativo implementado e sua eficiência, em relação à subrotina ME28.

Finalizando as aplicações, trabalharemos com sistemas maiores, como na propagação em uma junção em ângulo reto, baseada em PBG (guia curvo), de um trabalho de doutorado [53]. Neste caso, a malha será refinada até 243.608 nós, que permitirá uma maior clareza do comportamento da propagação da onda eletromagnética.

5.1.1 Feixe Gaussiano no espaço livre

Consideramos um feixe propagando-se no espaço livre, na direção z (BPM), com domínio transversal quadrado (x, y), com dimensão de 24 μm.

A equação diferencial que descreve a difração de um pacote de ondas [56] é dada por (5.1).

$$\frac{\partial Aa}{\partial z} - \frac{i}{2k_0}\nabla_1^2 Aa = 0 \qquad (5.1)$$

Resolveremos a expressão (5.1) acima para:

$$\nabla_1^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}, \qquad (5.2)$$

Obtendo a equação (5.3) que nos permite calcular a dispersão em qualquer ponto (x,y,z).

$$Aa(x,y,z) = \frac{1}{1+\frac{2iz}{k_0 w_0^2}} \exp\left(-\frac{x^2+y^2}{w_0^2+\frac{2iz}{k_0}}\right), \qquad (5.3)$$

onde:

$$k_0 = 2\pi/\lambda_0 \qquad \lambda_0 = 1\,\mu\text{m}, \qquad (5.4)$$

Da equação (5.3), calcularemos os valores no ponto x = 0, y = 0, um feixe com largura espacial de w_0 = 2,5 μm, para as distâncias de propagação z = 6,8 μm, z = 17 μm, z = 34 μm e compararemos com os valores obtidos com a simulação numérica. Na Figura 5.16 temos o feixe em $z = 0\mu$m.

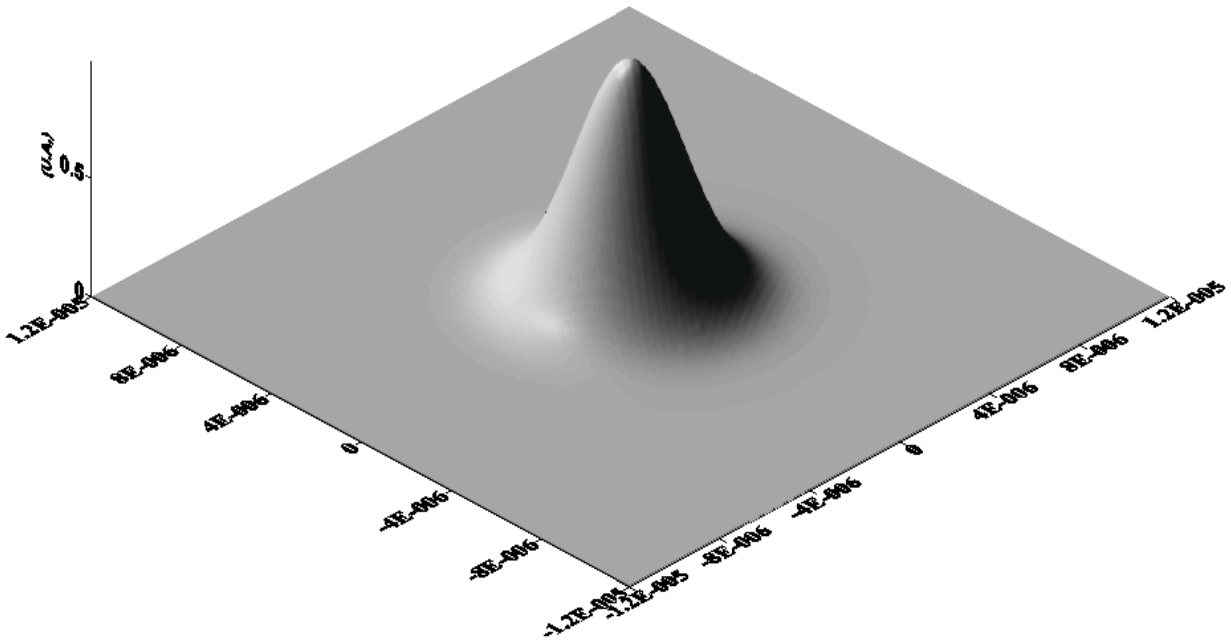


Figura 5.1- Feixe gaussiano inicial

A Tabela 5.1 temos a comparação entre solução analítica e o resultado da LIRIOS, nos pontos $x = 0$ e $y = 0$, para a matriz de dimensão 7.120 com 49.166 valores não nulos.

z	**6,8 μm**	**17 μm**	**34 μm**
solução analítica	0,9449371926	0,7560121886	0,5000963666
solução numérica	0,9402911228	0,7495689042	0,4968986253

Tabela 5.1 - Valores obtidos com passo de 0,0025 μm – dimensão 7e3

A Figura 5.2 ilustra os resultados numéricos ao longo da propagação.

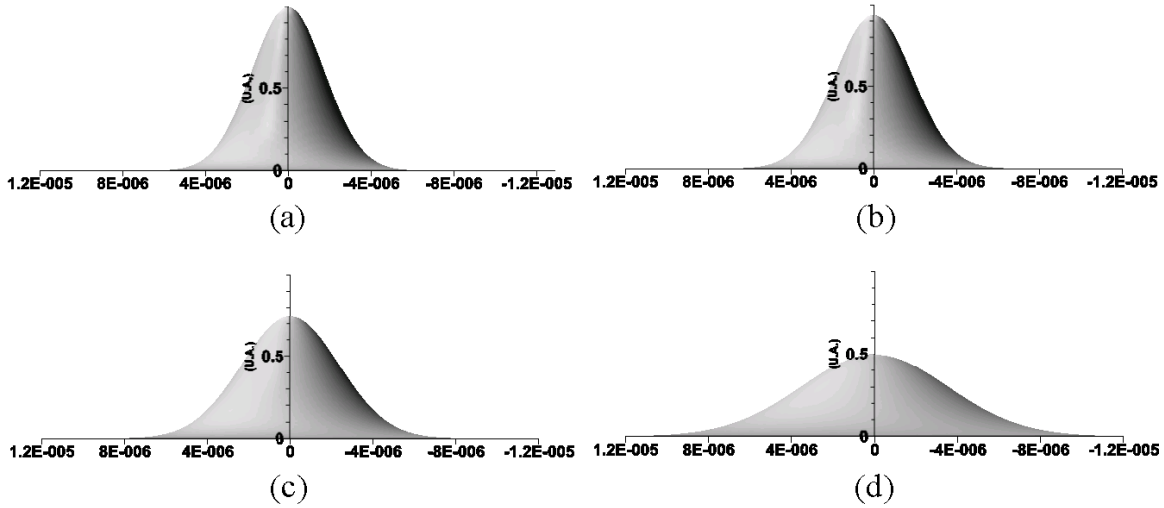


Figura 5.2 - Feixe gaussiano para passo de 0,0025μm , malha de dimensão 7e3, vista lateral em (a) $z = 0$ μm, (b) $z = 6,8$ μm, (c) $z = 17$ μm, (d) $z = 34$ μm

Utilizamos os seguintes parâmetros, para a decomposição incompleta: *lfil* = 7.120 e *droptol* = 1e-6, a tolerância no método iterativo foi de 1e-14, o tempo médio de solução para cada passo foi de 0,029 s no Linux e 0,04 s no Windows.

Podemos afirmar, para este sistema, através da Tabela 5.2, que a LIRIOS é válida. Mais adiante, faremos esta mesma avaliação para matrizes de dimensões maiores.

A Tabela 5.3 mostra os resultados para a matriz de dimensão 66.535 com 457.051 valores não-nulos, nos pontos $x = 0$ e $y = 0$; os resultados permaneceram confiáveis:

z	6,8 μm	17 μm	34 μm
solução analítica	0,9449371926	0,7560121886	0,5000963666
solução numérica	0,9429773228	0,7525675675	0,4983894364

Tabela 5.2 - Valores obtidas com passo de 0,0025 μm – dimensão 6e4

A Figura 5.3 mostra a difração do feixe gaussiano, para uma malha de 66.535 nós, obtida pelo método numérico.

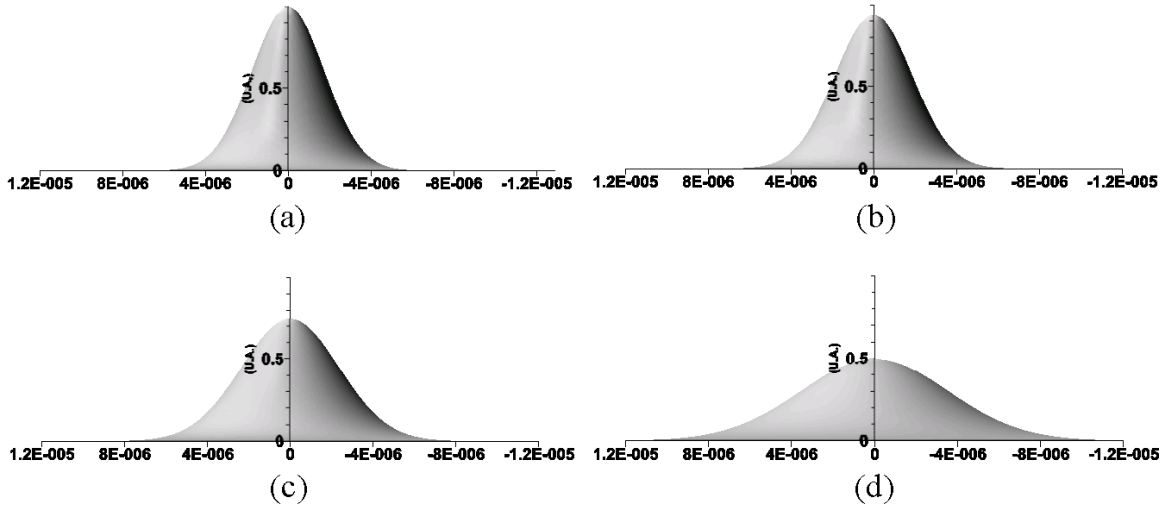


Figura 5.3 - Feixe gaussiano para passo de 0,0025μm , malha de dimensão 6e4; vista lateral em (a) $z = 0$ μm (b) $z = 6,8$ μm, (c) $z = 17$ μm, (d) $z = 34$ μm

Os parâmetros utilizados na decomposição incompleta foram: *lfil* = 100 e *droptol* = 1e-5, a tolerância no método iterativo foi de 1e-14, o tempo médio de solução para cada passo foi de 3,368 s, no ambiente Linux, e 4,394 s no Windows.

Na Tabela 5.3 temos os resultados para o sistema de dimensão 103.892 com 716.364 valores não-nulos, nos pontos $x = 0$ e $y = 0$; notamos que os resultados permaneceram consistentes com a solução analítica:

z	6,8 μm	17 μm	34 μm
solução analítica	0,9449371926	0,7560121886	0,5000963666
solução numérica	0,9431124996	0,7527087765	0,4984558401

Tabela 5.3 -Valores obtidas com passo de 0,0025 μm – dimensão 1e5

A Figura 5.4 mostra a difração do feixe gaussiano, obtida pelo método numérico.

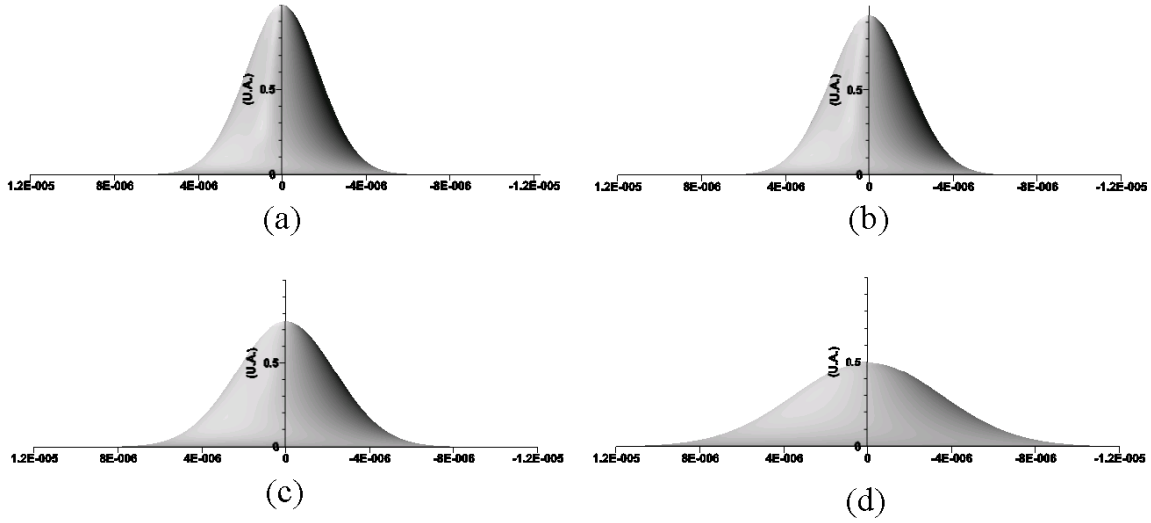


Figura 5.4 - Feixe gaussiano para passo de 0,0025μm , malha de dimensão 1e5, vista lateral em (a) z = 0μm (b) z = 6,8 μm, (c) z = 17 μm, (d) z = 34μm

Os parâmetros utilizados na decomposição incompleta foram: *lfil* = 150 e *droptol* = 1e-5, o tempo médio de solução para cada passo, no Linux, foi de 4,938 s e, no Windows, 8,447 s, a tolerância no método iterativo foi de 1e-14.

Nos três sistemas avaliados, a pequena diferença notada está mais relacionada com o método de discretização do que com a solução do sistema linear. Com essa ressalva, podemos afirmar que houve concordância com os resultados analíticos, o que valida a LIRIOS. Nas próximas aplicações, continuaremos a confrontar resultados, mas apenas com soluções numéricas.

5.1.2 Acoplador direcional

Neste caso resolveremos o sistema linear obtido da formulação de um acoplador abordado em tese de doutorado [55]; o método de propagação, deste caso, é o BPM com formulação vetorial.

O acoplador é formado por duas fibras de índices n_2 = 1,6 e a casca possui um índice de n_1 = 1,5 e uma distância d = 0,2 μm. O acoplador consiste na passagem do pulso de uma fibra para a outra, neste caso o pulso inicial é aplicado na fibra da esquerda, a medida que ocorre a propagação o pulso passará para a fibra da direita.

Este tipo de estrutura é aplicada em óptica integrada, como por exemplo: chaveamentos, divisão de potência, modulação, etc [55].

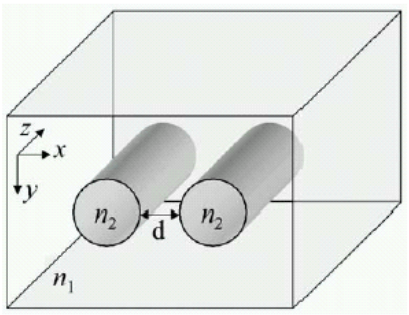


Figura 5.5 - Acoplador direcional [55]

O desempenho da ME28 e da LIRIOS é dado pela Tabela 5.4. Notamos que a LIRIOS não obteve resultados expressivos nesta aplicação, em termos de desempenho, mas quanto aos resultados, a LIRIOS trabalhou muito bem.

LIRIOS	0,0620 s
ME28	0,0074 s

Tabela 5.4 - Comparação entre rotinas - acoplador direcional

Na LIRIOS, utilizamos os seguintes parâmetros *lfil*=20, *droptol* = 1e-3 e uma tolerância de 5e-15; os testes foram realizados no ambiente Windows.

O malhador utilizado, neste caso, foi o GID. A esparsidade da matriz é mostrada na Figura 5.6 notamos que há uma concentração dos valores não-nulos ao redor da diagonal da matriz.

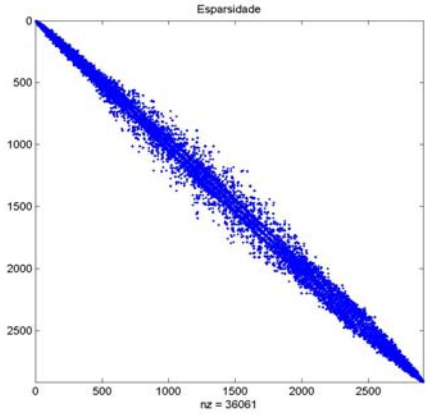


Figura 5.6 - Esparsidade da matriz - acoplador direcional

A matriz do sistema em questão é não-simétrica, não-hermitiana, e com uma dimensão de 2.914 variáveis e com 36.061 valores não-nulos.

As soluções obtidas pela LIRIOS são condizentes com aquelas obtidas em [55], com a utilização da subrotina ME28, conforme podem ser vistas nas Figuras 5.7, 5.8, 5.9 e 5.10. Nestas figuras temos a ilustração do pulso passando da fibra da esquerda para direita.

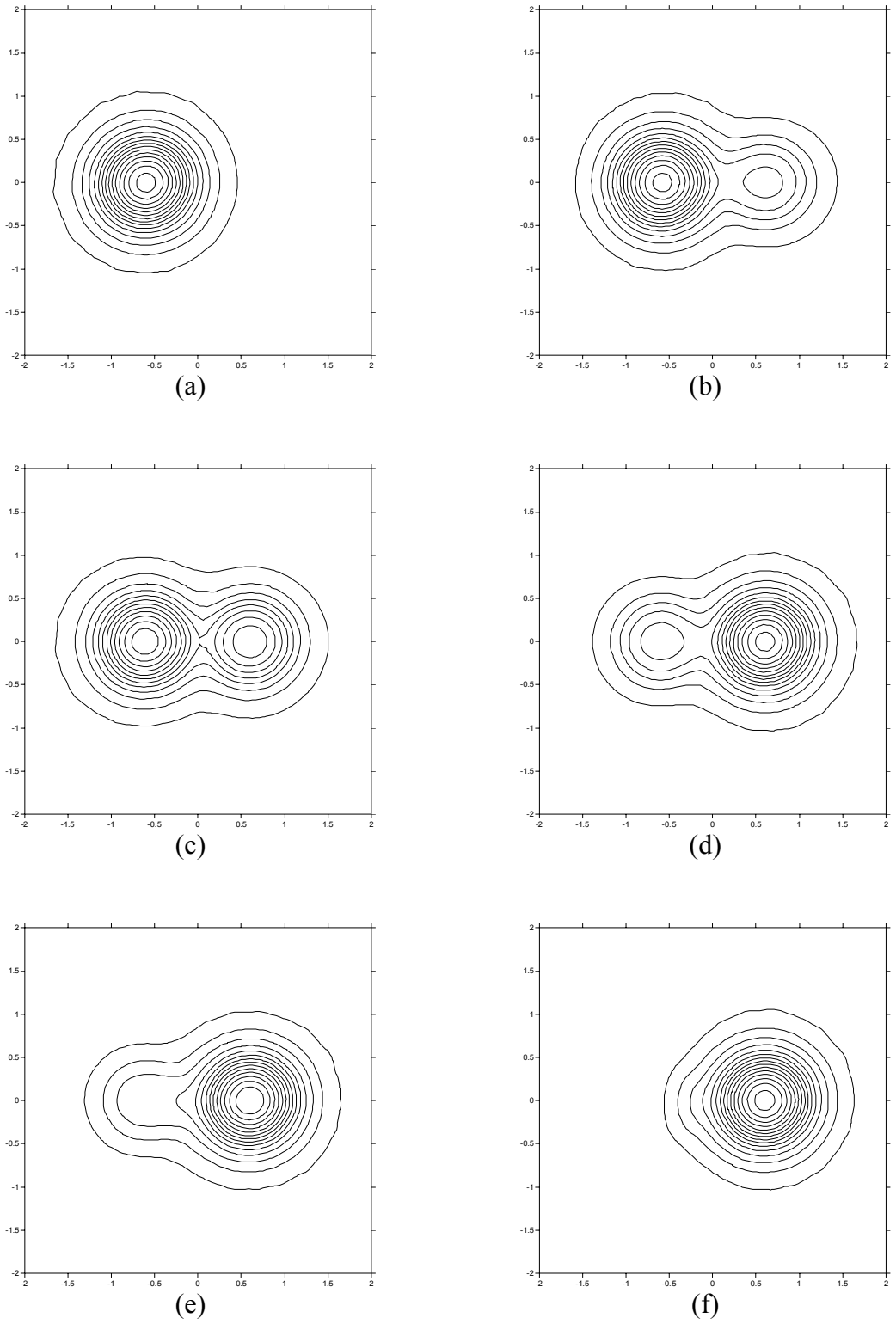


Figura 5.7 - Curvas de contornos para o valor absoluto da componente h_x utilizando a subtorina implementada, para: (a) z = 0 μm, (b) z = 9 μm, (c) z = 15 μm, (d) z = 30 μm, (e) z = 33 μm e (f) z = 39 μm.

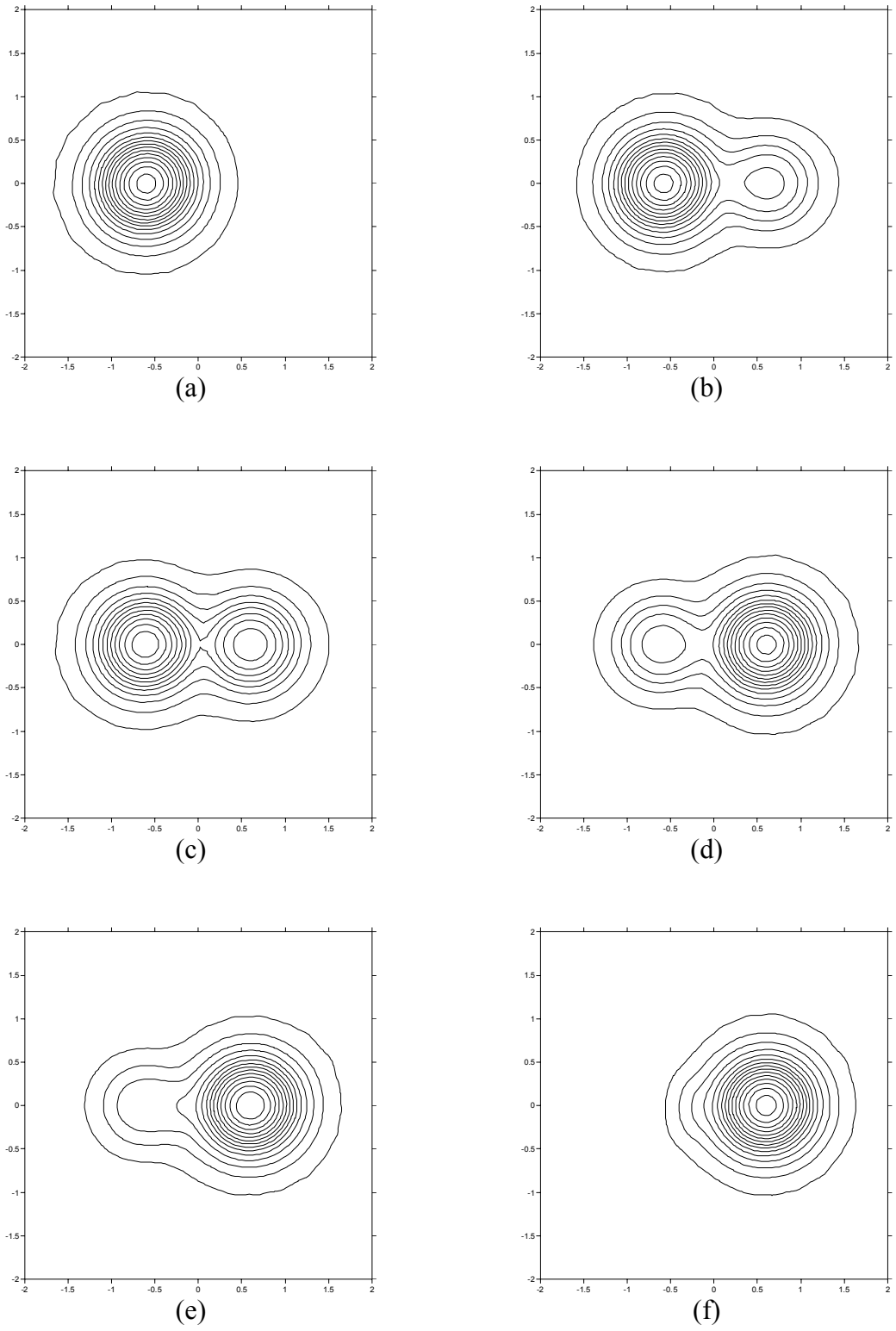


Figura 5.8 - Curvas de contornos para o valor absoluto da componente h_x utilizando a ME28, para: (a) z = 0 μm, (b) z = 9 μm, (c) z = 15 μm, (d) z = 30 μm, (e) z = 33 μm e (f) z = 39 μm.

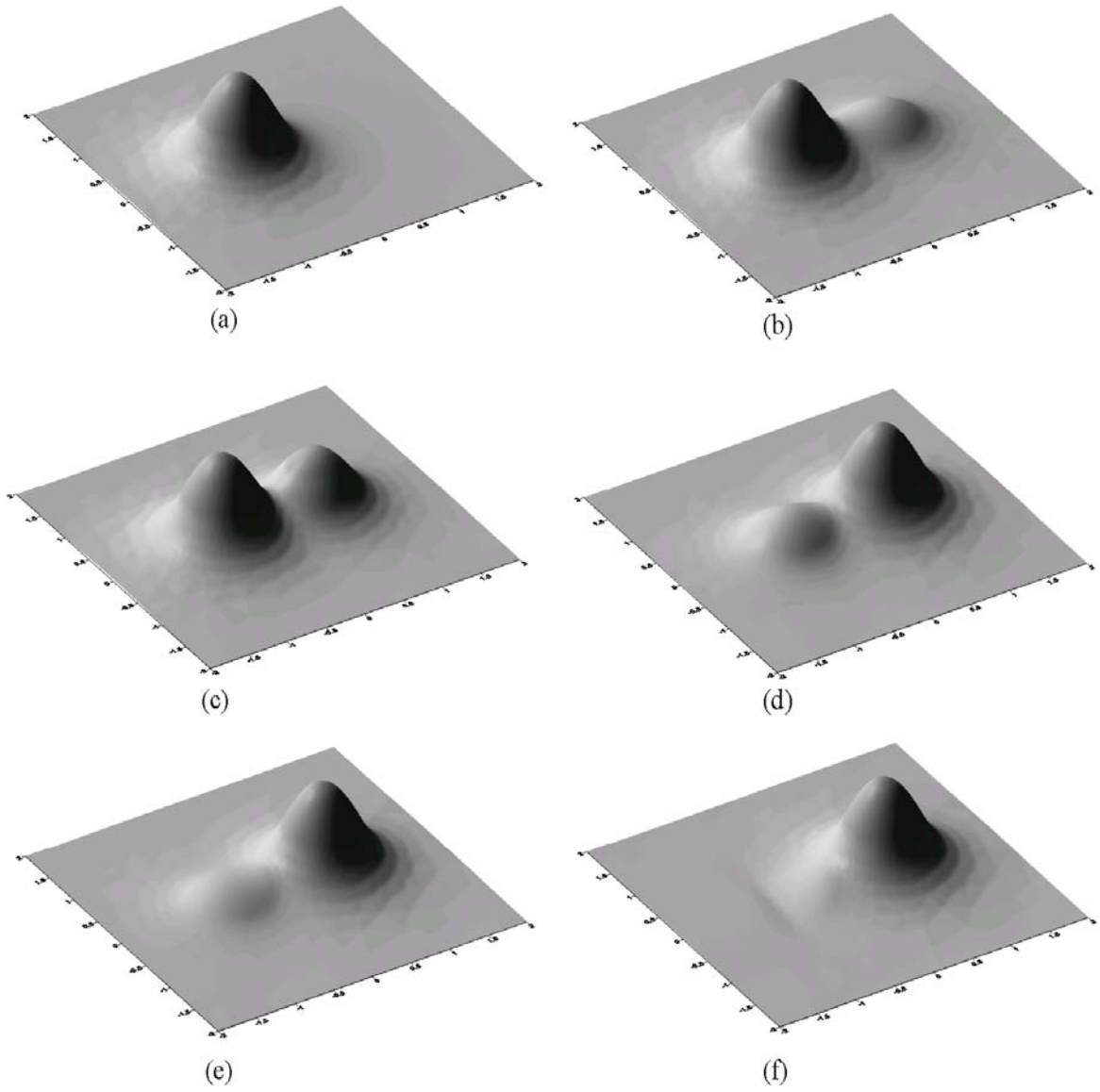


Figura 5.9 - Superfícies para o valor absoluto da componente h_x, utilizando a LIRIOS, para: (a) z = 0 μm, (b) z = 9 μm, (c) z = 15 μm, (d) z = 30 μm, (e) z = 33 μm e (f) z = 39 μm.

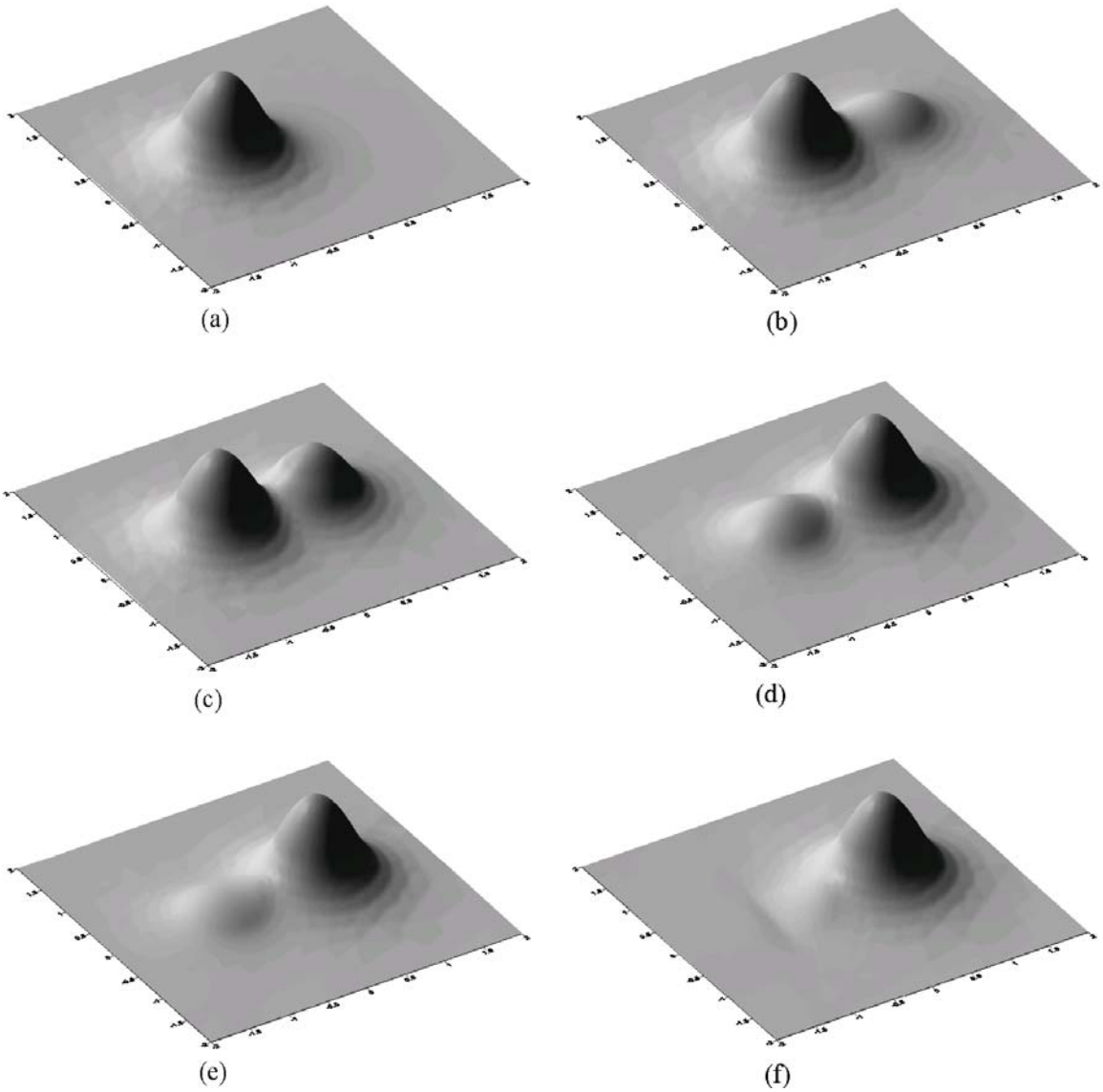


Figura 5.10 - Superfícies para o valor absoluto da componente h_x, utilizando a ME28, para: (a) $z = 0$ μm, (b) $z = 9$ μm, (c) $z = 15$ μm, (d) $z = 30$ μm, (e) $z = 33$ μm e (f) $z = 39$ μm.

5.1.3 Guia de onda óptico anisotrópico em substrato $LiNO_3$

Este problema é referente a uma tese de doutorado [55] cuja solução do sistema linear utilizou a ME28; o método de propagação utilizado, em [55], foi o BPM.

Nesta formulação, a matriz é atualizada em cada passo de propagação, devido à anisotropia do material [55], [57]. Esta atualização implica em uma decomposição ILU em cada passo, desta forma este caso diferencia-se das aplicações aqui apresentadas.

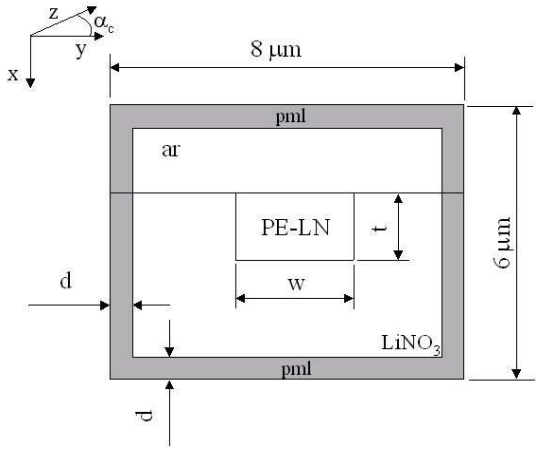


Figura 5. 11 - Guia de onda PE-LN

O guia de onda, representado na Figura 5.11, é rodeado por PMLs, cujo substrato é o $LiNbO_3$ (LN) e a região de guiamento é o $LiNbO_3$, obtido por troca iônica (PE-LN) [57]. Possuindo as seguintes características: w = 2 μm, t = 2 μm, os índices ordinários e extraordinários: n_o = 2,25 e n_e = 2,172, a diferença entre os índices do substrato e a região de guiamento é dado por Δn_e = 0,02.

Abaixo temos as componentes dos tensores permissividade elétrica:

$$\varepsilon_{xx} = n_0^2 \cos^2 \alpha_c + n_e^2 \operatorname{sen}^2 \alpha_c \qquad (5.5)$$

$$\varepsilon_{yy} = n_e^2 \cos^2 \alpha_c + n_0^2 \operatorname{sen}^2 \alpha_c \qquad (5.6)$$

$$\varepsilon_{zz} = n_0^2 \qquad (5.7)$$

$$\varepsilon_{xy} = \varepsilon_{yx} = (n_e^2 - n_0^2) \cos\alpha_c \operatorname{sen}\alpha_c , \qquad (5.8)$$

onde $\alpha_c = 45^o$.

Consideramos um passo de propagação de Δz = 0,1 μm, comprimento de onda de 0,84 μm, o domínio computacional de 8 μm na direção x e 6 μm na direção y, com uma PML de d = 2μm.

O campo inicial h_x, polarizado na direção x, é dado por um feixe gaussiano na forma:

$$h_x = Aa\ exp(-(x^2 - y^2) / 2w), \qquad (5.9)$$

onde Aa = 1 ua é a amplitude, e w = 0,6 μm a largura do feixe.

A malha obtida com o GID possui 3.754 nós, a matriz possui 50.149 valores não-nulos, é não-hermitiana e não-simétrica. O desempenho das duas subrotinas, ME28 e a implementada, é apresentado na Tabela 5.5, notamos um melhor desempenho para a LIRIOS em relação a ME28.

LIRIOS	1,06 s
ME28	1,57 s

Tabela 5.5 - Comparação entre rotinas - guia de onda anisotrópico

Na Figura 5.12 temos o gráfico da esparsidade da matriz, que mostra uma concentração dos valores não nulos ao redor da diagonal.

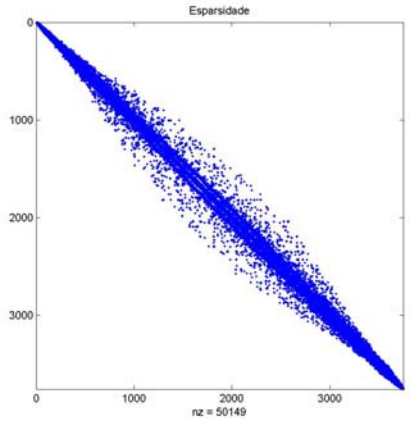


Figura 5.12 - Esparsidade da matriz - guia anisotrópico

Os parâmetros utilizados na LIRIOS foram: *lfil =3.754* e *droptol = 1e-3,* com uma tolerância de 5e-15.

O índice de referência é atualizado a cada passo de propagação, pelo propagador, convergindo para o índice do modo fundamental do guia, que pode ser visto na Figura 5.13, o resultado obtido é idêntico ao obtido utilizando a ME28.

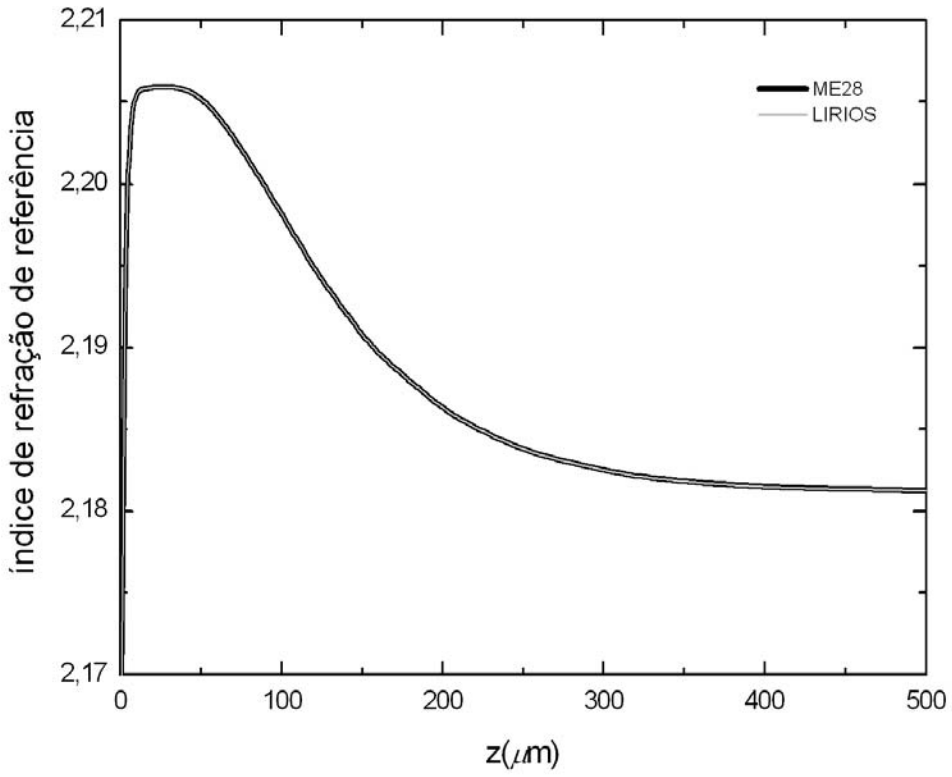
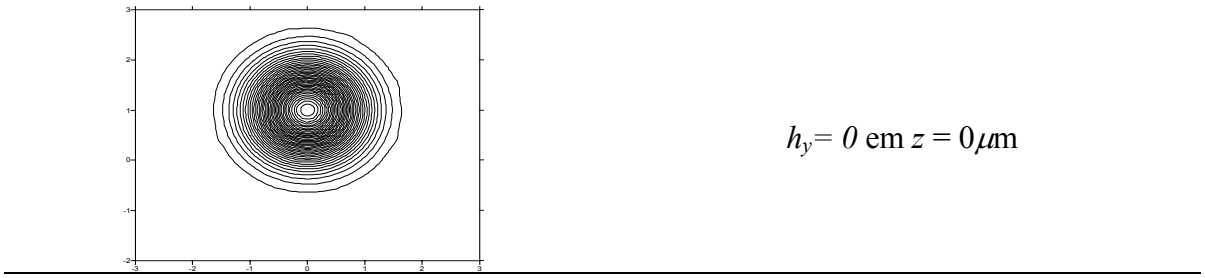


Figura 5.13 - Índice de refração de referência - guia anisotrópico

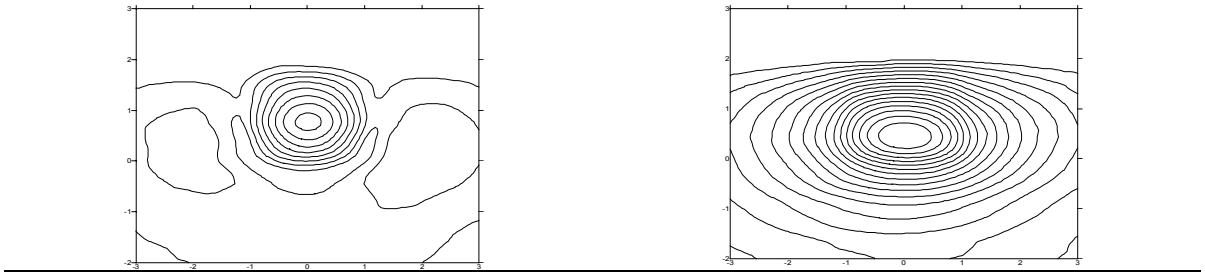
Nas Figuras 5.14 e 5.15, notamos que a componente h_y sai de um valor nulo até igualar à componente h_x, algo esperado pela devido a característica híbrida do modo fundamental.

Nas Figuras 5.14 e 5.15 temos uma concordância dos resultados entre as subrotinas avaliadas, o mesmo ocorrendo nas Figuras de superfícies 5.16 e 5.17.

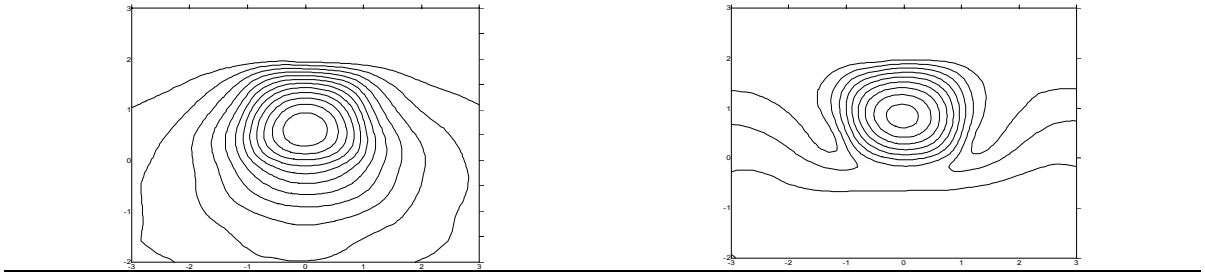


$h_y = 0$ em $z = 0\mu$m

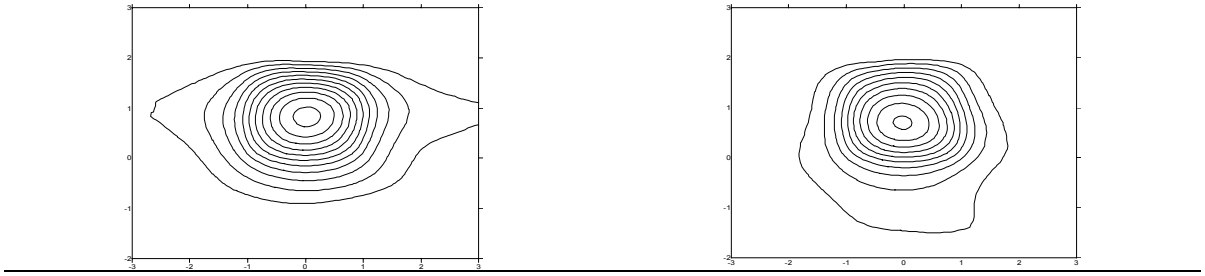
(a)



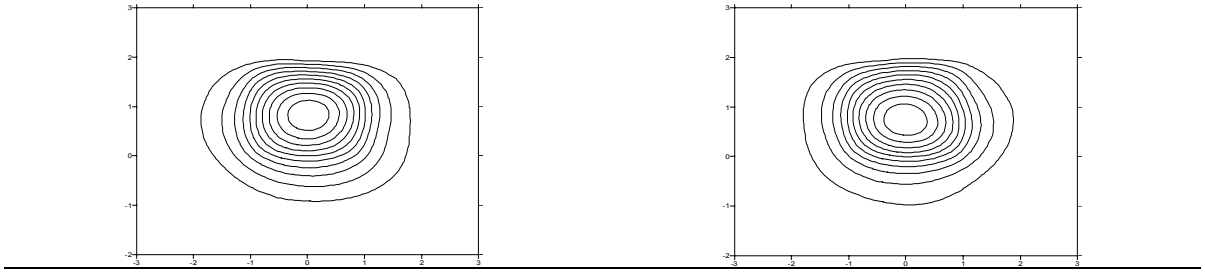
(b)



(c)



(d)



(e)

Figura 5.14- Componentes em módulo de h_x, e h_y (esquerda e direita) utilizando a ME28, para (a) $z = 0$ μm, (b) $z = 20$ μm, (c) $z = 40$ μm, (d) $z = 200\mu$m e (e) $z = 500\mu$m

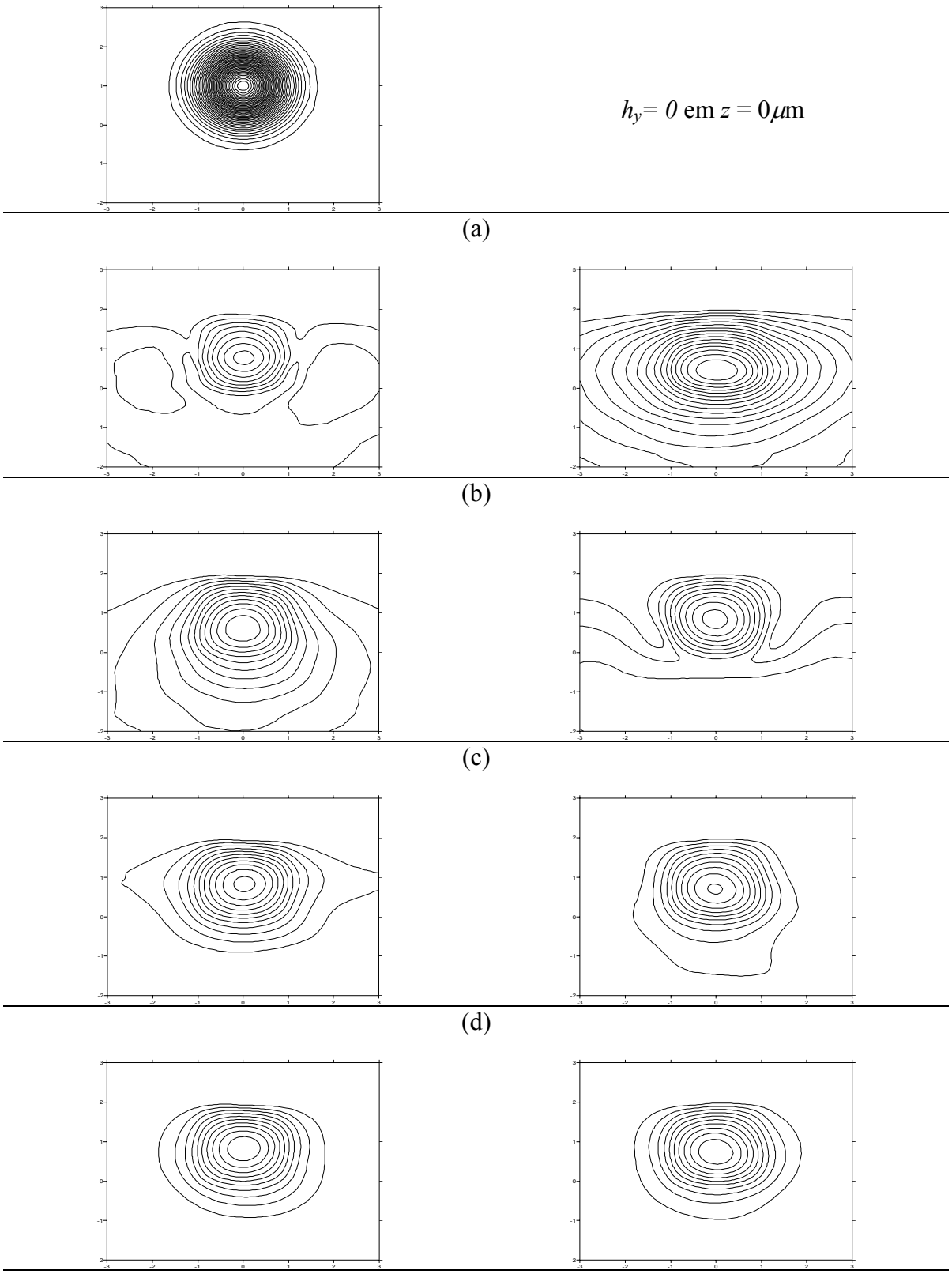


Figura 5.15 - Componentes em módulo, de h_x, e h_y (esquerda e direita), utilizando a LIRIOS, para (a) z = 0 μm, (b) z = 20 μm, (c) z = 40 μm, (d) z = 200 μm e (e) z =500 μm

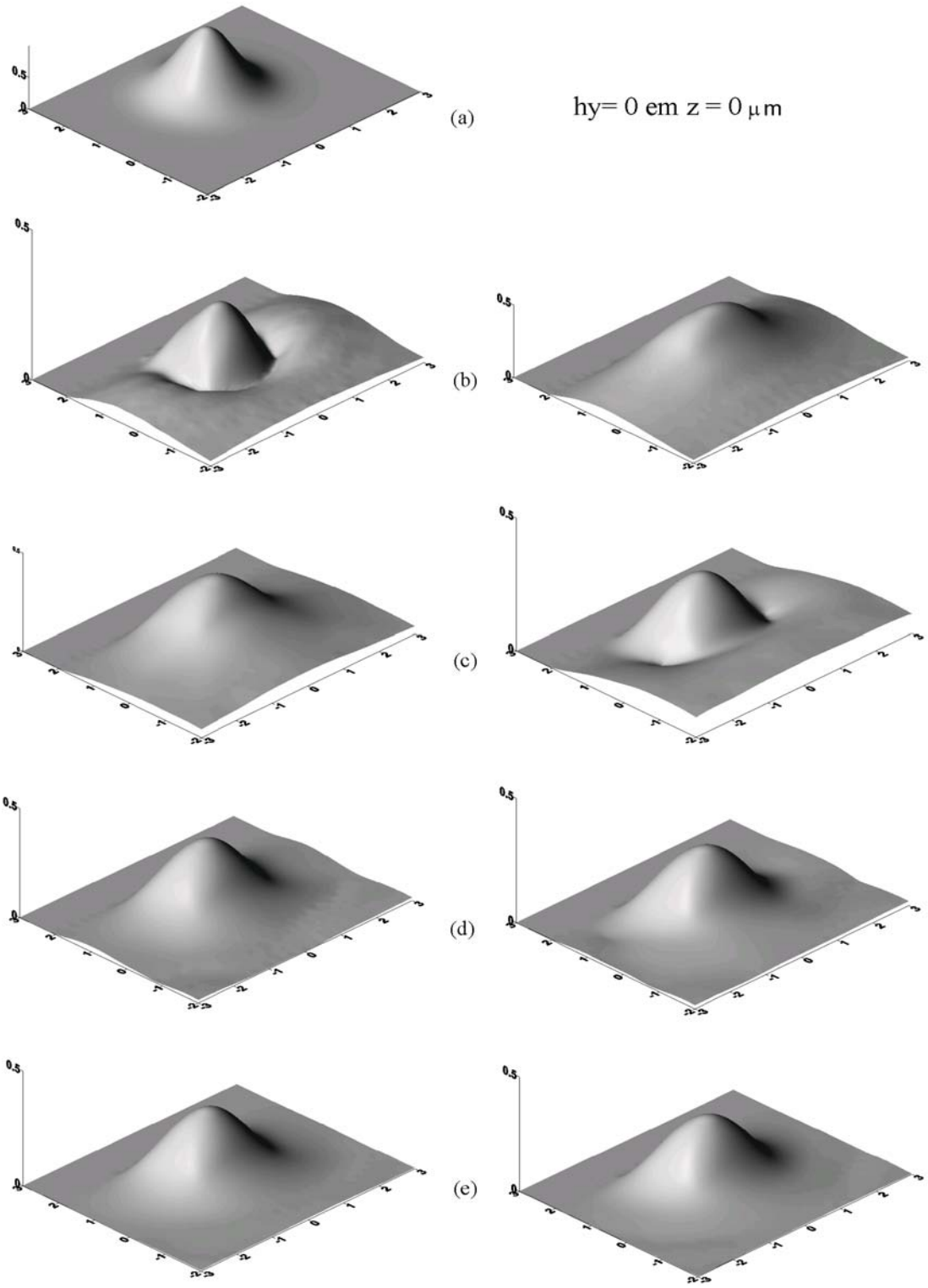


Figura 5.16 - Superfícies das componentes em módulo, h_x, e h_y (esquerda e direita), utilizando a ME28, para (a) $z = 0$ μm, (b) $z = 20$ μm, (c) $z = 40$ μm, (d) $z = 200$ μm e (e) $z = 500$ μm

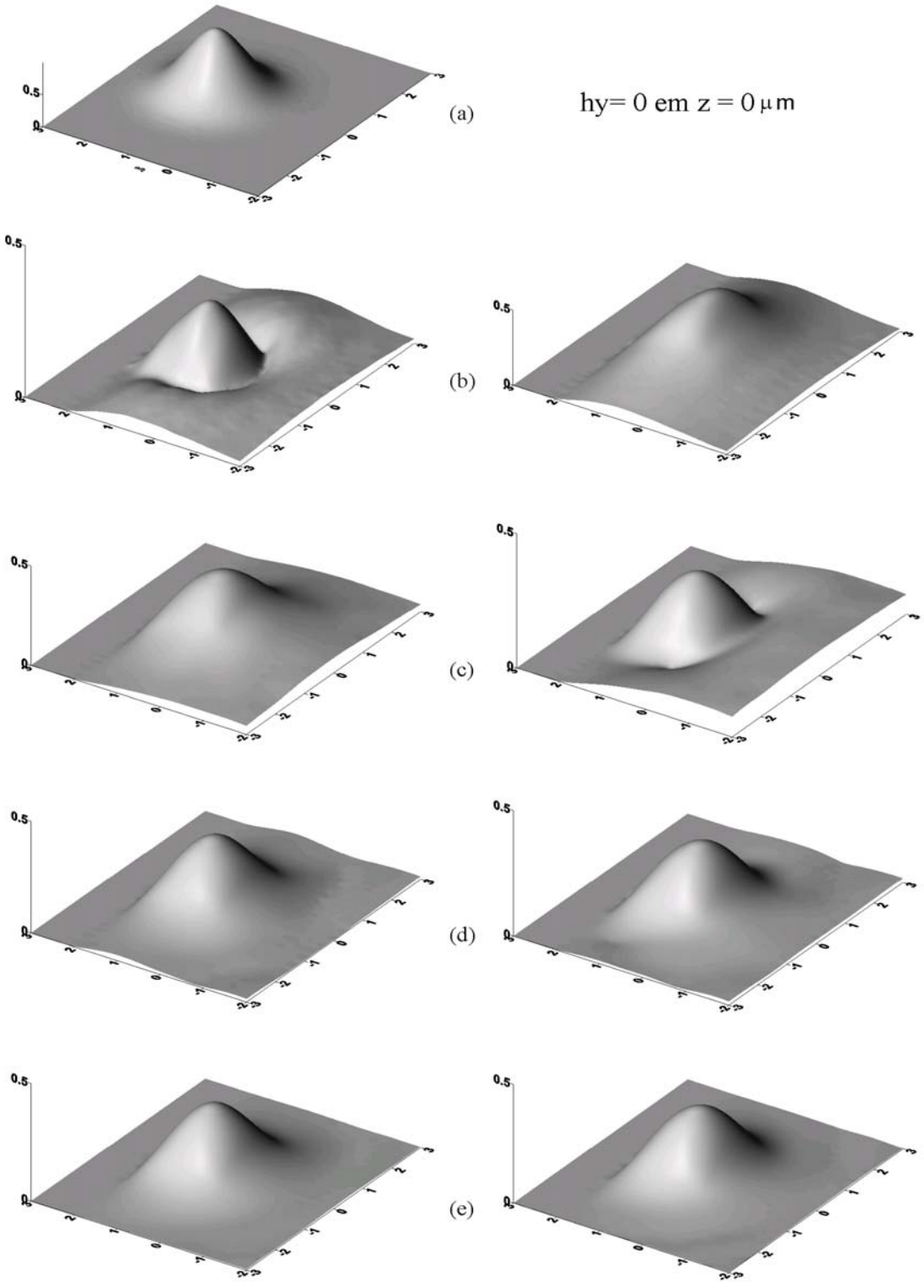


Figura 5.17 - Superfícies das componentes em módulo, h_x, e h_y (esquerda e direita), utilizando a LIRIOS, para (a) z = 0 μm, (b) z = 20 μm, (c) z = 40 μm, (d) z = 200 μm e (e) z =500 μm

5.1.4 Junção em ângulo reto PBG

Apresentaremos, abaixo, os resultados obtidos para a junção em ângulo reto com, reflexão em colunas dielétricas, apresentada em tese de doutorado [53], a propagação é realizada no domínio do tempo. Na Figura 5.18 temos a ilustração do guia, no canto superior esquerdo temos as colunas dielétricas (cristais fotônicos), que funcionam como um refletor, a onda se propaga pelo guia e ao encontrar com as colunas dielétricas são refletidas a 90^{o} e continuando a se propagar pelo guia.

As malhas utilizadas possuem, aproximadamente, 80.000, 180.000 e 250.000 nós, as colunas dieléricas, de índice 3,841, possuem um raio de 86,8 *n*m, os índices de refração do guia e do substrato são 1,50 e 1,465, respectivamente. A avaliação dos resultados foram entre o Matlab, software utilizado na tese de doutorado [53], e a LIRIOS. O matlab dispõe de uma função que resolve sistemas através do método do gradiente biconjugado estabilizado, com precondicionamento; a decomposição ILU foi utilizada, também, nestes casos. Desta forma os resultados obtidos em [53] servirão de comparação para verificação da eficácia da LIRIOS.

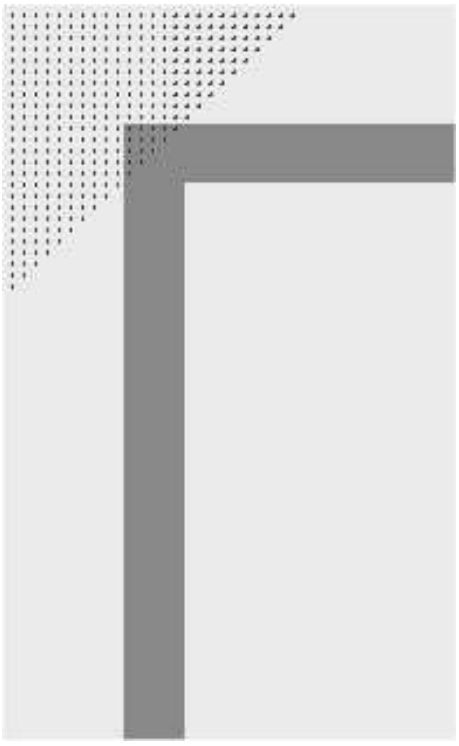


Figura 5.18 - Junção em ângulo reto

Os testes de desempenho foram feitos para o caso mais crítico: sistema linear com 250.000 equações, aproximadamente. Quanto à solução do sistema linear, as curvas de níveis das Figuras 5.20 e 5.21 mostram que a solução com a LIRIOS é idêntica a solução obtida através da abordagem desenvolvida na tese de doutorado [53], utilizando o Matlab.

	Linux	**Windows**
LIRIOS	2,17 s	4,27 s
Matlab 6.2	--------	16,60 s

Tabela 5.6 - Comparação de desempenho entre plataformas - junção em ângulo reto

Na Tabela 5.6 temos o tempo de solução para cada iteração; notamos que os resultados obtidos com o sistema operacional Linux possuem uma significativa vantagem em relação ao Windows, chegando a uma economia de 51% no desempenho, utilizando uma uma tolerância de 5e-15 no método iterativo.

Na Tabela 5.7 temos o desempenho referente à uma decomposição ILU incompleta, com *droptol* de *1e-7*; notamos, novamente, uma vantagem do sistema Linux em relação ao Windows.

	Linux	**Windows**
LIRIOS	43,44 s	72,42 s
Matlab 6.2	----------	156,56 s

Tabela 5.7 - Comparação de desempenho entre plataformas - lu incompleta

Na comparação de desempenho entre o Matlab e a LIRIOS nas Tabelas 5.7 e 5.8 obtém-se algo que era esperado, pois uma trabalha com linguagem interpretada e compilada (Fortran77), respectivamente. Isto ocorre pelo fato da linguagem compilada trabalhar em nível de programação mais baixo, em relação a linguagem interpretada .

A malha foi obtida através do software GID. A matriz, neste caso, é hermitiana, o número de elementos não-nulos é de 1.701.730. A esparsidade pode ser visualizada na Figura 5.19, notamos novamente uma concentração dos valores próximos a diagonal da matriz.

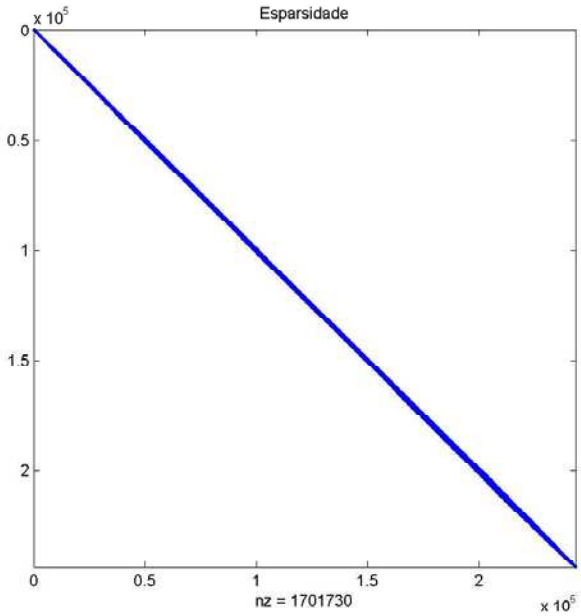


Figura 5.19 - Esparsidade da matriz - junção em ângulo reto

Analisando a seqüência das Figuras 5.20, 5.22 e 5.23 percebemos que à medida que a malha é refinada temos uma melhor definição do comportamento do pulso, para os valores de amplitudes menores; este mesmo fato pode ser notado nas Figuras 5.24, 5.26 e 5.27.

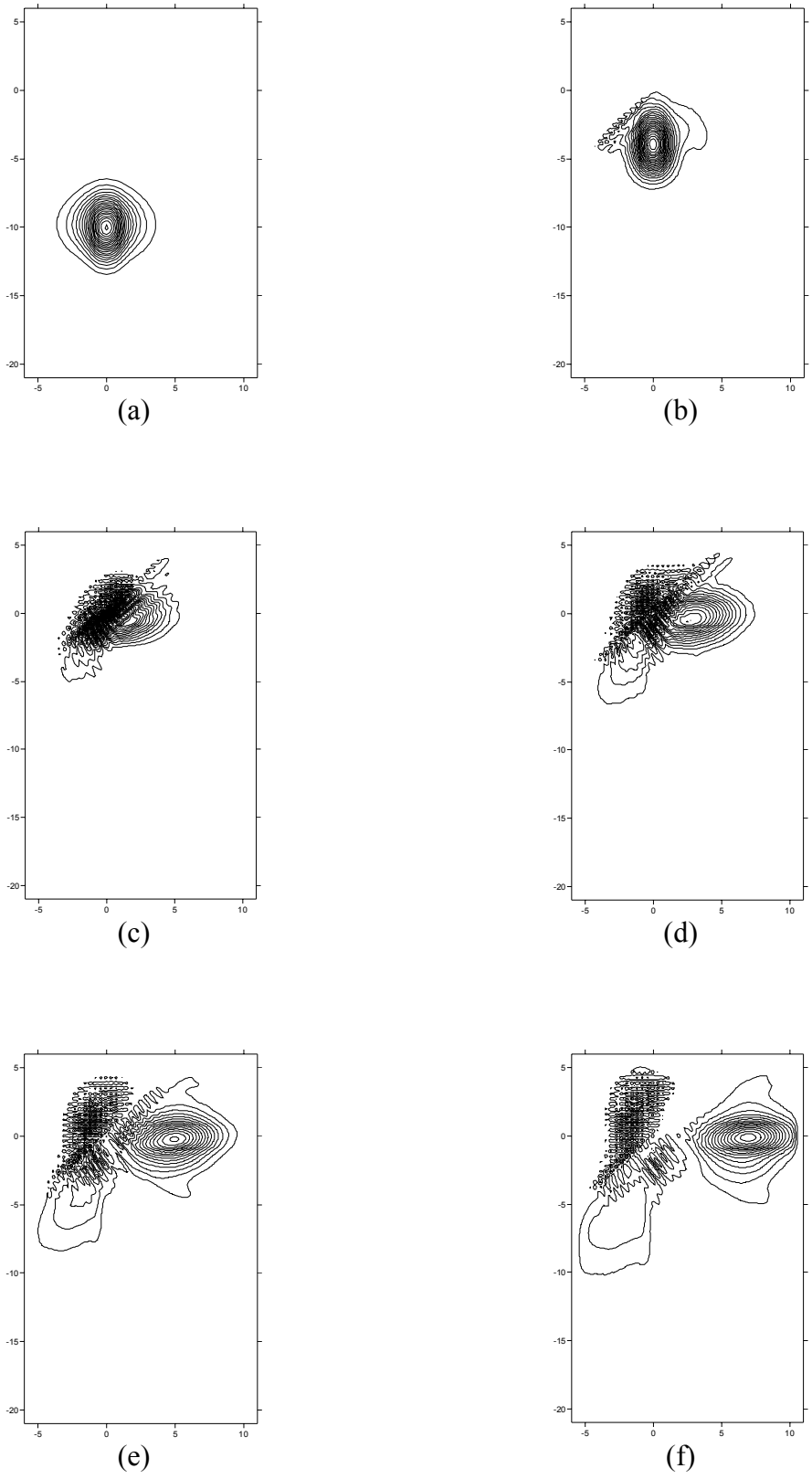


Figura 5.20 - Curvas de contornos para o valor absoluto, utilizando a LIRIOS (*n* = 8e4), nos instantes (a) 10 *f*s, (b) 40 *f*s, (c) 70 *f*s, (d) 80 *f*s, (e) 90 *f*s e (f) 100 *f*s

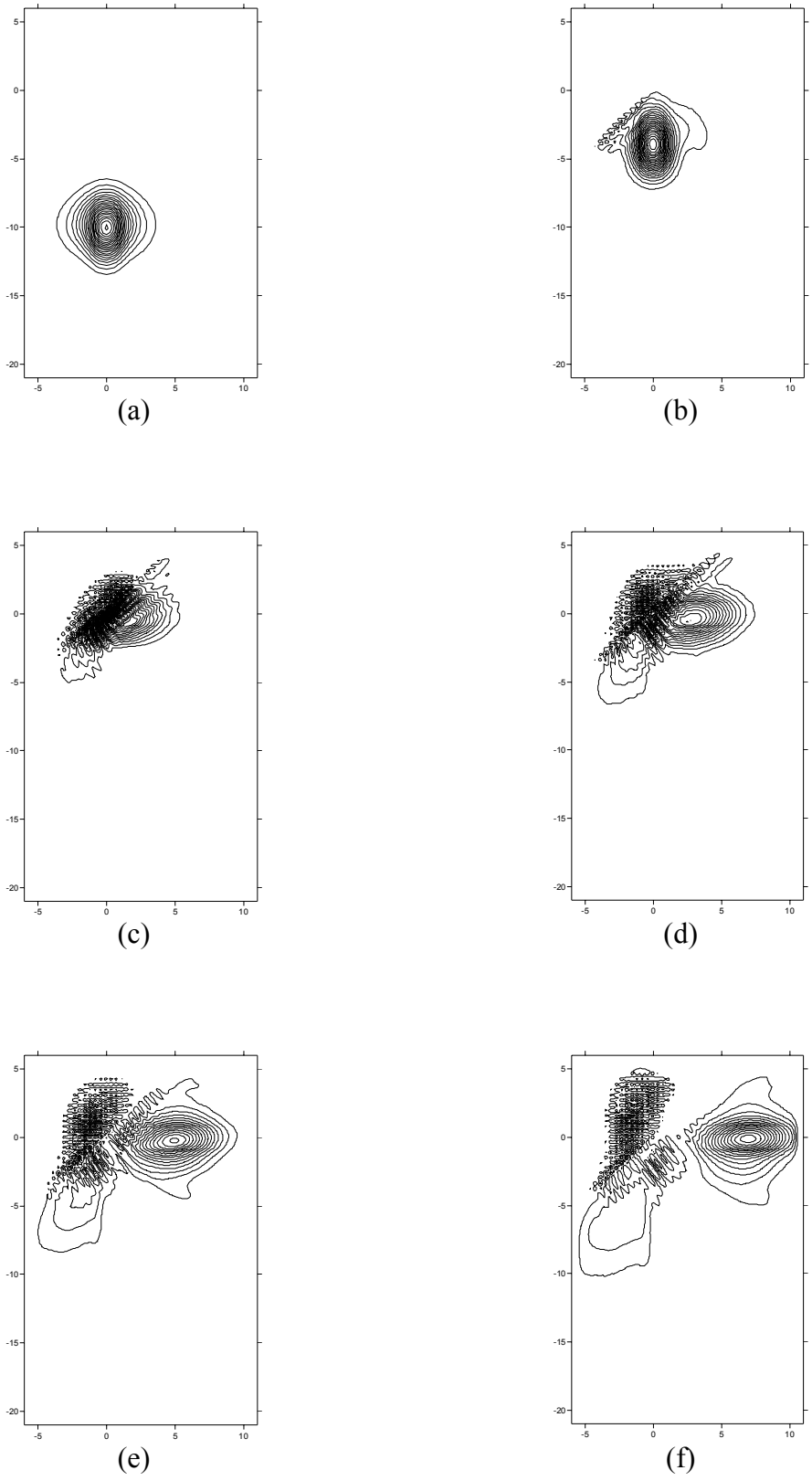


Figura 5.21 - Curvas de contornos para o valor absoluto, utilizando o matlab (*n* = 8e4), nos instantes (a) 10 *f*s, (b) 40 *f*s, (c) 70 *f*s, (d) 80 *f*s, (e) 90 *f*s e (f) 100 *f*s

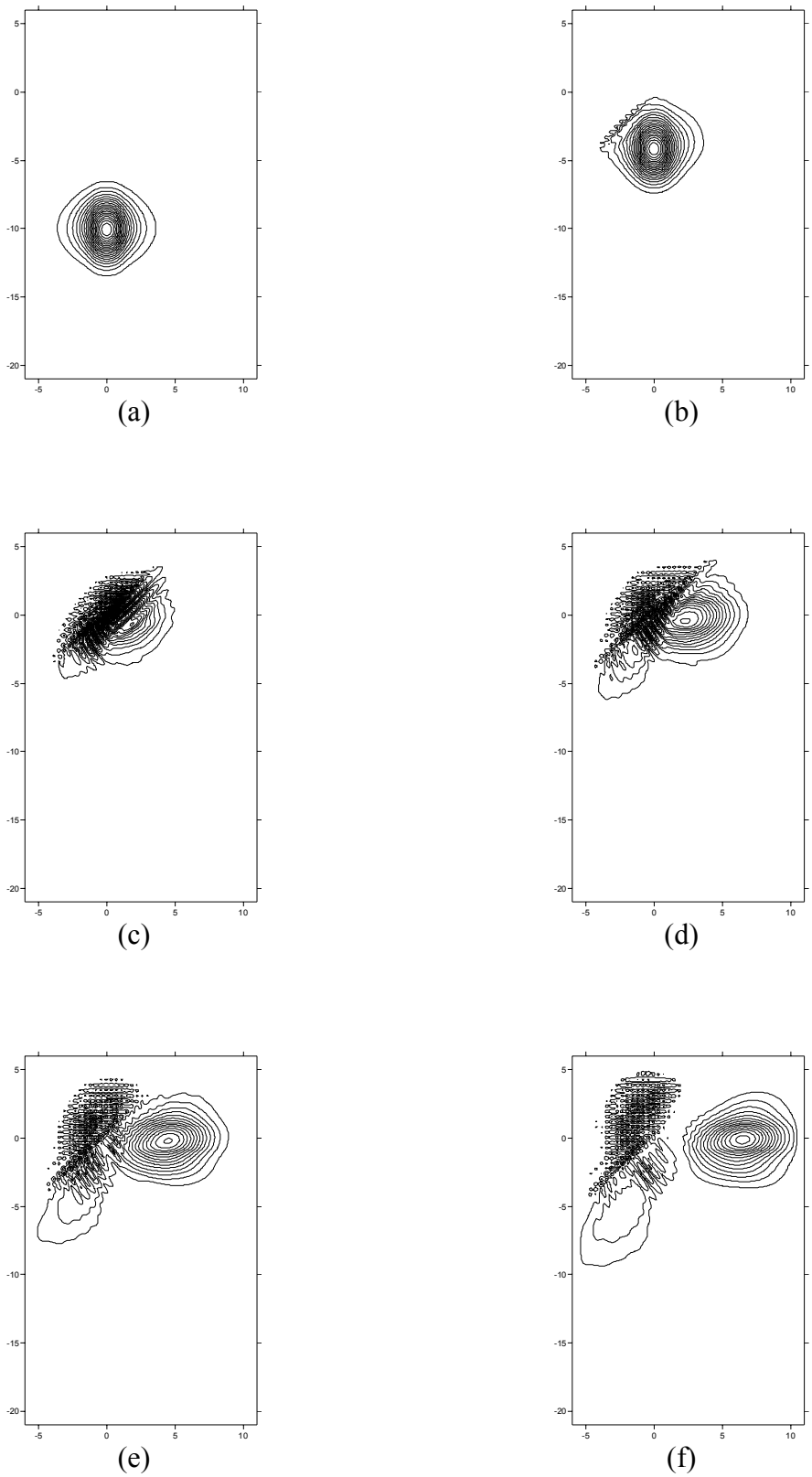


Figura 5.22 - Curvas de níveis para o valor absoluto, utilizando a LIRIOS (*n* = 18e4), nos instantes: (a) 10 *f*s, (b) 40 *f*s, (c) 70 *f*s, (d) 80 *f*s, (e) 90 *f*s e (f) 100 *f*s

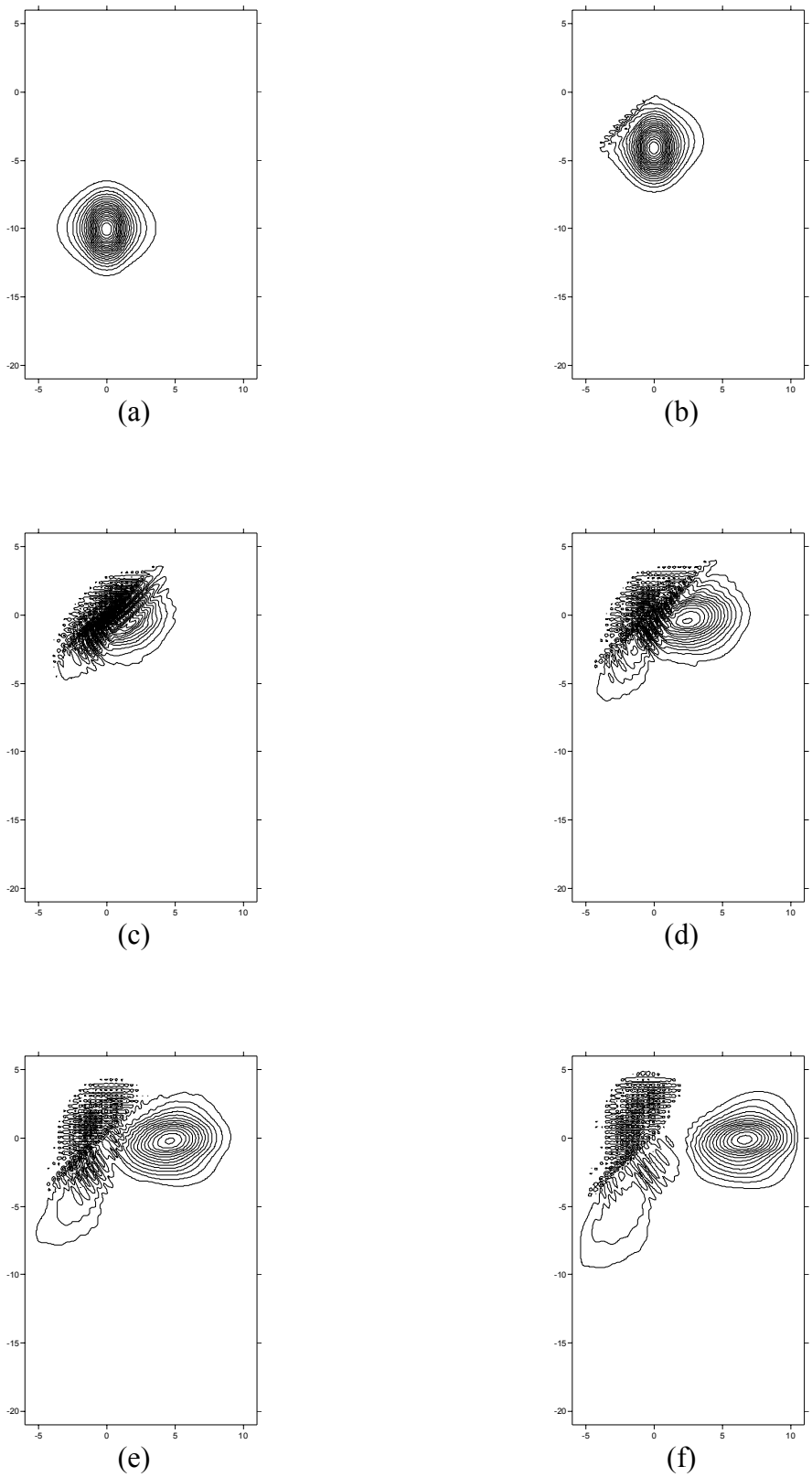


Figura 5.23 - Curvas de níveis para o valor absoluto, utilizando a LIRIOS (*n* = 25e4), nos instantes: (a) 10 *f*s, (b) 40 *f*s, (c) 70 *f*s, (d) 80 *f*s, (e) 90 *f*s e (f) 100 *f*s

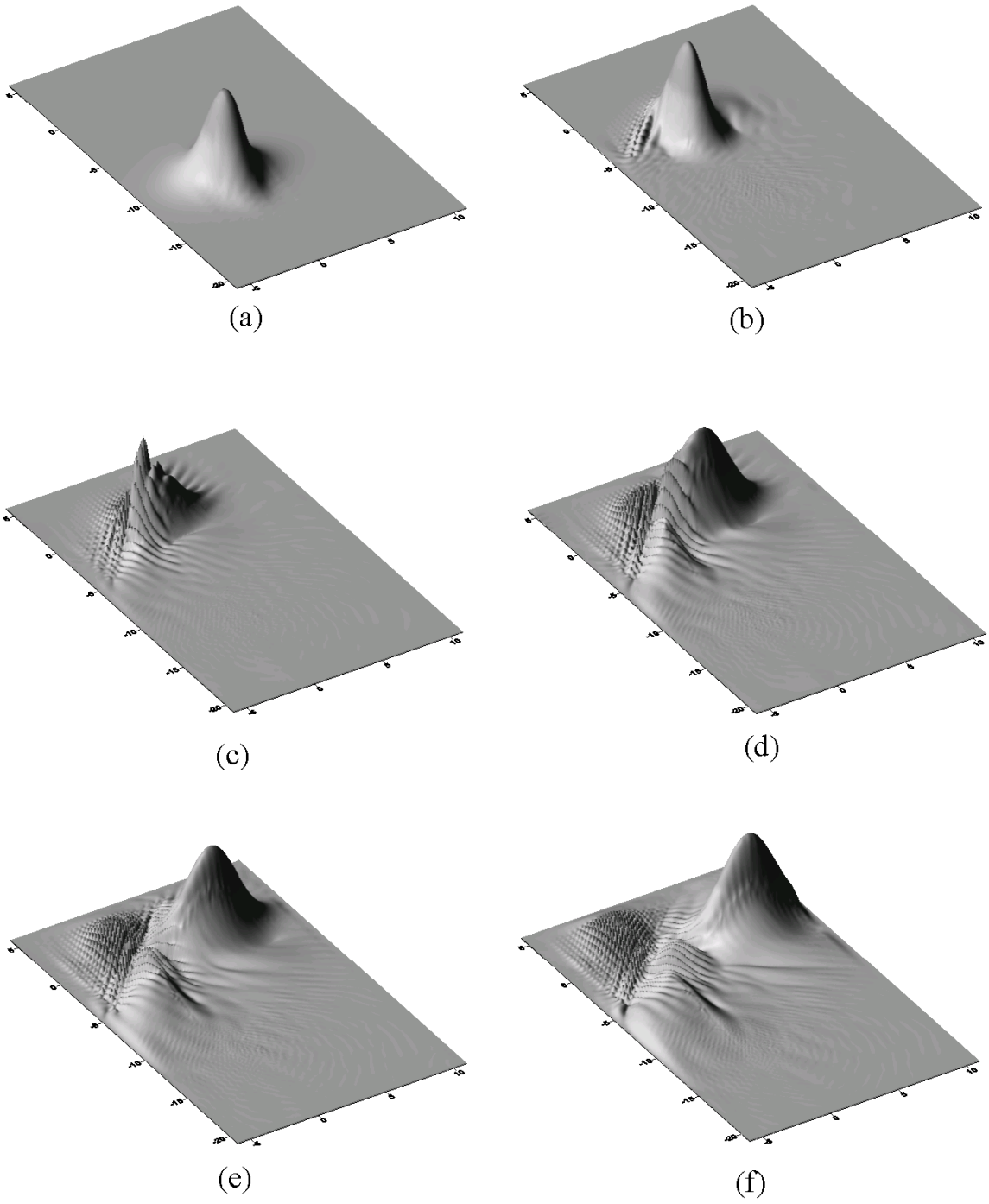


Figura 5.24 - Superfícies para o valor absoluto, utilizando a LIRIOS (*n* = 8e4), nos instantes: (a) 10 *f*s, (b) 40 *f*s, (c) 70 *f*s, (d) 80 *f*s, (e) 90 *f*s e (f) 100 *f*s

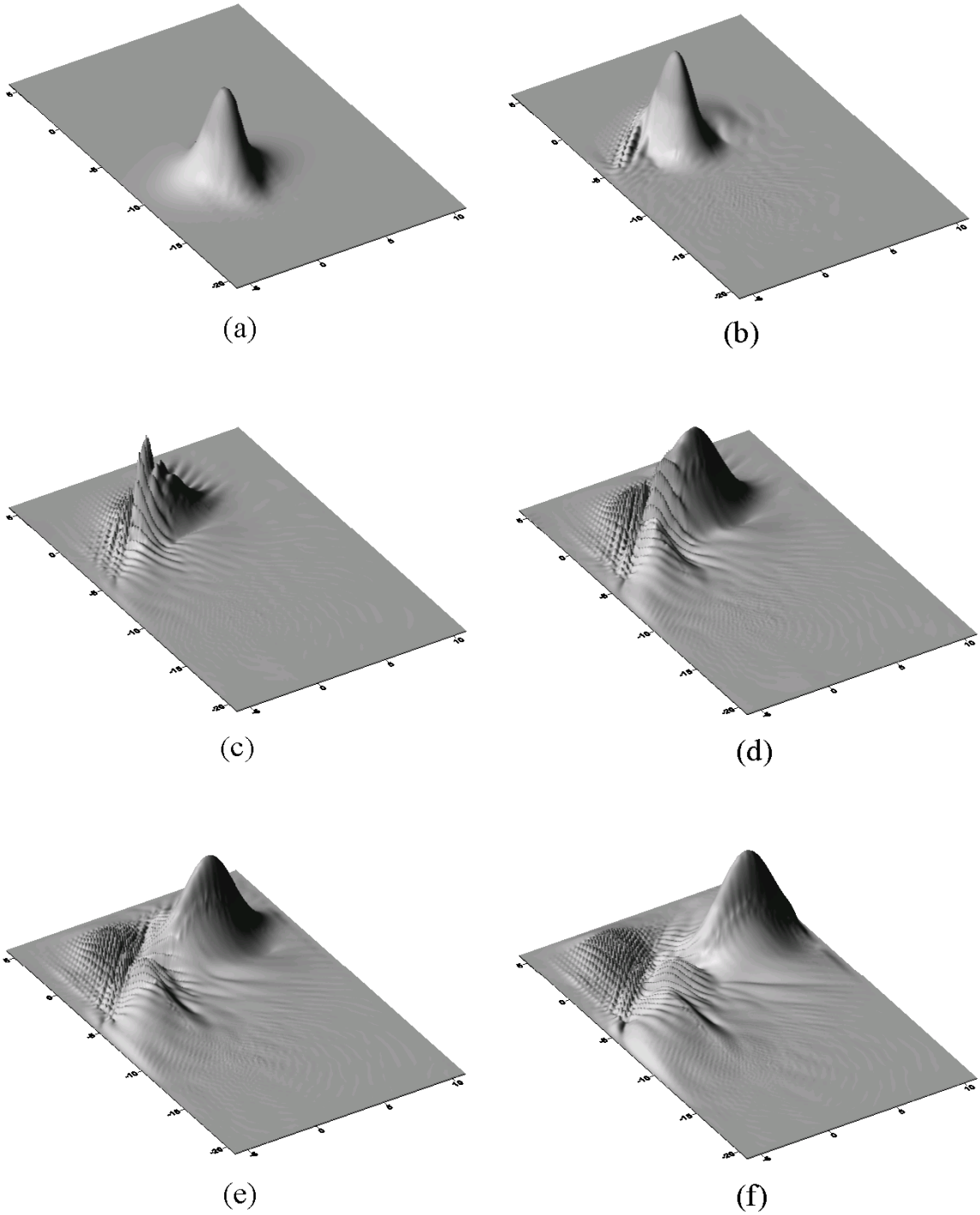


Figura 5.25 - Superfícies para o valor absoluto, utilizando o Matlab (*n* = 8e4), nos instantes: (a) 10 *f*s, (b)40 *f*s, (c)70 *f*s, (d) 80 *f*s, (e) 90 *f*s e (f) 100 *f*s

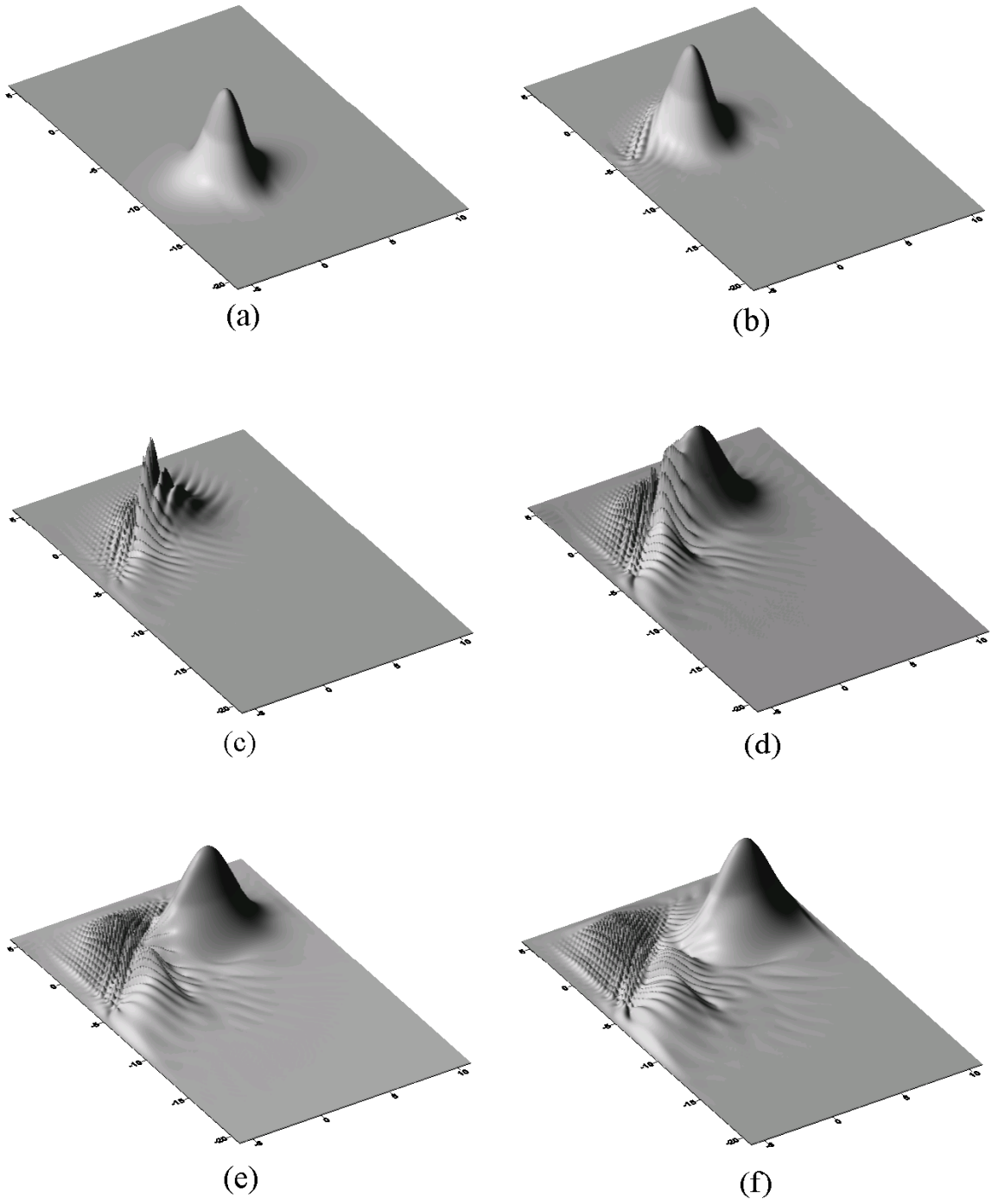


Figura 5.26 - Superfícies para o valor absoluto, utilizando a LIRIOS (*n* = 18e4), nos instantes: (a) 10 *f*s, (b) 40 *f*s, (c) 70 *f*s, (d) 80 *f*s, (e) 90 *f*s e (f) 100 *f*s

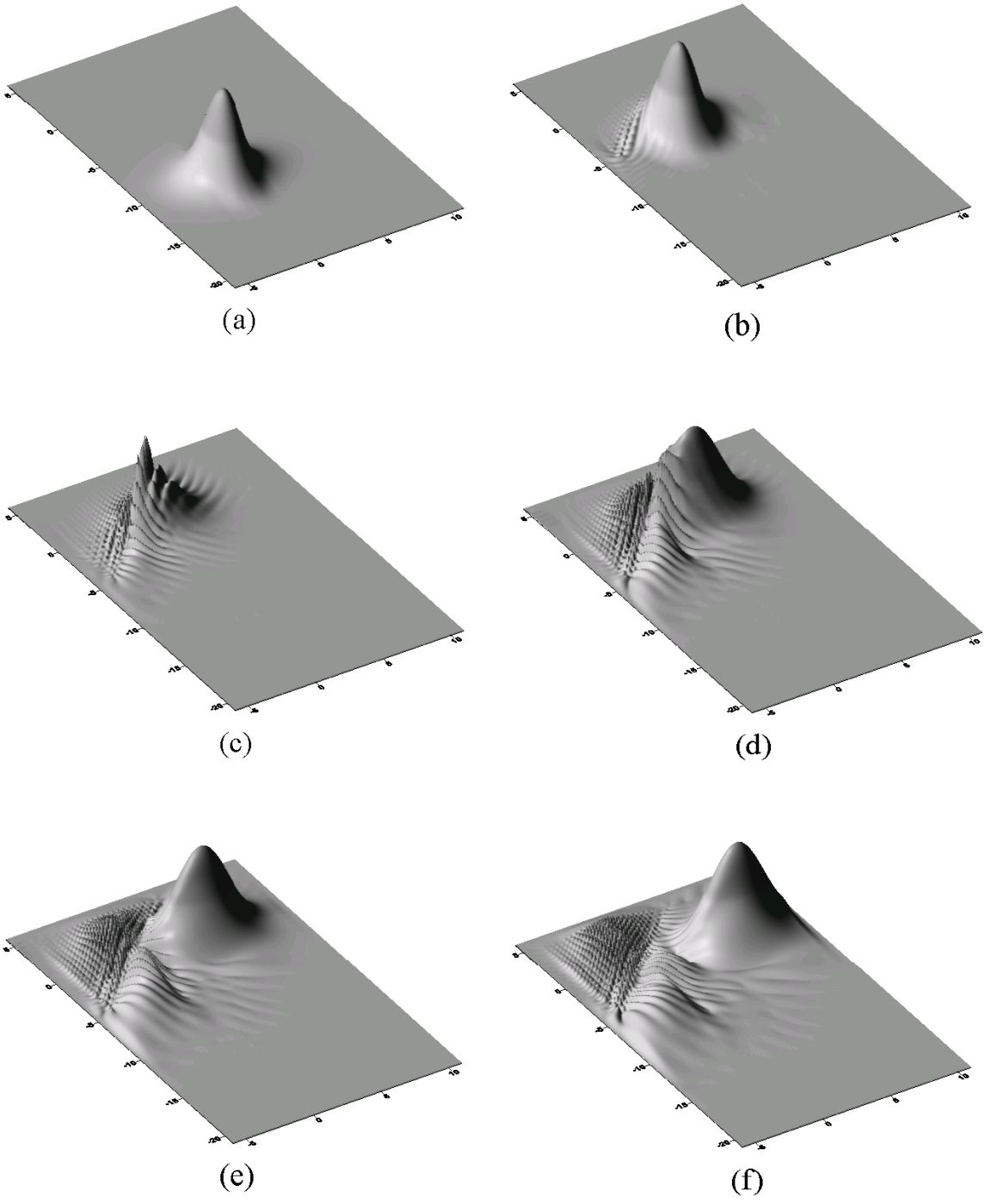


Figura 5.27 - Superfícies para o valor absoluto, utilizando LIRIOS (*n* = 25e4), nos instantes (a) 10 *f*s, (b) 40 *f*s, (c) 70 *f*s, (d) 80 *f*s, (e) 90 *f*s e (f) 100 *f*s

5.2 Análise para sistemas de grande porte

Consideramos como sistemas de grande porte aqueles de dimensões da ordem de 10^6 variáveis. Nesta seção, apresentaremos um breve estudo que permitirá avaliar o condicionamento da matriz, a partir de sistemas de dimensões menores.

Ao refinarmos uma malha, utilizando um mesmo algoritmo para sua construção, é esperado, pelo fato destes algoritmos serem determinísticos, que algumas características deste processo seja carregado pelas matrizes resultantes. Uma característica muito importante é o condicionamento da matriz, pois este influenciará na convergência dos métodos iterativos e, também, na estabilidade dos métodos diretos.

O condicionamento da matriz é dado em função da inversa da matriz de coeficientes (equação(5.10)) ou em função do maior e menor autovalores. Neste trabalho, o *número de condição* foi calculado através do Matlab, que utiliza a condição espectral dada pela equação (5.11).

$$condição = \left\| A \right\| . \left\| A^{-1} \right\|, \text{ onde } \| A \| = \max_{\|x\|=1} \| A x \|, \qquad (5.10)$$

$$cond\_espec = \lambda_{max} \Big/ \lambda_{min}, \qquad (5.11)$$

onde λmax e λmin são os autovalores máximo e mínimo, em valores absolutos.

Tentando evitar o cálculo do número de condição do sistema de grande porte, podemos fazer sua estimativa a partir do número de condição dos sistemas de menores dimensões, provenientes do mesmo problema físico, do mesmo método de discretização e ainda do mesmo malhador. Desta forma, assumimos que as mesmas características numéricas dos processos de cálculos das matrizes de pequeno porte, são atribuídas às matrizes de grande porte.

Com esse estudo um tanto heurístico, pela sua concepção de avaliação de um problema numérico, espera-se algum tipo de regularidade nas instâncias maiores. Caso o algoritmo numérico apresente irregularidades, por algum motivo, em sua evolução, o procedimento aqui aplicado poderá falhar. Mas, o principal ponto de instabilidade nos problemas aqui avaliados, estaria no malhador; como o malhador utilizado é o GID, podemos esperar que existam restrições que irão impedir a existência de triângulos degenerados, que poderiam invalidar a metodologia aqui proposta.

Realizamos os estudos do condicionamento para o problema da caixa condutora 2D e para o guia PBG. Calculamos o número de condição das matrizes de coeficientes no Matlab até onde foi possível, cujas restrições foram colocadas pela capacidade computacional disponível. . A máquina utilizada possui as seguintes características: Xeon 2Ghz, biprocessada, 512 Kb de Cache e 2Gb de memória RAM.

Temos, na Figura 5.28, o gráfico para o problema da caixa condutora:

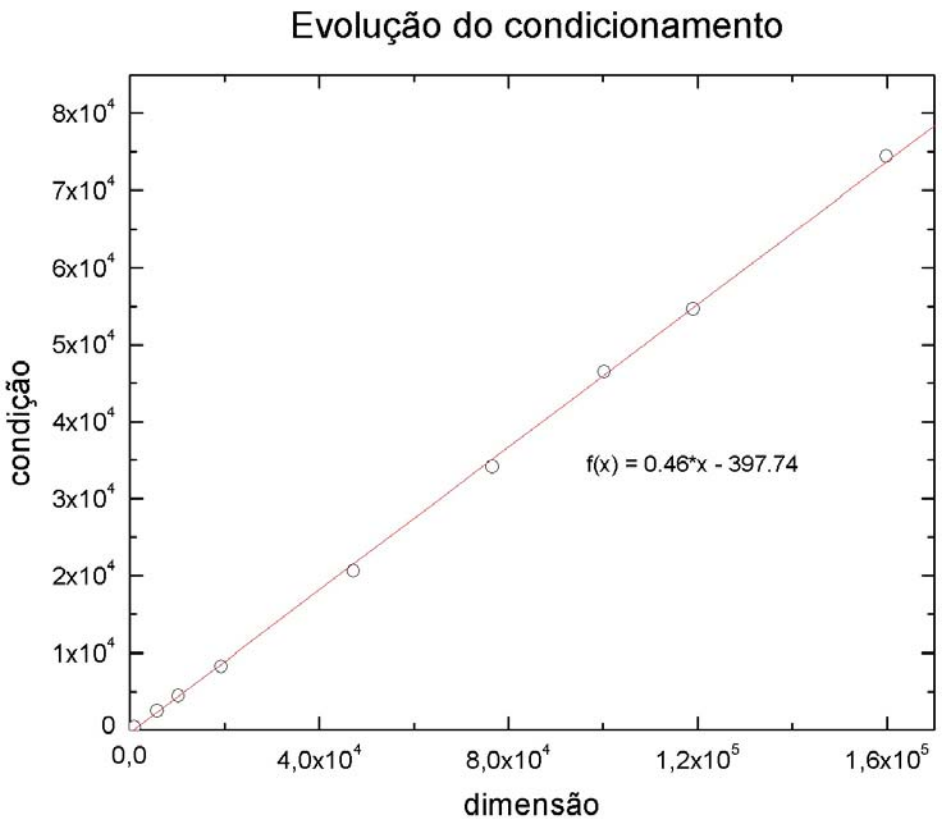


Figura 5.28 - Evolução do condicionamento-caixa condutora 2D

Como era esperado, temos uma regularidade no crescimento do número de condição das matrizes e podemos estimar, pelo método dos mínimos quadrados, a reta que rege a evolução do processo numérico deste problema, dada por (5.12).

$$f(x) = 0{,}46\,x - 397{,}74, \qquad (5.12)$$

onde x é a dimensão da matriz; para uma matriz de dimensão da ordem de 10^6, teremos um número de condição de aproximadamente $4{,}6\text{x}10^5$.

Na Figura 5.29 temos a avaliação para o guia canal PBG.

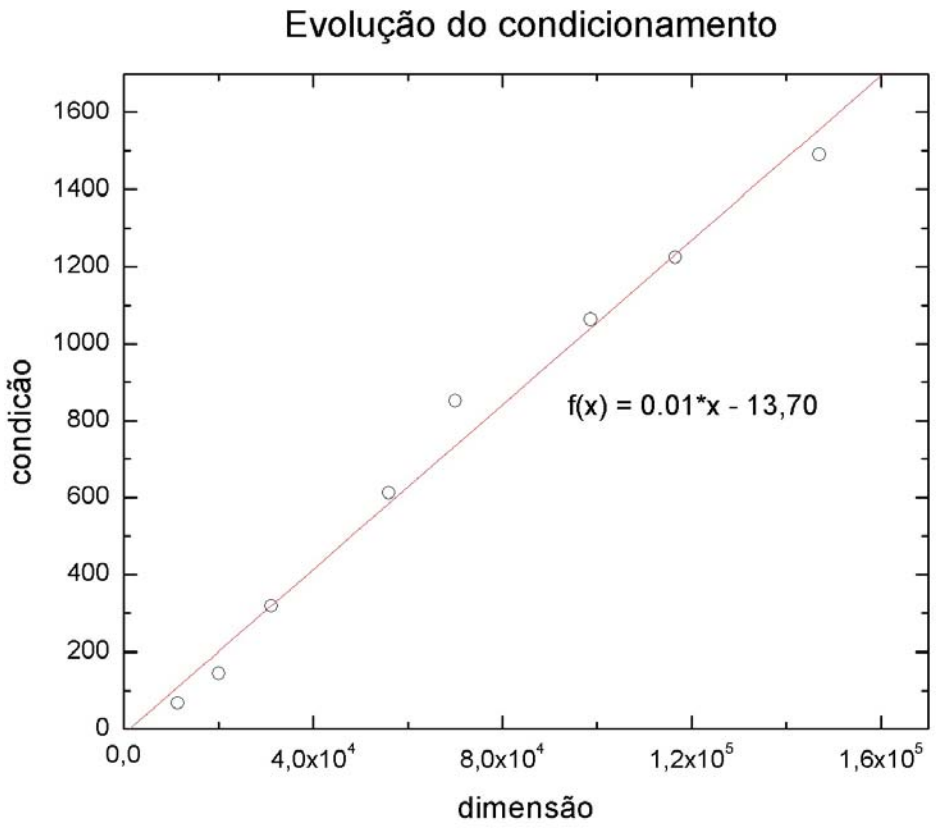


Figura 5.29 - Evolução do condicionamento-guia canal PBG

Para este problema, o número de condição da matriz pode ser estimado por (5.13).

$$f(x) = 0{,}01x - 13{,}70\ , \qquad (5.13)$$

onde x é a dimensão da matriz; então, para um sistema de dimensão 10^6, o número de condição desta matriz, é aproximadamente, 10^4.

Esta maneira de estimativa do cálculo de condicionamento sugere que ela pode ser aplicada para avaliar métodos de construção da malha e discretização, pois, calculando o coeficiente de cada reta, pode-se afirmar qual método é melhor, em relação ao condicionamento da matriz. Este tipo de análise poderá ser útil nos trabalhos futuros, para sistemas lineares de grande porte.

Capítulo 6

Conclusões

À medida que os trabalhos foram realizados notamos que a resolução de sistemas de grande porte defronta-se com duas barreiras: a primeira é a quantidade de memória disponível e, a segunda, a capacidade de processamento. Quando trabalha-se com o método direto temos um grande problema na quantidade de *fill-ins* que surgem no decorrer da decomposição LU (ME28) ao mesmo tempo que sobrecarrega-se a quantidade de processamentos.

Este problema, em relação à memória, pode ser observado através da escalabilidade nas Figuras 5.2 e 5.12, onde a ME28 foi capaz de resolver sistemas somente até 75.000 variáveis, com custo de processamento alto em relação à ***LIRIOS*** (para os casos variantes). Nos casos invariantes, dos testes analisados, percebemos que a escolha depende do número de passos que serão realizados na propagação, embora o limite de 75.000 variáveis se mantenha. Este limite de variáveis foi alcançado utilizando 1 GB de memória RAM.

Nos métodos iterativos, temos um refinamento de uma solução inicial, não trabalhando diretamente com a estrutura da matriz , portanto ocupando pouco espaço de memória, porém nos casos precondicionados com decomposição ILU incompleta novamente é colocada a barreira presente no método direto; mas, desta vez, saímos de um patamar de solução em torno de 75.000 para um patamar de 250.000 variáveis.

Com a LIRIOS, o grande ganho está na economia de memória, possibilitando solução de sistemas maiores, juntamente com uma baixa escalabilidade do método, tornando-o muito atraente e substituindo, com vantagens, a ME28 nas aplicações aqui tratadas.

O parâmetro *lfil* da decomposição ILUT, permite um controle do preenchimento da matriz, assim com um valor *lfil* menor, nos problemas das Seções 5.1.3, 5.2.1 e 5.2.4, poderíamos resolver sistemas de equações maiores, porém comprometendo o desempenho.

Quando relaxamos os parâmetros da decomposição incompleta, também significa que poderá ocorrer um aumento do nível de processamento, pois o método iterativo executará um número maior de iterações para obter a convergência, mas, em contrapartida, teremos a possibilidade de trabalharmos com matrizes maiores. Outro fator que permitiu minimizar a influência da memória, na solução do sistema, foi o reordenamento da matriz.

Ao aplicarmos a LIRIOS nos problemas do pulso gaussiano, tivemos a confirmação de sua eficiência, pois os valores encontrados estão muito próximos dos valores esperados. No guia canal e no acoplador direcional, o desempenho da LIRIOS foi melhor que o da ME28, e para a junção em ângulo reto com PBG, a substituição do Matlab pela LIRIOS também é vantajosa. Desta forma a LIRIOS apresenta-se com vantagens, como uma boa alternativa para os problemas tratados pelo grupo.

Os resultados foram obtidos em uma máquina com 1 GB de memória *RAM*, dois processadores *Xeon* de 512 kB de memória cache, os sistemas utilizados foram Linux RED HAT 7.2 e Windows 2000 (ambos em rede); os compiladores foram: F77 sob licença GNU e, no windows o software comercial F90 da Digital.

Na Seção 5.3, ao realizarmos uma análise do crescimento do número de condição, em função do número de nós da malha, para o problema do potencial e para o guia de onda canal PBG, temos parâmetros de avaliação do método numérico, formados pelo malhador e pelo método utilizado para a discretização do problema.

Este tipo de avaliação do método numérico pode ser útil para determinar o quanto um método é mais estável que outro, bem como para estimar o número de condição de matrizes de grande porte.

Esgotando as possibilidades no processamento serial, uma alternativa que se apresenta, de maneira cada vez mais efetiva, é o processamento paralelo. Com o avanço tecnológico, temos à disposição maiores capacidades de armazenamento nas memórias rápidas (RAM e CACHE), bem como redes de comunicação com altas taxas de transmissão, que permitem suavizar a influência da comunicação entre os processos.

6.1 Trabalhos futuros

Ao final deste trabalho, temos dois caminhos a seguir: um continuar com o processamento serial e, o outro, iniciarmos trabalhos em processamento paralelo.

No caso serial, poderíamos aperfeiçoar a LIRIOS quanto ao ordenamento e ao precondicionamento. Pode-se implementar o método de grau mínimo e um precondicionador que esteja mais próximo do problema físico.

Também é possível, com a experiência adquirida com as ferramentas para matrizes esparsas, complementar os programas implementados com outros recursos de solução tornando as ferramentas aqui desenvolvidas em um pacote (LIRIOS) de solução de sistemas lineares esparso completo.

No caso paralelo, temos o estado da arte em resolução de sistemas lineares de grande porte, tanto para métodos diretos quanto para métodos iterativos; nos métodos diretos, o método multi-frontal é uma ferramenta muito utilizada e, no caso do método iterativo, o paralelismo intrínseco nas operações algébricas é uma vantagem.

Pretende-se aprofundar estudos em plataformas paralelas para os métodos iterativos e, também, para os métodos diretos, pois o método direto contribuirá para o desenvolvimento de precondicionadores paralelos para o método iterativo.

Referências Bibliográficas

[1] J. Jim, *The finite element method in electromagnetics*, Jonh Willey and Sons, 1993.

[2] Y.Saad, *Iterative methods for sparse linear systems*, PWS, 1996.

[3] I. S. Duff, A. M. Erisman e J. K. Reid, *Direct methods for sparse matrices*, Clarendon Press, Oxford, 1986.

[4] A.N. Krylov, "On the numerical solution of the equation by which in technical questions frequencies of small oscillations of material systems are determined", *Izvestija AN SSSR (News of Academy of Sciences of the USSR),* vol. VII, no. 4, pp 491-539, 1931.

[5] I. S. Duff, "ME 28: a sparse unsymmetric linear equation solver for complex equations", *ACM Trans. on Math. Soft.*, vol. 7, no. 4, pp. 505-511, 1981.

[6] H.A. Vorst, "Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput.*, vol. 13(2), pp. 631-644, 1992.

[7] *http://www-users.cs.umn.edu/~saad/software/SPARSKIT/sparskit.html*.

[8] S. D. Conte, *Elementos de Análise Numérica*, Globo, 1977.

[9] A. Jennings, *Matrix Computations*, Jonh Willey and Sons, 2ed., 1992.

[10] H.M.Markowitz, "The elimination form of the inverse and its application to linear programming", *Math. Comp*., 34, pp. 473-498, 1957.

[11] N.Sato e W.F.Tinney, "Techiniques for exploiting the sparsity of the network admittance matrix", *IEEE Trans. on Power App. and Sys.*, vol. 12, n. 82, pp. 944 – 950, 1974.

[12] W.F. Tinney e J.W. Walker, "Direct solutions of sparse network equations by optimally ordered triangular factorization", *Proceedings of the Institute of Electr. and Electron. Engn.*, vol. 11, n. 55, pp. 1801-1809, 1967.

[13] W.F. Tinney, W.L. Powell e N.M. Peterson, "Sparse-oriented network reduction", *IEEE Trans. on Power App. and Sys.*, vol. PA-93, n. 1, 1974.

[14] I.S. Duff e J.K. Reid, "A comparison of sparsity orderings for obtaining a pivotal sequence in gaussian elimination", *J. of the Inst. Math. its Apl.*, v. 14, pp. 281-291, 1974.

[15] A. M. Erisman e W.F. Tinney, "On computing certain elements of the inverse of a sparse matrix", *Communications of ACM*, vol 18, n. 3, pp. 177-179, 1975.

[16] B.M. Irons, "A frontal solution program for finite-element analysis", *J. Inst. Maths. Appics*., n. 9, pp. 351-361, 1970.

[17] K. Zollenkopf, *Bi-factorization – basic computational algoritm and programming techniques*, Academic Press, pp. 75-79.

[18] S.C. Eisenstat, M. H. Schultz e A.H. Sherman, "Algorithms and data structures for sparse symmetric gaussian elimination", *SIAM J. Sci. Stat. Comp.*, vol. 2, n. 2, pp.225-237, 1981.

[19] R. Barret, M. Berry, T.F.Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine e H.A. Vorst , *Templates for the solution of linear systems: building blocks for iterative methods* , SIAM, 1994.

[20] T. Chan e H.A.Vorst, *Linear systems solvers: sparse iterative methods parallel numerical algorithms*, ICASE/LaRC Interdisciplinary Series in Science and Engineering, Volume 4, Kluwer Academic, Dordecht, pp91-118,1997.

[21] Y.Saad e H.A. Vorst, "Iterative solution of linear systems in the 20th Century", *J. Comp. Appl. Math.*, 123, pp. 1-33, 2000.

[22] H.S. Varga , Matrix iterative analysis, Prentice Hall, Englewood Cliffs, NJ, 1992.

[23] L.F.Richardson, "The approximate arithmetical solution by finite differences of physical problems involving differential equations with application to the stress to a masonry dam", *Philos. Trans. Roy. Soc. London ser. A*, pp. 307-357, 1910.

[24] S.Frankel, "Convergence rates of iteratives treatments of partial differential equations", *MTAC*, pp. 65-75, 1950.

[25] D. M.Young, *Iterative methods for solving partial diferential equations of elliptic type*, PhD thesis, Harvard University, Cambridge (MA USA), 1950.

[26] M.R. Hestenes, E.L.Stiefel, "Methods of conjugate gradients for solving linear systems", *J. Res. Nat. Bur. Stand, Sec. B*, vol. 49, pp. 409-436, 1952.

[27] C. Lanczos , "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators", *J. Res. Nature Stand.*, vol. 14, pp. 255-282, 1950.

[28] R. Flecher, " Conjugate gradients methods for indefinite systems", G.A. Watson editor, *Proceedings of the Dundee Biennal Conference on Numerical Analysis 1974*, Spring Verlag, pp 73-89, 1975.

[29] P.Sonneveld, "CGS: a fast Lanczos-type solver for nonsymetric linear systems", *SIAM J. Sci. Statist. Comput.*, vol 10, pp. 36-52, 1989.

[30] M.L.Bittncourt e R.A. Feijóo, "Análise comparativa de métodos diretos e iterativos para a solução de sistemas de equações", *Rev. Intern. de Mét. Num. para Calc. e Dis. en Ingeniería*, vol. 13, pp 123-148, 1997.

[31] M.Ruggiero e V.L.Lopes, *Cálculo numérico, aspectos teóricos e computacionais*, Makron Books, pg. 176, 2ed., 1997.

[32] W. Kahan, "Gauss-Seidel methods of solving large systems of linear equations", PhD thesis, University of Toronto, 1958.

[33] A. Greenbaum, *Iterative methods for lolving linear systems*, SIAM, 1997.

[34] M.Engeli, T. Ginsbug, H. Rutshauser e E. Stiefel, "Refined iterative methods for computation of the solution and the eigenvalues of self-adjoint boundary value problems", Birkhauser, Basel/Stuttgart, 1959.

[35] B.N.Parllet, D.R.Taylor e Z.A. Liu,"A look-ahead Lanczos algorithm for unsymmetric matrices", *Math. of Comp.*, vol. 44, pp.105-124,1985.

[36] Tony F. Chan e Henk A. Van der Vorst, *Aproximate and Incomplete Factorizations, Parallel Numerical Algorithms*, ICASE/LaRC Interdisciplinary Series in Science and Engineering, vol.4, Kluwer Academic, Dordecht, pp.167-202, 1997.

[37] S.T. Barnard e M.J. Grote, "A block version of the SPAI precondictioner", *Proc. 9th SIAM Conf. on Parall. Process. for Sci. Comp.*, março, 1999.

[38] N.I. Buleev, "A Numerical method solution of two-dimensional and three-dimensional equations of diffusion", *Math. Sb.*, vol. 51, pp. 227-238, 1991.

[39] T.A. Olifhant, “An Implicit Numerical Methods for solving two-dimensional time dependent diffusion problems”, *Quart. Appl. Math.*, vol. 19, pp. 221-229, 1961.
[40] T.A. Olifhant, “An Extrapolation process for solving linear systems”, *Quart. Appl. Math.*, vol. 20, pp. 257-267, 1962.
[41] D.J. Evans, “The use of preconditioning in iterative methods for solving linear equations with symmetric positive matrices”, *J. Inst. Maths. Applics.*, vol. 4, pp. 295-314, 1968.
[42] H. Stone, “Iterative solution of implicit aproximations of multidimensional partial differential equations”, *SIAM J. Numer. Anal.*, vol. 5, pp. 530-558, 1968.
[43] T. Dupont, R. Kendall e H. Rachford, “An approximate factoring procedure for solving self-adjoint ellipitic difference equations”, *SIAM J. Numer. Anal.*, vol. 5, pp. 559-573, 1968.
[44] J.A. Meijerink DE H.A. van der Vorst , “An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix”, *Math. Comp.*, vol. 31, pp. 148-162, 1977.
[45] D.S. Kershaw; “The incomplete Cholesky-conjugate gradient method for the iterative solution of systems linear equations”, *J. Comp. Phys.*, vol. 11, pp. 767-793, 1990.
[46] J. Appleyard e I.Cheshire, “Nested factorizations in Reservoil”, Simulation Synposium of the SPIE”, paper 12264, 1983.
[47] MATLAB*, http://www.mathworks.com/.*
[48] J. P. Berenger, “A perfect Mached Layer for the absortion of electromagnetic waves”, *J. Comput. Physics*, no. 114, pp.185-200, 1994.
[49] H.F. Pinheiro, *Método vetorial da propagação de feixes ópticos baseado em elementos finitos*, tese de doutorado, DMO/FEEC/Unicamp, 2000.
[50] M.A.Corrêa, *Ambiente MATLAB – elementos finitos para eletromagnetismo*, tese de mestrado, DMO/FEEC/UNICAMP, 2001.
[51] J.D.Kraus, *Electromagnetcs*, McGraw Hill, 4ª ed., 1992.
[52] Geometry and Data, “Internacional center for numerical methods in engineering”, 2002. http://gig.cimme.upc.es/.
[53] V.F.R. Esquerre, *Modelagem de estruturas fotônicas por elementos finitos 2D no domínio do tempo*, tese de doutorado, DMO/FEEC/Unicamp, 2003.
[54] M. Koshiba, Y. Ysuji e M. Hikari, “Time-domain beam propagation method and its application to photonic crystal circuits”, *J. of Light. Tech*., vol. 18, no. 1, Janeiro 2000.
[55] J.P. da Silva, *Simulação por elementos finitos da propagação de feixes ópticos em estruturas fotônicas*, tese de doutorado, DMO/FEEC/Unicamp, 2003.
[56] A. C. Newell e J. V. Moloney, *Nonlinear Optics*, Addison Wesley, pp. 43,1992.
[57] K.Sitoh e M.Koshiba, “Approximate scalar finite-element beam-propagation method with perfectly matched layers for anisotrosopic optical waveguides”, *IEEE J. of Light. Tech.,* vol.19, no. 5, pp. 786-792, Maio 2001.
[58] B.Ball, F.Walter, Steve Shah, S. Veeraraghaven, T.Bohlsen e D. Pitts, *Using Linux*, Que, 1998.
[59] G.A.Geist, A.Beguelin, Jack Dongarra, W. Jiang, R. Manchek e Vaidy Sunderam, *PVM - Parallel Virtual Machin, A users’guide and tutorial for networked parallel computing,* MIT Press, 1994.

[60] G.A.Geist, J.A, Kohl e P.M. Papadopoulos, “PVM and MPI: a comparison of features”, *Calculateurs Paralleles*, vol.8, no. 2, 1996.
[61] S.V. Wiel, D. Nathanson e D.Lilja, “Performance and program complexity in comtemporary networked-based parallel computing systems”, *Technical Report HPPC-96-02,* University of Minesota, 1996.
[62] M. Beck, J.J. Dongarra, G.E. Fagg, G.A. Geist, P.Gray, J.Kohl, M.Migliardi, K. Moore, T. Moore, P.M. Papadopoulos, S.L. Scott e V. Sunderam, “HARNESS: a next generation distribuited virtual machine”, *Future Generation Systems*, no.15, pp. 571-582, 1999.
[63] http://www.csm.ornl.gov/harness/.

Apêndice A - Tabelas de testes

Nas tabelas temos: *N* - dimensão da matriz, *Nz* - valores não nulos, *NZ_LU* valores não-nulos das matrizes L e U, *tbcgs* - tempo de solução do biconjugado estabilizado, tilu - tempo da decomposição incompleta, *tbi* - soma de *tbcgs* e *tilu*, *lfil* e *droptol* são parâmetros da decomposição incompleta, *iter* - números de iterações necessárias para convergência.

Guia de onda com formulação vetorial e PML

Invariante sem ordenamento:

N	*NZ*	*NZ_LU*	*tbcgs*	*tilu*	*tbi*	*lfil*	*droptol*	*iter*
86	972	285	0	0	0	5	0,1	11,5
98	1100	5856	0	0,01	0,01	98	1E-8	3,5
202	2556	780	0	0	0	5	0,1	13,5
500	6608	2075	0	0,01	0,01	5	0,1	12,5
964	13104	3955	0,01	0,01	0,02	5	0,1	14,5
2040	28082	28657	0,01	0,04	0,05	150	0,01	7,5
5071	69993	44806	0,06	0,14	0,2	5	1E-7	9,5
8607	119247	65801	0,14	0,12	0,26	5	0,01	9,5
9726	134600	30162	0,2	0,06	0,26	50	0,1	14,5
13070	181420	37078	0,31	0,08	0,39	100	0,1	15,5

Tabela A.1 - Resultados para guia isotrópico, simulação invariante sem ordenamento

Variante sem ordenamento:

N	*NZ*	*NZ_LU*	*tbcgs*	*tilu*	*tbi*	*lfil*	*droptol*	*iter*
86	972	285	0	0	0	5	0,1	11,5
98	1100	5856	0	0	0	100	1E-8	3,5
202	2556	780	0	0	0	5	0,1	13,5
500	6608	2104	0	0	0	100	0,1	13,5
964	13104	3955	0,01	0,01	0,02	5	0,1	14,5
2040	28082	7987	0,04	0,01	0,05	150	0,1	14,5
5071	69993	16503	0,08	0,04	0,12	15	0,1	14,5
8607	119247	26239	0,18	0,06	0,24	5	0,1	14,5
9726	134600	30162	0,2	0,06	0,26	50	0,1	14,5
13070	181420	37078	0,31	0,08	0,39	100	0,1	15,5

Tabela A.2 - Resultados para guia isotrópico, simulação variante sem ordenamento

Invariante com ordenamento:

N	*NZ*	*NZ_LU*	*tbcgs*	*tilu*	*tbi*	*lfil*	*droptol*	*iter*
86	972	343	0	0	0	5	0,1	11,5
98	1100	402	0	0	0	5	0,1	10,5
202	2556	862	0	0	0	5	0,1	13,5
500	6608	8968	0	0,01	0,01	100	1E-3	5,5
964	13104	4359	0,01	0,01	0,02	5	0,1	13,5
2040	28082	24418	0,02	0,03	0,05	50	0,01	7,5
5071	69993	42501	0,07	0,12	0,19	5	1E-6	9,5
8607	119247	71997	0,15	0,22	0,37	5	1E-8	10,5
9726	134600	81081	0,19	0,26	0,45	5	1E-8	11,5
13070	181420	40613	0,31	0,1	0,41	5	0,1	15,5

Tabela A.3 - Resultados para guia isotrópico, simulação invariante com ordenamento

Variante com ordenamento:

N	*NZ*	*NZ_LU*	*tbcgs*	*tilu*	*tbi*	*lfil*	*droptol*	*iter*
86	972	343	0	0	0	5	0,1	11,5
86	972	343	0	0	0	5	0,1	11,5
202	2556	862	0	0	0	5	0,1	13,5
500	6608	3749	0	0	0	5	0,01	7,5
964	13104	4359	0,01	0,01	0,02	5	0,1	13,5
2040	28082	8861	0,04	0,01	0,05	50	0,1	15,5
5071	69993	19423	0,09	0,03	0,12	20	0,1	14,5
8607	119247	31896	0,16	0,06	0,22	15	0,1	13,5
9726	134600	36063	0,2	0,07	0,27	15	0,1	14,5
13070	181420	40613	0,31	0,1	0,41	5	0,1	15,5

Tabela A.4 - Resultados para guia isotrópico, simulação variante com ordenamento

Caixa condutora 2D

Invariante sem ordenamento:

N	NZ	NZ_LU	tbcgs	tilu	tbi	lfil	droptol	iter
204	1006	1125	0	0	0	5	0,1	11,5
962	5772	38700	0	0,01	0,01	50	1E-8	1
5872	38702	651820	0,03	0,37	0,4	5872	1E-8	1
10333	69121	1,54074E6	0,06	1,06	1,12	10333	1E-8	1
47192	323380	1,54307E7	1,06	31	32,06	47192	1E-8	1,5
76644	527710	3,22067E7	1,47	64,57	66,04	76644	1E-8	1,5
100276	691864	4,84187E7	2,18	111,75	113,93	100276	1E-8	1,5
119066	822374	6,37105E7	2,82	156,04	158,86	119066	1E-8	1,5
159817	1,10598E6	4,53852E7	11,34	82,65	93,99	150	1E-6	7,5
199010	1,3788E6	4,28102E7	14,59	57,18	71,77	199010	1E-5	9,5
255548	1,77256E6	5,54162E7	22,87	75,271	98,141	255548	1E-5	11,5

Tabela A.5 - Resultados para caixa condutora, simulação invariante sem ordenamento

Variante sem ordenamento:

N	NZ	NZ_LU	tbcgs	tilu	tbi	lfil	droptol	iter
204	1006	1125	0	0	0	5	0,1	11,5
962	5772	6591	0	0	0	962	0,1	22,5
5872	38702	86572	0,13	0,02	0,15	50	0,01	26,5
10333	69121	155271	0,31	0,04	0,35	150	0,01	34,5
19222	130142	371713	0,67	0,15	0,82	10	1E-3	33,5
47192	323380	1,351E6	2,29	0,58	2,87	15	1E-3	35,5
76644	527710	2,84737E6	4,45	1,31	5,76	76644	1E-3	34,5
100276	691864	3,75688E6	6,71	1,56	8,27	50	1E-3	39,5
119066	822374	3,45885E6	8,9	1,54	10,44	15	1E-3	53,5
159817	1,10598E6	5,73137E6	13,48	2,56	16,04	20	1E-3	50,5
199010	1,3788E6	7,15774E6	18,01	3,271	21,28	20	1E-3	54,5
255548	1,77256E6	7,48219E6	29,061	3,38	32,441	15	1E-3	80,5

Tabela A.6 - Resultados para caixa condutora, simulação variante sem ordenamento

Invariante com ordenamento:

N	NZ	NZ_LU	tbcgs	tilu	tbi	lfil	droptol	iter
204	1006	5885	0	0,01	0,01	204	1E-8	1
962	5772	80669	0	0,07	0,07	962	1E-8	1,5
5872	38702	911675	0,05	1,77	1,82	5872	1E-8	1,5
10333	69121	1,50919E6	0,11	2,9	3,01	10333	1E-7	2
19222	130142	4,18227E6	0,28	11,96	12,24	19222	1E-8	2
47192	323380	1,35472E7	0,87	57,07	57,94	47192	1E-8	2
76644	527710	2,4358E7	1,94	117,34	119,28	76644	1E-8	2,5
100276	691864	3,26038E7	2,61	174,45	177,06	100276	1E-8	2,5
119066	822374	3,69692E7	2,96	189,78	192,74	119066	1E-8	2,5
159817	1,10598E6	5,30456E7	5,81	640,9	646,71	159817	1E-8	2,5
199010	1,3788E6	4,89536E7	8,041	256,621	264,662	199010	1E-7	3,5
255548	1,77256E6	2,29948E7	24,891	66,391	91,281	150	1E-5	17,5

Tabela A.7 - Resultados para caixa condutora, simulação invariante com ordenamento

Variante com ordenamento:

N	NZ	NZ_LU	tbcgs	tilu	tbi	lfil	droptol	iter
204	1006	3482	0	0	0	20	1E-8	3,5
5872	38702	102030	0,15	0,04	0,19	100	0,01	26,5
10333	69121	315342	0,23	0,18	0,41	50	1E-3	14,5
19222	130142	611053	0,66	0,35	1,01	50	1E-3	19,5
47192	323380	1,5474E6	2,36	0,82	3,18	50	1E-3	32,5
76644	527710	2,66741E6	5,46	1,68	7,14	100	1E-3	38,5
100276	691864	3,5036E6	7,92	2,17	10,09	100276	1E-3	42,5
119066	822374	4,10755E6	9,56	2,61	12,17	150	1E-3	43,5
159817	1,10598E6	5,28517E6	16,23	3,72	19,95	50	1E-3	57,5
199010	1,3788E6	6,9629E6	35,15	5,9	41,051	150	1E-3	61,5
255548	1,77256E6	6,20217E6	55,971	5,891	61,862	20	1E-3	104,5

Tabela A.8 - Resultados para caixa condutora, simulação variante com ordenamento

Guia PBG

Invariante sem ordenamento:

N	NZ	NZ_LU	tbcgs	tilu	tbi	lfil	droptol	iter
11406	79300	490060	0,07	0,5	0,57	150	1E-6	1,5
31051	216095	2,26573E6	0,25	2,74	2,99	150	1E-7	1,5
55942	389980	6,1869E6	0,53	9	9,53	55942	1E-8	1,5
72273	504117	8,94215E6	0,74	14,15	14,89	72273	1E-8	1,5
98625	688445	1,39154E7	1,09	24,59	25,68	100	1E-8	1,5
116598	814084	1,72637E7	1,32	31,47	32,79	150	1E-8	1,5
146889	1,02594E6	2,38652E7	1,75	46,87	48,62	150	1E-8	1,5
205298	1,43457E6	2,31287E7	3,29	37,4	40,69	150	1E-6	2,5
254172	1,77647E6	3,09942E7	4,24	52,399	56,64	150	1E-6	2,5

Tabela A.9 - Resultados para guia canal, simulação invariante sem ordenamento

Variante sem ordenamento:

N	NZ	NZ_LU	tbcgs	tilu	tbi	lfil	droptol	iter
11406	79300	112309	0,13	0,09	0,22	150	0,01	7,5
31051	216095	340144	0,46	0,27	0,73	150	0,01	9,5
55942	389980	638593	1,1	0,51	1,61	100	0,01	12,5
72273	504117	841486	1,65	0,65	2,3	10	0,01	14,5
98625	688445	1,18295E6	2,43	0,92	3,35	20	0,01	15,5
116598	814084	1,40809E6	2,89	1,09	3,98	100	0,01	15,5
146889	1,02594E6	3,8145E6	3,12	3,27	6,39	146889	1E-3	8,5
205298	1,43457E6	5,54011E6	5,01	4,8	9,81	50	1E-3	9,5
254172	1,77647E6	6,79194E6	6,67	6,35	13,02	15	1E-3	10,5

Tabela A.10 - Resultados para guia canal, simulação variante sem ordenamento

Invariante com ordenamento:

N	NZ	NZ_LU	tbcgs	tilu	tbi	lfil	droptol	iter
11406	79300	494155	0,08	0,68	0,76	150	1E-5	2
31051	216095	3,93642E6	0,3	11,28	11,58	31051	1E-8	1,5
55942	389980	8,31062E6	0,59	26,9	27,49	55942	1E-8	1,5
72273	504117	6,45329E6	0,93	14,89	15,82	72273	1E-6	2,5
98625	688445	1,66979E7	1,13	61,39	62,52	98625	1E-8	1,5
116598	814084	2,04429E7	1,38	78,03	79,41	116598	1E-8	1,5
146889	1,02594E6	1,48086E7	2,03	37,94	39,97	146889	1E-6	2,5
205298	1,43457E6	2,17963E7	3,01	59,449	62,459	205298	1E-6	2,5
254172	1,77647E6	2,76111E7	3,779	76,52	80,299	254172	1E-6	2,5

Tabela A.11 - Resultados para guia canal, simulação invariante com ordenamento

Variante com ordenamento:

N	NZ	NZ_LU	tbcgs	tilu	tbi	lfil	droptol	iter
11406	79300	148689	0,14	0,13	0,27	11406	0,01	6,5
31051	216095	427514	0,65	0,39	1,04	15	0,01	10,5
55942	389980	831254	1,8	0,75	2,55	20	0,01	15,5
72273	504117	1,9264E6	1,28	2,08	3,36	100	1E-3	6,5
98625	688445	2,6997E6	2,03	2,97	4,98	100	1E-3	7,5
116598	814084	3,22643E6	2,42	3,56	5,98	116598	1E-3	7,5
146889	1,02594E6	4,11464E6	3,45	4,67	8,12	100	1E-3	8,5
205298	1,43457E6	5,82684E6	5,46	6,71	12,17	205298	1E-3	9,5
254172	1,77647E6	7,27477E6	7,51	8,59	16,1	100	1E-3	10,5

Tabela A.12 - Resultados para guia canal, simulação variante com ordenamento

Apêndice B - LINUX

Linux é um sistema operacional, relativamente novo, que está em fase de consolidação. Sua principal vantagem é o fato de possuir códigos abertos, bem como várias versões gratuitas, como Red Hat e Mandrake e, ainda, ser robusto com grande estabilidade. Neste apêndice não se pretende aprofundar muito neste sistema, mas apresentar informações básicas necessárias para desenvolver o trabalho de leitura de "papers", compilação, gerenciamento e compactação de arquivos.

A utilização deste sistema se fez necessária, tendo em vista a sua intensa utilização na área científica, bem como a do sistema UNIX. Além da computação serial largamente difundida, várias ferramentas de computação de alto-desempenho são utilizadas em ambiente linux, tais com MPI e PVM; desta forma, a inserção do cientista de cálculo numérico neste sistema é fundamental.

Utilizamos para executar nossos trabalhos o linux Mandrake 7.2 e o Red Hat 8.0; eventualmente, as considerações neste apêndice estão relacionadas ao Mandrake 7.2, embora estas duas versões sejam muito parecidas; eventualmente poderá ocorrer algumas diferenças.

O Mandrake está disponível no site oficial:

www.mandrake.com

O RedHat está no site:

www.redhat.com

Nestas páginas é possível obter maiores informações e a disponibilidade de programas e "drivers" para hardware.

Instalação

O Linux exige que crie se, no mínimo, três partições no disco rígido:

SWAP, HOME
/

A partição *Swap* é o espaço de troca do disco, que funciona como uma extensão da memória RAM. Esta partição é similar à memória virtual do Windows.

A partição *Home* é o espaço do disco rígido onde grava-se os arquivos e informações de cada usuário.

A partição principal / é onde grava-se o sistema operacional e os programas.

No Mandrake, o particionamento é facilitado pelo Diskdrake, permitindo o particionamento em um ambiente gráfico.

O ambiente gráfico é chamado de X. Este ambiente é o responsável pelo gerenciamento das janelas que permite uma aproximação ao Windows. Porém é possível utilizá-lo sem o X, tornando-o mais hostil aos iniciantes, pois exige conhecimentos mais avançados sobre o Linux.

O bom funcionamento do ambiente X está diretamente relacionado à boa configuração dos hardwares responsáveis pelas saídas de dados tais como: monitor e placa de vídeo. Portanto, para um sistema estável, deve-se observar principalmente se o *driver* da placa de vídeo está disponível no sistema linux e se o monitor e a resolução foram configurados de maneira correta.

Trabalhando no LINUX

O modo console, citado acima, pode ser simulado dentro do X. No console é onde podemos compilar e executar programas em Fortran, C, C++, etc. Também tem-se acesso a todo sistemas de arquivos, permitindo gerenciá-lo através do "logon" de super-usuário e permite a configuração do sistema operacional: instalação e remoção de programas, adicionar e remover usuários, configurar rede local, etc.

No Linux não é necessário desfragmentar, organizar os dados de tempos em tempos; o disco rígido deste sistema operacional grava os dados de maneira já otimizada.

Abaixo citaremos alguns comandos úteis, para serem utilizados no simulador do modo console:

Editando programas:

>> nedit teste .f & → abre o arquivo teste.f no editor nedit e libera o Console

No editor de texto Nedit é possível editar vários tipos de programas, os *.m do matlab, c, c++, fortran, etc. Porém a opção de mostrar as palavras chaves em destaque e a linguagem, precisa ser configurada pelo usuário e salva. Existem outros editores, como o sofisticado Xemacs ou o editor de texto Pico, que roda no próprio console, mas sem muitos recursos de edição de programas.

Compilando programas:

>> f77 –O4 –o *.exe principal.f <subrotinas> <objetos> → executáveis
>> f77 –O4 –c <subrotinas> → objetos
Executando o programa:
>> **.**/ *.exe
ou ainda
>> *.exe

A opção -*O4* indica que será executada uma compilação otimizada; esta opção é mais lenta para compilar, porém os executáveis gerados são mais eficientes.

Os executáveis podem assumir qualquer nome e extensão; acima utilizamos a extensão .exe, apenas por ser sugestiva. Ao executar os programas, a utilização do *ponto barra* dependerá da configuração do arquivo ".bash_profile", que localiza-se no "home" do usuário. O arquivo ".bash_profile" permite carregar informações individuais para cada usuário; as informações gerais são carregadas pelo arquivo "profile", que localiza-se na pasta /etc e poderá ser alterada apenas pelo super-usuário .

Gerenciando Arquivos

Nas informações abaixo, utilizamos as palavras origem, pois não é necessário estar no diretório para alteração dos arquivos no qual o arquivo ou diretório que se queira acessar, ou em um imediatamente anterior.

>>cd	→ entra na pasta "home" do usuário
>>cd <destino>	→ entra no diretório
>> ls –l	→ lista arquivos com detalhes
>>ls –la → lista todos arquivos inclusive os ocultos que iniciam com ponto.	
>>cp <origem> <destino>	→ cópia de arquivos
>>rm <origem>	→ remoção de arquivos
>>rm -r <origem>	→ remoção de diretórios
>>mkdir <destino>	→ cria-se um diretório
>>locate <nome>	→ busca de arquivo ou diretório

Voltando ao arquivo ".bash_profile", para visualizá-lo devemos digitar:

>>cd	→ vai para o "home" do usuário
>>ls –la	→ lista todos os arquivos
>>nedit .bash_profile &	

A linha de comando *>>nedit .bash_profile* irá abrir o arquivo no editor de textos Nedit; a linha do "PATH" deverá ser alterada conforme indica o quadro abaixo:

```
Antes:
        PATH=$PATH:$HOME/BIN
DEPOIS
        PATH=$PATH:$HOME/BIN:./
```

Este mesmo procedimento pode ser feito no arquivo /etc/profile, e esta alteração será valida para todos os usuários. Quando instalamos programas, devemos indicar aonde está o executável; algumas instalações faz isto automaticamente, outras não, sendo necessário indicá-lo manualmente, da mesma forma que o **./** .

No Mandrake é possível gerenciar os arquivos em um ambiente gráfico, tais como o XWC ou no Konqueror; estes gerenciadores são muito parecidos com o Explorer, do Windows.

Existem outros comandos que são úteis, que serão apresentados logo a seguir.

Tutoriais sobre comandos:

>>man <nome do comando> → retorna a documentação do comando

Atualização do banco de dados utilizado pelo comando locate:

>>su → "logon" como super-usuário
>>password: → senha de super-usuário ("root")
>>slocate –u →inicia a atualização

Verificação do espaço em cada partição:

>>df → retorna informações sobre as partições

Criando bibliotecas:

>>ar –r <nome> <arquivos>
>>ar –rv teste .a *.o → cria a biblioteca teste.a adicionado todos os arquivos *.o

A formação de bibliotecas (*.a) pode facilitar na hora da compilação de programas, pois não será necessário compilar todas as subrotinas novamente, a cada vez que o programa principal for recompilado.

>> f77 –O4 –o *.exe principal.f <subrotinas> <Bibliotecas *.a> → executáveis

Compactando e extraindo arquivos *.gz, *.tgz:

>>gunzip <nome> → descompacta arquivo
>>gzip <nome> → compacta arquivo
>>tar –xvvzf teste.tgz → descompacta arquivo *teste.tgz*
>>tar –cvvzf teste.tgz teste/ teste.a → compacta o diretório *teste* e o arquivo *teste.a* em um arquivo de nome *teste.tgz*

O comando *c*, em –cvvf, significa criar arquivos, *x* significa extrair arquivos, z informa que o arquivo deverá ser compactado ou descompactado, dependendo do primeiro comando indicar criando(*c*) arquivo ou extraindo(*x*) arquivo.

A utilização do comando tar é muito útil, pois é possível, desta forma compactar diretórios e arquivos.

Extraindo e criando arquivos de extensão tar:

>>tar –cvvf teste.tar teste/ teste.a → cria-se o arquivo teste.tar adicionando o diretório teste e o arquivo teste.a
>>tar –xvvf teste.tar → extração dos arquivos e/ou diretórios contidos no arquivos teste.tar

Visualizando arquivo *.pdf e *.ps:

>> xpdf paper.pdf → abre o programa Xpdf e o arquivo paper.pdf
>>gv paper.ps → abre o programa GhostView e o paper .os

Uma boa referência, que auxiliou muito na inicialização a este sistema operacional, foi: ”Using Linux”, da editora QUE, pelos autores Bill Ball e outros [58]. O auxílio de um livro poderá contribuir significativamente na aprendizagem deste sistema, que possui muitas peculiaridades, as quais um usuário iniciante deve tomar conhecimento.

Apêndice C - Ambiente Paralelo

A computação paralela é desenvolvida com base na troca de informações entre os processadores do ambiente; para gerenciar este fluxo de informações temos os softwares PVM e MPI. Chamamos de ambiente um conjunto de máquinas (paralelas ou seriais) em uma rede**.**

PVM

O projeto PVM (Parallel Virtual Machine) teve início em 1989 no Laboratório Nacional *Oak Ridge*; a versão 1.0 foi construída pelos pesquisadores Vaidy Sunderam e Al Geist. A versão 2.0 foi programada na Universidade de Tennesse e disponibilizada em 1991. Em 1993 finalizou-se a versão 3.0, atualmente está na versão 3.3 [59].

O PVM trabalha com a abstração da rede de computadores ser uma grande máquina virtual, com memória distribuída, e seus recursos computacionais são gerenciados como um único computador paralelo. Esta abstração permite, ao PVM, duas características: heterogeneidade e portabilidade.[60]

A heterogeneidade de uma rede pode ser referente aos seguintes fatores: Arquitetura, Formato dos dados, Velocidade dos computadores, Carga de trabalho em cada máquina, Carga de informações na rede.

Esta heterogeneidade não existe em computadores paralelos. Nos computadores paralelos, os recursos computacionais (processadores, memórias, disco rígido, etc) são idênticos e o software é comum a todos estes recursos.

O PVM é um ambiente que permite o gerenciamento das comunicações entre computadores em uma rede heterogênea, visualizando-a como se fosse um único computador paralelo.

O pacote PVM está disponível gratuitamente na Internet:

http://www.netlib.org/pvm3

Também está disponível, on-line, neste site o livro PVM: Parallel Virtul Machine – *A Users' Guide and Tutorial for Networked Parallel Computing*.

MPI

O MPI surgiu nos anos 90, após encontros entre indústrias e pesquisadores nos anos de 1993 e 1994, com a finalidade de que os fabricantes de MPP ("Massively Parallel Processor") tivessem um padrão de "message-passing", desta forma melhorando a portabilidade entre as diversas máquinas.

Em 1995 surge o MPI-2, corrigindo problemas de portabilidade [60], e incluindo novas funções de comunicações; neste ponto, o MPI-2 aumenta significativamente seus recursos de comunicação em relação ao PVM. No entanto, ainda não existe no MPI –2 recursos de aplicações de tolerância a falhas.

Existem várias implementações de domínio público do MPI:

MPICH–http:\\www-unix.mcs.anl.gov/mpi/mpich

LAMMPI–http:\\www.lam-mpi.org

A Universidade de Tenesse tem, também, um bom suporte ao usuário do MPICH com tutoriais e livro na página do mpich.

A Universidade de Notredame oferece curso de MPI via internet, bem como fórum de discussão e livros on-line no endereço do lam-mpi.

PVM ou MPI

"PVM ou MPI", o trabalho feito por Geist e Kohl [60], nos dá algumas ferramentas para a melhor escolha; porém ambos possuem suas vantagens e desvantagens, como por exemplo MPI possui maior número de funções facilitando, desta forma, a programação; porém não fornece recursos em caso de falhas e o PVM sim. As versões do MPI não são compatíveis, enquanto as do PVM sim. O maior número de funções do MPI permite a princípio um melhor desempenho nos grandes computadores paralelos (MPP), mas isto não ocorreu nos estudos apresentados no trabalho feito por S. Vander e outros [61].

Além do PVM e MPI, existe um novo ambiente chamado HARNESS (Heterogeneous Adaptable Reconfigurable Networked Systems) [62]; aprimorando o PVM temos o DVM (Distributed Virtual Machine) que consiste em uma rede de computadores paralelos virtuais. O Harness oferece serviços adaptáveis ao usuários e reconfiguráveis. Este projeto é desenvolvido juntamente com os pesquisadores que desenvolvem o PVM, atualmente ele suporta tanto o PVM quanto o MPI [63].

Apêndice D - Elementos Finitos

Faremos, nesta seção, uma breve exposição do método dos elementos finitos [1], e um pouco sobre a história deste método. A formalização do método será feita para o caso bidimensional.

Os problemas físicos, em engenharia e matemática muitas vezes não possuem soluções analíticas, ou são muito caras para serem calculadas de maneira exata. Com os crescentes avanços computacionais, as técnicas de discretização que se baseiam no paradigma de divisão e conquista, tornam-se cada vez mais populares; algumas destas técnicas de discretização de um espaço matemático contínuo são: o método dos elementos finitos e o método das diferenças finitas.

Neste trabalho, o modelo de discretização do espaço contínuo foi feito pelo método dos elementos finitos, passando de um espaço com infinitos graus de liberdade para um espaço com graus de liberdade finito. A aplicação do método de elementos finitos ocorre nas resoluções de equações diferenciais parciais que regem o problema físico, gerando um sistema de equações lineares geralmente esparsos. É de fundamental importância a resolução destes sistemas de maneira eficiente, pois este representa o gargalo do método.

Esta metodologia que discretiza um problema em partes, para depois reunir as soluções parciais destas partes, formando a solução do problema como um todo, tem seus primórdios na matemática egípcia, em 1800 aC, bem como nos estudos de Archimedes, nos estudos realizados sobre círculos, em 250 aC .

Na Figura D1 mostra-se o processo de discretização, para cálculo do número PI.

Figura D1 - Discretização do círculo de raio r

Os lados do polígono, na Figura D2, mostram a discretização da circunferência representada pelos segmentos 1-2-3-4-5-6-7-8-1. Ao calcularmos o comprimento do lado de cada segmento, e depois somá-los, estamos empregando a idéia de elementos finitos numa discretização de um espaço contínuo e posterior união dos resultados obtidos no espaço discreto.

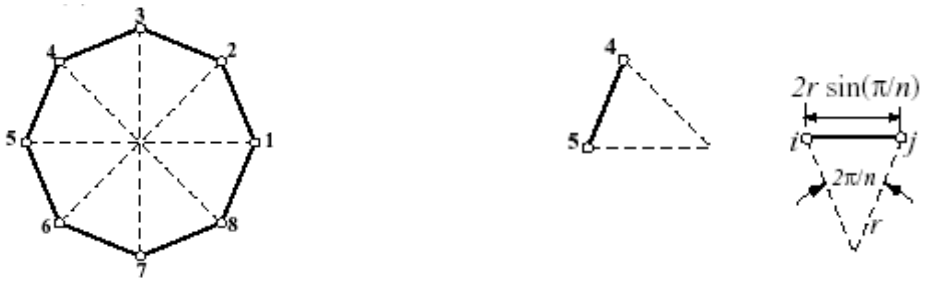


Figura D2 - Discretização do círculo, em detalhes

O método de elementos finitos é uma técnica numérica de solução de problemas de valores de contorno. Foi proposto em 1940 e iniciada sua aplicação em 1950, para elaborar projeto de aviões.

O método de elementos finitos tem sido aplicado extensivamente em problemas estruturais, embora aumenta-se continuamente as aplicações em outras áreas, como em problemas de eletromagnetismo.

O método consiste em discretizar o domínio a ser analisado em uma malha formada por elementos triangulares, conforme a Figura D3; com isso, é possível modelar as mais variadas formas. No caso das diferenças finitas, os elementos são retangulares, o que impossibilita a discretização de superfícies curvas.

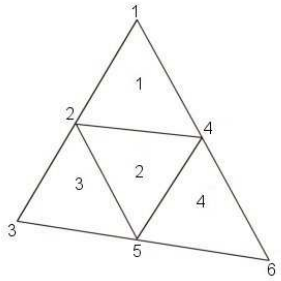


Figura D3 - Malha com elementos triangulares

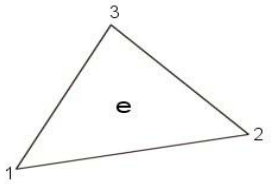


Figura D4 - Elemento triangular

Na discretização de um domínio Ω (bidimensional) é preciso seguir algumas regras básicas: os elementos devem estar conectados somente pelos nós (vértices) do elemento vizinho, deve-se evitar elementos com ângulos internos muitos pequenos e não deve ocorrer sobreposição, ou buracos, entre os elementos.

Uma vez discretizado o domínio, é necessário aproximar a solução Φ desconhecida, em cada elemento do domínio, conforme (1).

$$\phi^{e}(x,y) = a^{e} + b^{e}x + c^{e}y. \qquad (1)$$

As constantes *a*,*b* e *c* são valores a serem determinados.

Para cada nó do elemento, da Figura D4, podemos reescrever a equação (1):

$$\begin{aligned} \phi^{e}_{1}(x,y) &= a^{e} + b^{e}x_1 + c^{e}y_1, \\ \phi^{e}_{2}(x,y) &= a^{e} + b^{e}x_2 + c^{e}y_2, \\ \phi^{e}_{3}(x,y) &= a^{e} + b^{e}x_3 + c^{e}y_3. \end{aligned} \qquad (2)$$

Resolvendo o sistema representado na equação (2), reescrevemos a equação (1) em função dos nós:

$$\phi^{e}(x,y) = \sum_{j=1}^{3} N^{e}_{j}(x,y)\,\phi^{e}_{j}, \qquad (3)$$

onde o *N,* na equação (3), representa a função de base e pode ser mostrado que:

$$N^{e}_{j}(x^{e}_{j}, y^{e}_{j}) = \delta_{i,j} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}, \qquad (4)$$

$$N^{e}_{j}(x,y) = \frac{1}{2\Delta^{e}}\left(a^{e}_{j} + b^{e}_{j}x + c^{e}_{j}y\right) \quad j=1,2,3. \qquad (5)$$

Os valores constantes em (5) são:

$$\begin{aligned} a^{e}_{1} &= x^{e}_{2}y^{e}_{3} - y^{e}_{2}x^{e}_{3} & b^{e}_{1} &= y^{e}_{2} - y^{e}_{3} & c^{e}_{1} &= x^{e}_{3} - x^{e}_{2} \\ a^{e}_{2} &= x^{e}_{3}y^{e}_{1} - y^{e}_{3}x^{e}_{1} & b^{e}_{2} &= y^{e}_{3} - y^{e}_{1} & c^{e}_{2} &= x^{e}_{1} - x^{e}_{3} \\ a^{e}_{3} &= x^{e}_{1}y^{e}_{2} - y^{e}_{1}x^{e}_{2} & b^{e}_{3} &= y^{e}_{1} - y^{e}_{2} & c^{e}_{3} &= x^{e}_{2} - x^{e}_{1} \end{aligned} \qquad (6)$$

$$\Delta^{e} = \frac{1}{2}\begin{vmatrix} 1 & x^{e}_{1} & y^{e}_{1} \\ 1 & x^{e}_{2} & y^{e}_{2} \\ 1 & x^{e}_{3} & y^{e}_{3} \end{vmatrix} = \frac{1}{2}\left(b^{e}_{1}c^{e}_{2} - b^{e}_{2}c^{e}_{1}\right) = \text{área do elemento } e. \qquad (7)$$

Na equação (7) temos a área do elemento e na equação (5) esta área entra no denominador; este fato pode representar uma instabilidade no método, quando temos triângulos degenerados, pois neste caso, um dos lados do triângulo seria muito pequeno em relação aos outros dois lados, o que representaria uma área próxima de 0; tal valor poderia ser relativamente menor que o numerador, gerando erros que comprometeriam o método.

Aplicando-se a equação integral a cada elemento e depois com uma condição de "estabilidade" que envolve todo o domínio, temos o sistema de equações lineares a ser resolvido pela LIRIOS. A dimensão do sistema linear obtido é dada pelo número de nós da malha.