Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e Computação
Departamento de Microondas e Ótica

Tese de Mestrado

Ambiente MATLAB – Elementos Finitos para Eletromagnetismo

Eng. Mário Aparecido Corrêa

Orientador: Prof. Dr. Hugo Enrique Hernández Figueroa

Comissão Examinadora:
Hugo Enrique Hernández Figueroa - DMO/FEEC/UNICAMP-Presidente
Maria Aparecida G. Martinez - Universidade Presbiteriana Mackenzie
Philippe R. B. Devloo - FEC/UNICAMP
Rui Fragassi Souza - DMO/FEEC/UNICAMP

Campinas – SP
2001

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

C817a

Corrêa, Mário Aparecido
Ambiente MATLAB – elementos finitos para eletromagnetismo / Mário Aparecido Corrêa. -- Campinas, SP: [s.n.], 2001.

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

1. MATLAB (Programa de computador). 2. Método dos elementos finitos. 3. Eletromagnetismo. 4. Métodos de simulação. I. Hernández Figueroa, Hugo Enrique. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Sumário

Apresenta-se um ambiente de simulação computacional, desenvolvido sobre a plataforma MATLAB, para diversos problemas e efeitos eletromagnéticos. Pela sua versatilidade e abrangência o Método dos Elementos Finitos foi adotado na resolução das equações correspondentes, derivadas das Equações de Maxwell. Aspectos computacionais e vários exemplos de aplicação são discutidos em detalhe.

A técnica de simulação pelo Método de Elementos Finitos vem sendo largamente utilizada em diversos ramos da engenharia e, em especial, como auxiliar no modelamento de dispositivos baseados em ondas eletromagnéticas. Por outro lado, com uma difusão no meio científico e industrial já bastante ampla, o MATLAB tem se mostrado como uma poderosa ferramenta para o desenvolvimento, testes e implementação dos mais variados algoritmos de computação numérica.

Aliar o ambiente do MATLAB às técnicas de programação para o Método de Elementos Finitos torna-se uma tarefa quase que natural se, levarmos em consideração o aspecto matricial do método e, as funcionalidades de manipulação matricial do ambiente. Isto motivou a concepção da *toolbox* mefTool, apresentada neste trabalho em detalhe, a qual tem por filosofia ser um **Ambiente MATLAB - Elementos Finitos para Eletromagnetismo**.

Abstract

A computational simulation environment is presented, which was developed over the MATLAB platform, for a variety of electromagnetic problems and effects. Due to its versatility and wide scope, the Finite Element Method was adopted for the solution of the corresponding equations, which are derived from the Maxwell's Equations. Computational aspects and several illustrative examples are discussed in detail.

The simulation technique for the Finite Element Method has been broadly used in several branches of engineering and, especially, as an effective aid in the modeling of wave electromagnetic devices. On the other hand, with a very wide diffusion in the scientific and industrial market, MATLAB has shown itself as a powerful tool for the development, test and implementation of a large variety of routines for numerical computation.

The merge between MATLAB and the Finite Element programming techniques becomes an almost natural task, if we take into account the characteristics of the method and the MATLAB's functionality to manipulate arrays, in an efficient and effective manner. This motivated the conception of the toolbox presented here, called meftool, which has for philosophy to be a **MATLAB - Finite Element Environment for Electromagnetics**.

Agradecimentos

Agradeço ao Professor Doutor Hugo Enrique Hernández Figueroa pela dedicação, orientação e estímulo para o desenvolvimento deste trabalho.

Agradeço à minha esposa Rosana que muito me apoiou, estimulou e contribuiu para que eu concluísse esta importante etapa. Dedico este trabalho à ela e à minha filha Amanda.

Por fim, agradeço a todos que de um jeito ou de outro me apoiaram.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Um dos pontos fundamentais na formação de um profissional na área de ciências exatas e em especial na engenharia, é sem dúvida a experimentação.

Atualmente existe um justificado interesse por parte da indústria em dispor, sempre que possível, das mais recentes técnicas de simulação computacional, gerando assim um fértil mercado para profissionais mais capacitados em realizá-las.

É neste contexto que apresentamos este trabalho, cujo principal interesse é o de expor o Método de Elementos Finitos como uma das mais utilizadas técnicas computacionais para simulação na atualidade, aliada a uma ferramenta extremamente versátil para o desenvolvimento e testes de aplicativos e, para o aprendizado de técnicas numéricas. Estamos nos referindo ao já muito difundido MATLAB.

Por não ser o escopo deste, não vamos nos deter no formalismo matemático do método pois, existem inúmeros trabalhos muito bem redigidos para esta finalidade. Vamos sim, focarmos nossa atenção em uma *toolbox* chamada `meftool`, desenvolvida para se tornar um ambiente de experimentação e desenvolvimento e cujas funcionalidades estão projetadas para facilitar ao máximo o entendimento do método e do MATLAB. A `meftool` tem ainda por característica ser modular e portável de acordo com as necessidades, podendo ser aplicada tanto em situações de baixa, média ou alta complexidade.

Em paralelo a esse aspecto técnico, existe o aspecto didático, onde sabemos que, discorrer sobre assuntos abstratos como o Eletromagnetismo sem lançar mão de artifícios

lúdicos é uma tarefa árdua para os mestres e para os alunos. Porém, a `meftool` pode dar a esses, uma ferramenta não pedagógia, porém que facilite o aprendizado e estimule a experimentação baseados na simulação.

Para atingirmos estes objetivos, este trabalho foi dividido em quatro grandes blocos a saber:

1. O MATLAB;
2. A `meftool`;
3. Alguns Exemplos;
4. O Método de Elementos Finitos.

Todo o conteúdo teórico dos exemplos deste trabalho é voltado para o estudo de Eletromagnetismo.

Para finalizar, cabe mencionar que o MATLAB é uma poderosa ferramenta para técnicas computacionas, liberando o interessado da árdua tarefa de codificar extensos programas em linguagens como FORTRAN ou C++, permitindo que o mesmo realize testes preliminares de algorítimos mais refinados. No entanto a própria MathWorks desaconselha o uso do MATLAB em seu estado puro pois, quando há necessidade de um grande número de manipulações de dados e sendo o MATLAB um ambiente interpretado, o custo computacional torna-se extremamente alto e os tempos dispendidos para uma simulação podem tornar-se impraticáveis. Como solução para esta questão, o ambiente de desenvolvimento do MATLAB permite a utilização de arquivos previamente compilados, conhecidos como `mex-files`, ou que se exporte os códigos fontes das aplicações para serem compilados em outras linguagens, onde a mais comumente utilizada para este fim é a C/C++.

Capítulo 2

O MATLAB

2.1 Introdução ao MATLAB

O MATLAB é um software interativo cujo elemento de dados básico é uma matriz que não requer dimensionamento, o que permite solucionar muitos problemas computacionais, principalmente os que envolvem formulações matriciais ou vetoriais, como no caso de elementos finitos.

É importante mencionar o fato de que o MATLAB não é uma linguagem de programação no contexto convencional sendo, no entanto, possível utilizá-lo como linguagem interpretada. Escrever funções computacionais é um dos grandes atrativos do MATLAB especialmente por ser relativamente simples, não requer prévio conhecimento de sistemas operacionais, compiladores e demais componentes de uma linguagem tradicional. Porém é bom que se tenha em mente que, o aspecto interativo e interpretado do MATLAB pode reduzir a velocidade de desenvolvimento, mas não necessariamente aumenta a performance computacional.

Por outro lado, o poder dos códigos escritos para MATLAB reside no tamanho e na simplicidade. Por exemplo, uma página de código para MATLAB equivale a várias páginas de código em uma linguagem como FORTRAN. Isto é possível devido ao fato de que para os cálculos numéricos, o MATLAB utiliza uma extensa coleção de sub-rotinas

altamente especializadas e otimizadas, tais como LINPACK e EISPAC, além de rotinas específicas para visualizações gráficas 2D e 3D e capacidade de se produzir animações.

Por fim, com o MATLAB é possível criar interfaces gráficas com o usuário (GUI). Para tanto, o mesmo possui um conjunto básico de funcionalidades para a criação de objetos.

2.2 A Área de Trabalho do MATLAB

Conceitualmente a área de trabalho do MATLAB é uma pilha de memória com camadas definidas para variáveis, comandos digitados, constantes e demais dados lançados na linha de comandos.

Para se obter os nomes das variáveis que estão na área de trabalho, utiliza-se o comando `who`.

Para se obter informações mais detalhadas das variáveis, utiliza-se o comando `whos`. Com este comando, cada variável é listada com sua dimensão, o número de bytes usados e seu tipo. O comando `whos` é especialmente útil quando as variáveis são vetores ou matrizes. Por exemplo:

```
» who
Your variables are:
variável1 variável2 vetor x
» whos
 Name Size Bytes Class
 variável1 2x4 64 double array
 variável2 1x1 8 double array
 vetor 2x4 64 double array
 x 1x1 8 double array
```

Outra função importante na área de trabalho é o `clear`, que tem por objetivo limpar as variáveis da área de trabalho.

```
» clear all % apaga todas as variáveis
» clear variável_1 % apaga somente variável_1
```

```
» clear variável_1;variável_2 % apaga lista de variáveis.
```

Além dessas funções, o item **Show Workspace** do menu **File** cria uma janela chamada de **Workspace Browser**, que contém as mesmas informações fornecidas pelo comando `whos`, porém esta interface permite que se apague ou se atribua novos valores a variáveis selecionadas, semelhante a uma planilha no Microsof Excel, sendo especialmente útil para se editar os elementos de uma matriz.

2.2.1 Variáveis no MATLAB

Assim como em qualquer outra linguagem de programação, o MATLAB tem regras a respeito dos nomes de variáveis, como por exemplo: não é permitido utilizar-se de duas ou mais palavras separadas por espaço para a composição do nome. Mais especificamente, as regras para nomes de variáveis são:

- As variáveis são "case sensitive " (sensíveis a maiúsculas e a minúsculas);
- As variáveis podem conter até 31 caracteres, sendo que tudo o que exceder o 31^o caractere será ignorado;
- Não se deve começar um nome de variável com números e ou caracteres especiais.

Além destas regras, o MATLAB possui diversas variáveis especiais tais como:

- `ans` $\Rightarrow$ Variável padrão para resultados;
- `pi` $\Rightarrow$ O número π;
- `i` ou `j` $\Rightarrow i = j = \sqrt{-1}$;
- `narguin` $\Rightarrow$ Número de argumentos de entrada em uma função;
- `nargout` $\Rightarrow$ Número de argumentos de saída em uma função.

2.2.2 A Linha de Comandos

Apesar de se poder visualizar em uma janela tudo o que foi sendo lançado durante uma seção, somente a última linha do vídeo é que contém o "prompt " para entrada de dados por digitação. Esta linha é também chamada de **linha de comando**.

Atribuir valores a uma variável na linha de comando, é simplesmente digitar a variável seguida do sinal de atribuição (=) e, do valor que se deseja atribuir, como por exemplo $x = 3$. Para se obter o valor atribuído, basta, na linha de comandos, digitar a variável que o MATLAB se encarrega de apresentá-lo como ilustrado abaixo.

```
» x=3 % Atribui-se o valor à variável
x =
 3 % Resposta do MATLAB
» x % Questionando o valor da variável
x =
 3 % Resposta do MATLAB
```

2.2.3 Formatação Numérica

Quando o MATLAB apresenta números, segue diversas regras. Por definição, se o resultado for um número inteiro, o MATLAB o apresentará como inteiro. Da mesma forma, quando o resultado for um número real, o MATLAB irá apresentá-lo com, aproximadamente, quatro dígitos de precisão. Se os dígitos são significativos no resultado e estiverem fora deste limite, o MATLAB irá apresentá-lo com notação científica.

Existem duas maneiras de se alterar estes padrões:

1. No menu **File** pelo ítem **Preferences**, ou;
2. Utilizando-se o comando `format`.

Por exemplo, se pegarmos a fração $\frac{347}{3}$ obtemos:

- `Default` $\Rightarrow$ 115.6667;

- `format short` ⇒ 115.6667;
- `format long` ⇒ 1.15666666667;
- `format short e` ⇒ 1.1567e+002;
- `format long e` ⇒ 1.156666666666667e+002;
- `format hex` ⇒ 405ceaaaaaaaaaab;
- `format bank` ⇒ 115.67;
- `format rat` ⇒ 347/3.

É importante observar que o MATLAB não altera a representação interna do número quando optamos por diferentes formatos de apresentação.

2.3 Os Arquivos M

O equivalente ao código fonte de uma linguagem tradicional, para o MATLAB, é conhecido como arquivo M, onde M é a extensão de um arquivo de texto simples do tipo `arq.m`. Na verdade, um arquivo M pode ser encarado como um script que será interpretado pelo kernel do MATLAB.

Existem dois tipos de arquivos M :

- Arquivos M de comandos e,
- Arquivos M de função.

Dentro do corpo do arquivo, quanto utilizamos no começo da linha o caracter %, temos uma linha de comentário. Se quisermos utilizar as primeiras linhas de um arquivo como um pequeno "help ", as mesmas deverão ser iniciadas com % e dessa forma serão exibidas na janela do ambiente quando, na linha de comandos, digitarmos: `help arq`, onde `arq` é o nome do arquivo M.

2.3.1 Arquivos M de Comandos

O termo "de comandos " sugere a idéia de que o MATLAB simplesmente executa os comandos contidos no arquivo. E é exatamente o que ocorre.

Se por exemplo criarmos um arquivo `exemplo1.m` contendo as seguintes linhas:

```
% Exemplo de Arquivo de Comando

constante1=50;

constante2=90;

variacao=0.01;

res=log2(constante1/constante2)/log2(1+variacao);
```

Quando o MATLAB interpreta o comando `exemplo1.m`, ele dá prioridade aos nomes de variáveis definidas recentemente e das funções internas do programa em relação aos nomes de arquivos M. Sendo assim, se `exemplo1` não for o nome de uma variável em uso ou de um comando interno, o MATLAB abrirá o arquivo `exemplo1.m` e executará os comandos lá contidos, como se tivessem sido inseridos diretamente no *prompt* de comandos.

Como principal característica deste tipo de arquivo M, temos que todas as variáveis do arquivo M tornam-se parte da área de trabalho, ou seja, todo e qualquer valor obtido e ou manipulado por um arquivo M de comandos permanecem na pilha de memória até que sejam intencionalmente eliminados ou enquanto o MATLAB estiver ativo.

Outro detalhe importante é que, nestes arquivos também é possível o uso de controle de fluxo, do tipo IF-ELSE-END, WHILE, SWITCH-CASE ou outra estrutura de programação.

2.3.2 Arquivos M de Função

Um arquivo M de função é semelhante a um arquivo M de comandos, tendo em vista que é um arquivo de texto com extensão **.m**.

A principal diferença entre ambos é que os arquivos M de função se comunicam com o MATLAB apenas por meio de parâmetros de entrada e, por meio de variáveis de saída.

Todas as variáveis definidas internamente pelas funções não aparecem e nem interagem com a área de trabalho do MATLAB. Todos os comandos do MATLAB são arquivos M de funções, compilados no formato `mex-files`, que apresentaremos mais adiante na Seção **2.6**

Se por exemplo, criarmos um arquivo `exemplo2.m` contendo as seguintes linhas:

```
function x=exemplo2(y)
% Exemplo de Arquivo M de Função
a=sin(y);
b=y + 1.056;
x= sqrt(a) - b^2;
```

Esta função recebe a variável `y` como parâmetro e devolve a variável de saída `x`, e, as variáveis `a` e `b` não são transferidas para a área de trabalho.

Os arquivos M de funções devem seguir regras específicas. Além disso, eles possuem várias propriedades importantes como descrito à seguir:

- O nome da função tem de ser idêntico ao nome do arquivo. Por exemplo, a função `exemplo2` é armazenada no arquivo `exemplo2.m`;
- Na primeira vez que o MATLAB executar uma função, ele abre o arquivo M correspondente e compila os comandos, guardando-os na memória de forma a acelerar sua execução;
- As linhas de comentário (iniciadas com %) que antecedem a primeira linha de comando, constituem o texto que é apresentado quando `help nome` for digitado;
- Cada função possui sua própria área de trabalho, sendo que a única ligação entre os arquivos M de função são as variáveis de entrada e de saída;
- Os arquivos M de função podem conter mais de uma função. No `exemplo2` usamos as funções `sin` e `sqrt`;

- Todo arquivo M de função deve começar com o comando `function`.

2.4 Interfaces Gráficas

Quando tivermos em mente a criação de uma aplicação de uso mais regular, ou seja, uma aplicação que exija uma interatividade, podemos lançar mão do uso de interfaces gráficas. Tais interfaces são baseadas em programação orientada por objeto, e por sua vez, torna a objetividade da ação bem mais evidente.

Por definição, uma interface gráfica com o usuário é um ponto de contato ou um método de interação entre uma pessoa (ou usuário) e o computador, ou mais especificamente, o programa de computador.

O objetivo das interfaces gráficas é tornar o uso dos aplicativos o mais amigável possível por intermédio da intuição, ou seja, de forma que o usuário tenha em mente apenas o problema a ser resolvido e não quais as funcionalidades e ou comandos do aplicativo.

Para este fim, uma GUI (*graphical user interface*) é constituída de objetos tais como ícones, menus, botões de comando, áreas de texto entre outros. Geralmente as ações são tomadas por um evento do mouse (como um clique por exemplo).

Toda interface gráfica no MATLAB é construída como um arquivo M de função específico, onde se devem declarar os objetos que a compõem (botões, caixas de texto, etc.), suas propriedades (cor de texto, largura, conteúdo, etc.) e que ações tomar após um evento de mouse.

Para facilitar um pouco o processo de criação de uma interface gráfica, o MATLAB possui uma funcionalidade no menu **File** chamada **GUI Layout Tool**.

Neste ponto, cabe mencionar que o MATLAB não é uma linguagem de programação e tam pouco uma linguagem orientada a objetos ou **OOL**. No entanto com artifícios herdados da linguagem C++ o MATLAB simula um ambiente OOL, porém com poucos e limitados recursos e objetos.

2.4.1 Os Objetos no MATLAB

Os objetos, ou controles, no MATLAB são identificados por um número inteiro chamado **Handel Graphics** (HG). Cada janela aberta no MATLAB recebe um HG pai que por sua vez é associado à um HG filho designado à cada objeto aberto na janela. Com isto, é possível se abrir um número qualquer de janelas, e mesmo assim saber qual objeto em qual janela foi acionado.

Todos os objetos são investidos de propriedades que podem ser manipuladas tanto em tempo de desenvolvimento quanto em tempo de execução. Estas propriedades definem, por exemplo, tamanho, cor de texto, cor de fundo, formato de dados entre outros.

A mais importante propriedade de um objeto de ação é a **CallBack**. Esta tem por objetivo passar parâmetros a serem executados a uma função chamada **eval**. Se por exemplo quiséssemos acionar o mouse sobre um botão **Sair**, sua propriedade `CallBack` seria:

»`CallBack [end]`

Os objetos disponíveis na versão 5.3 do MATLAB são:

- Menu;
- Popup Menus;
- Push Buttons;
- Radio Buttons;
- Check Boxes;
- Static Text Boxes;
- Editable Text Boxes;
- Sliders;
- Frames.

Com este conjunto de objetos é possível interfaciar todas as funç ões do MATLAB. No entanto existem características que poderiam ser classificadas de desvantagens técnicas, tais como:

- Não há possibilidade de se fazer um re-escalonamento ou re-posicionamento dos objetos quando se faz uma ampliação da janela em que os mesmos estão designados;
- Não há possibilidade de se fazer um tratamento de erros, sendo que freqüentemente o algorítimo sob controle da GUI é interrompido e os *status* de variáveis se perdem;
- Por ser a GUI uma janela gráfica originalmente destinada como área de traçados de dados, a mesma pode acabar recebendo em sua área de objetos, também chamada de **axes**, o resultado de uma saída de cálculo, se a mesma estiver em primeiro plano, o que geralmente acontece.

2.5 Organização das Funções do MATLAB

O MATLAB em sua versão básica apresenta vinte categorias principais de funções. Algumas das funções do MATLAB são incorporadas ao próprio interpretador, enquanto outras encontram-se sob a forma de arquivos M. As funções em arquivos M, assim como os arquivos M contendo texto de ajuda para funções incorporadas, estão organizadas em vinte diretórios contidos sob a árvore `toolbox/MATLAB`, cada um deles contendo os arquivos associados a uma dada categoria. O comando `help` do MATLAB apresenta uma tabela on-line dessas categorias principais, como mostrado a seguir.

Categorias	Descrição
color	Funções de controle de cor e modelamento de iluminação
datafun	Funções de análise de dados e transformata de Fourier
demos	Demonstrações e exemplos
elfun	Funções de matemática elementar
elmat	Matrizes elementares e manipulação de matrizes
funfun	Funções de função - métodos numéricos não lineares
general	Comandos de aplicação geral
graphics	Funções gráficas de aplicações gerais
iofun	Funções de baixo nível de entrada e saída de arquivos
lang	Estruturas e depuração de linguagem
matfun	Funções matriciais - álgebra linear numérica
ops	Operadores e caracteres especiais
graph2d	Gráficos bidimensionais
graph3d	Gráficos tridimensionais
polyfun	Funções polinomiais e de interpolação
sparfun	Funções de matrizes esparsas
specfun	Funções matemáticas especiais
specmat	Matrizes especiais
sounds	Funções de processamento de sons
strfun	Funções de cadeias de caracteres

2.5.1 As *toolboxes*

Quando criamos um conjunto de arquivos M com uma finalidade bem específica, com características próprias e que interagem entre sí, estamos criando uma *toolbox*. Na

versão completa do MATLAB 5.3, existem distribuídas algumas *toolboxes*, onde as mais conhecidas são a `simulink` e `pdetool`. A primeira foi especialmente desenvolvida para trabalhos em automação e controle e a segunda para a solução de problemas de valor de contorno.

Em nosso caso, elaboramos uma *toolbox* batizada de `meftool` na qual entraremos em detalhes mais adiante.

Uma *toolbox*, em geral, faz uso de características não disponíveis no modo interativo, como por exemplo o uso de objetos para a criação de interfaces com o usuário.

De um modo geral, as *toolboxes* são desenvolvidas por terceiros e são distribuídas, em geral, com licenças comerciais de uso.

2.5.2 Regras Gerais para Uso de Funções e Comandos

Quando estamos no modo interativo, ou quando recorremos ao uso de arquivos M, temos que levar em consideração uma sintaxe apropriada para a entrada de comandos ou para a entrada de dados. Descreveremos a seguir algumas regras básicas, especialmente para uso de funções de manipulação de matrizes.

- O MATLAB faz distinção entre caracteres maiúsculos e minúsculos. Logo a variável `A` é diferente da variável `a`;
- Um ponto e vírgula (`;`) no final de um comando, "desliga " a exibição de resultados;
- Todas as regras matemáticas, incluindo hierarquias, devem ser respeitadas;
- Quando não for especificada uma variável de saída, o MATLAB automaticamente especifica a variável `ans` como por exemplo:

```
>> c = 2 + 2 % Especificando a variável c

>> c = 4

>> 2 + 2 % Sem especificar variáveis

>> ans = 4
```

- O comando `whos` mostra todas as variáveis declaradas e em uso. Se após a digitação anterior, executássemos `whos`, obteríamos:

```
>> c ans
```

- O comando `dir` mostra todo o conteúdo do diretório corrente. É possível o uso de comandos do **OS**, tal como `cd, md` entre outros;

- Com o uso dos comandos `load` e `save`, é possível ler e armazenar arquivos de dados. A principal característica destes comandos é que o nome do arquivo tem que ser o mesmo da variável que vai receber os dados;

- A entrada de dados em uma matriz pode ser feita de duas maneiras:

  - Todos os elementos separados por vírgula (,) e as linhas em seqüências separadas por ponto e vírgula (;) tudo entre colchetes [].

```
>> A = [1,2,3;4,5,6;7,8,9]
        1  2  3
>> A =  4  5  6
        7  8  9
```

  - Linha a linha com os elementos separados por espaços tudo entre colchetes []

```
>> A = [1 2 3
>> 4 5 6
>> 7 8 9]
        1  2  3
>> A =  4  5  6
        7  8  9
```

- As operações com matrizes seguem as regras da álgebra linear;

- O comando `size` traz a dimensão da matriz:

```
>> size(A)
```

```
>> ans = 3 3
```

- O uso de dois pontos (:), pode especificar uma linha ou uma coluna dentro de uma matriz:

```
>> A( :, 3)
          3
>> ans =  6
          9
>> A ( 1, :  )
>> ans = 1 2 3
```

- Em uma expressão do tipo $Ax = b$ devemos utilizar o comando \

```
>> x = A \ b
```

- Uma chamada à uma função do tipo :

  `[K1,b1] = mefdchlt1d(K,b,p,nn,p2)`, obtemos como saída os valores `K1` e `b1` através dos parâmetros `K, b, p, nn` e `p2`.

Listamos aqui algumas características em conjunto com alguns comandos com o objetivo de apresentar a filosofia de trabalho do MATLAB. Sugerimos para se obter maiores detalhes o uso dos comandos `help` (para versão texto) ou `helpdesk` (para versão HTML), em conjunto com as Referências **[3]** e **[4]**.

2.6 Integração com Outras Linguagens

Uma importante característica do MATLAB é a capacidade de integração com outras linguagens de programação, tais como C/C++ ou FORTRAN.

Esta integração se dá de duas formas distintas:

1. Indiretamente ⇒ Através do comando `run` ou `!`;

2. Diretamente ⇒ Através de um compilador C/C++ (`mcc`), que basicamente converte arquivos M de comando, ou função, em fontes C/C++ ou importando-se os fontes escritos nesta linguagem para um arquivo M binário (extensão `mex`).

2.6.1 Método `run` ou !

Na verdade não é precisamente correto chamar estes comandos de método, pois, os mesmos tem por objetivo permitir a execução de funções do sistema operacional no qual o MATLAB está instalado. Porém, podemos utilizá-los como artifício para a execução de outros aplicativos já compilados. A integração neste caso pode se dar por meio de arquivos texto gerados pelo MATLAB que sirvam de "input " para o aplicativo, ou arquivos texto gerados como saída dos mesmos.

2.6.2 Método do `mex-files` e Compilador C/C++

O compilador integrado do MATLAB (mcc) pode traduzir arquivos M em fontes C/C++. Os fontes C resultantes podem ser utilizados para gerar executáveis, bibliotecas ou `mex-files` que são arquivos compilados para uso no MATLAB.

Existem três razões básicas para se utilizar este método:

1. Um grande incremento na velocidade de processamento;
2. Esconder algorítimos proprietários, pois no caso do `mex-files`, o arquivo é binário;
3. Criar aplicativos "standalone " ou ainda bibliotecas compartilhadas em C (DLLs por exemplo) ou bibliotecas estáticas para C++ .

Segundo a documentação que acompanha o produto, o incremento na velocidade de processamento se dá especialmente em algorítimos envolvendo muitas iterações. Porém as rotinas gráficas não sofrem nenhum tipo de melhoria.

Não utilizamos este método, principalmente porque estamos avaliando o MATLAB como ambiente e, teríamos nossos códigos convertidos em binário dificultando sua utilização em situações que não estejam contempladas neste trabalho.

No entanto uma vantagem é muito clara: Com este método, pode-se aproveitar as funções matemáticas do MATLAB e criar aplicativos "standalone " (aplicativos em modo executável que não dependa do seu ambiente de desenvolvimento), que possam ser distribuídos ou integrados a outros aplicativos comerciais ou não.

Outro ponto muito forte é que, com este método, podemos facilmente converter códigos já testados e estabilizados escritos em outras linguagens, especialmente FORTRAN e C/C++, em `mex-files`. Podemos, desta forma, aproveitarmos todos os esforços empregados na criação de algorítimos específicos. Estas funcionalidades fazem do MATLAB uma poderosa ferramenta de desenvolvimento, pois integra suas funções com as geradas pelo usuário, empacotando tudo depois em bibliotecas especializadas, utilizáveis em outras aplicações.

2.7 Considerações Finais

Como vimos até o momento, o MATLAB é uma poderosa ferramenta no que diz respeito a manipulação de dados e no tocante a execução de cálculos numéricos. Quanto a programação, o mesmo possui alguns recursos que podemos combinar, e com isto criarmos uma vasta gama de ferramentas para as mais diversas aplicações. No entanto, como pudemos observar, pelas características de ambiente interpretado do MATLAB, o mesmo torna-se especialmente lento quando tentamos manipular sistemas matriciais muito volumosos ou quando há um grande número de iterações. Como solução, a MathWorks aconselha a utilização do compilador `mcc`, que elimina a dependência com o ambiente interpretado, gerando através de uma conversão dos arquivos M, em `mex-files` binários que podem ser utilizados diretamente para o desenvolvimento de outros aplicativos ou como bilbiotecas compartilhadas. Os mex-files podem também ser gerados a partir de códigos escritos para FORTRAN ou C/C++ para serem executados no ambiente MATLAB.

Capítulo 3

A meftool

3.1 Introdução à `meftool`

Como havíamos citado anteriormente, criar códigos para o MATLAB é uma das mais marcantes características deste ambiente e, dada sua grande facilidade de manipulação matricial e comandos para álgebra linear, desenvolver um conjunto de funções específicas para elementos finitos é quase que uma evolução natural. Existem diversas publicações voltadas para esta finalidade, tais como as Referências **[13]** e **[14]**.

Porém estes trabalhos, em sua grande maioria, são voltados para a área de mecânica e, os códigos neles apresentados são parametrizados e caracterizados também para este fim. Logo, a criação de uma *toolbox* voltada para o estudo de eletromagnetismo vem preencher uma importante lacuna no uso deste ambiente. É com esta intenção que apresentamos a `meftool` (método de elementos finitos - *toolbox*) desenvolvida de modo a se integrar às potencialidades do MATLAB, ao estudo de eletromagnetismo, baseado na análise por elementos finitos. Embora todo o raciocínio empregado na construção desta *toolbox* estar voltado para o eletromagnetismo, a mesma tem por filosofia a multidisciplinariedade. Ou seja, todas as funções podem ser utilizadas para simulações em mecânica, termodinâmica ou em aplicações que exijam soluções aproximadas para problemas de valores de contorno.

Esta característica ficará mais evidente a medida em que apresentarmos sua estrutura.

3.2 A Estrutura da `meftool`

Por convenção, aplicativos desenvolvidos para o MATLAB recebem a designação de *toolbox*, e estas se encontram em uma árvore de diretórios sob o diretório onde o ambiente foi instalado, também com o nome de *toolbox*. Todos os arquivos que atualmente compõem a `meftool` estão dispostos nesta estrutura, acondicionados em um diretório com o nome de `meftool`, como ilustrado abaixo.

c:\matlabr11\toolbox\meftool

Os arquivos nesta estrutura tem as seguinte características:

- Todos os nomes de arquivos começam com o prefixo `mef`;
- Estão classificados de acordo com o domínio do problema;
- O arquivo `mefsolve.m` é comum a ambos os modos (1D e 2D);
- O arquivo `meftop.m` é um arquivo M de comandos, que tem por objetivo desmembrar um arquivo texto contendo a topologia de uma malha arbitrária bi-dimensional. Para gerar esta malha, utilizamos um aplicativo chamado GenMesh de uso público, disponibilizado pela Universidade Federal da Coreia;
- Os arquivos iniciados com bmp são os componentes do *Beam Propagation Method;*
- Os arquivos que contém o sufixo "gui " são as interfaces gráficas com os usuários;
- O sub-diretório malhas abriga os arquivos com as topologias das malhas utilizadas nos exemplos do Capítulo **4.**

Todos os arquivos M de função que compõem a `meftool` podem ser evocados:

- a partir da linha de comandos;
- a partir de um arquivo M de comandos;
- a partir de uma interface gráfica.

Pois, como vimos anteriormente, todo arquivo M de função passa a ser integrado ao conjunto de funções "*standard* " do MATLAB.

Apresentaremos em seguida um resumo das funções criadas para a `meftool`.

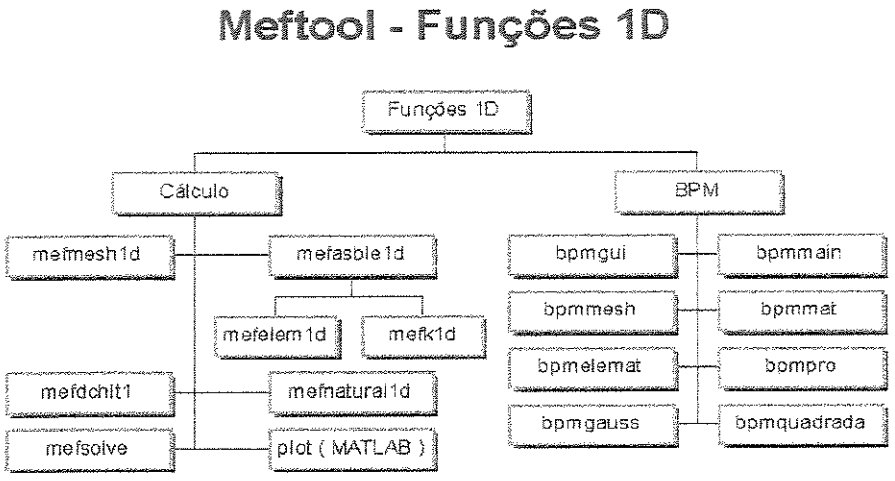

Fig. 3-1: Organização 1D

3.2.1 As Funções para 1D

Desenvolvidas para a solução de problemas do tipo:

$$-\frac{d}{dx}\left(\alpha\frac{d\phi}{dx}\right)+\beta\phi = f \qquad (3.1)$$
$$x \in (0, L)$$

Para tanto, a primeira função a ser considerada é a `mefmesh1D.m` que foi desenvolvida a partir do programa MESH.for do Prof. Dr. Hugo H. E. Figueroa.

Esta função tem como objetivo gerar a malha 1D, utiliza-se como parâmetros de entrada três arranjos contendo respectivamente:

- Coordenadas das interfaces em xb;

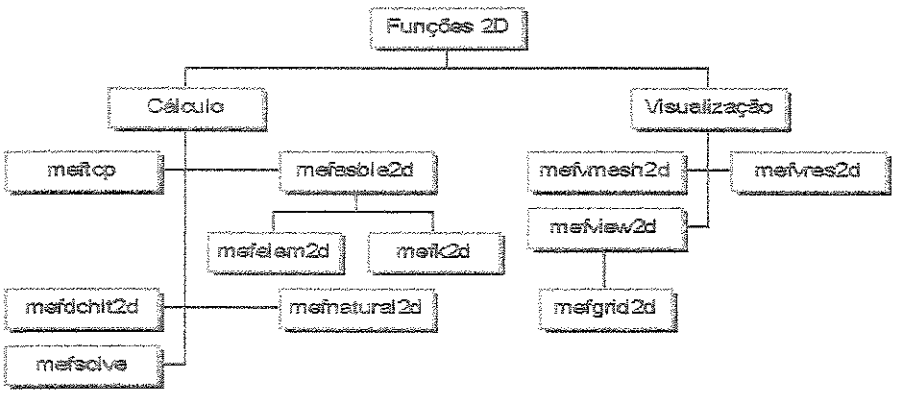

Fig. 3-2: Oeganização 2D

- Características físicas do problema (parâmetros α, β, f) em bx;
- Densidade da Malha por região em xd.

A chamada para esta função tem a seguinte sintaxe:

`[alpha,beta,f,coord,nel]=mefmesh1D(xb,bx,xd);`

A função `mefasble1d.m` monta a matriz global $[K]$ e o vetor global $\{b\}$ para o sistema:

$$[K]\{\phi\} = \{b\} \tag{3.2}$$

tendo como parâmetros de entrada, as saídas do `mefmesh1d.m`.

A sintaxe é `[K,b]=mefasble1d(coord,alpha,beta,f,nel)` onde **coord** é um arranjo contendo as coordenadas da malha. As variáveis **alpha**, **beta** e **f** são os parâmetros físicos do problema e, **nel** o número de elementos.

As funções `mefdchlt1d.m` e `mefnatural1d.m` servem para aplicarmos as condições de contorno. Tipicamente as condições de contorno são do tipo:

- **Dirichlet/Dirichlet**;

- **Neumman/Neumman;**
- **Dirichlet/Neumman;**
- **Neumman/Dirichlet.**

Em ambas funções temos o parâmetro **nn** que, quando for passado como sendo $nn = 1$, estaremos indicando, para qua sejam aplicadas as condições de contorno iguais em ambos os extremos do domínio, ou seja:

- **Dirichlet/Dirichlet** ou;
- **Neumman/Neumman.**

Por exemplo, se fizermos uma chamada do tipo :

`[K,b] = mefdchlt1d(K,b,p,nn,p2);`

com `nn` = 1, então estaremos aplicando no sistema `[K,b]` a condição de contorno do tipo **Dirichlet/Dirichlet**, e os parâmetros `p` e `p2`, são os valores que ϕ assume nos extremos.

Ou ainda se, `[K,b]= mefnatural1d(K,b,gamma,q,nel,nn,gamma2,q2);`

sendo $nn = 1$ temos a condição **Neumman/Neumman**. Em ambos os casos os parâmetros após **nn** serão ignorados se $nn \neq 1$.

O próximo passo será fazermos a chamada `sol=mefsolve(K,b);` que nos devolve o vetor solução $\{sol\}$ que pode ser plotado por meio do comando `plot(sol);`

Resumindo, a sequência para a solução de problemas envolvendo uma dimensão é:

1. `mefmesh1d;`
2. `mefasble1d;`
3. `mefdcht1d` ou `mefnatural1d;`
4. `mefsolve;`
5. `plot.`

3.2.2 *Beam Propagation Method* (BPM)

Um caso particular de aplicação em domínio uni-dimensional é o BPM, que está apresentado em detalhes nas Referências **[10]** e **[12]**. Trata-se de um método de simulação para a propagação de um feixe ao longo de um guia de ondas ou no espaço livre. Em sua essência é composto pelas mesmas funções acima citadas, mas que por se tratar de um método envolvendo iterações e animações gráficas, pequenas alterações tiveram que ser realizadas, gerando um segundo conjunto de funções para 1D.

As funções para o BPM são:

- `bpmgui` ⇒ Interface gráfica para o caso de guias com perfil composto de 3 ou 5 regiões;
- `bpmmain` ⇒ Função principal;
- `bpmmesh` ⇒ Gera a malha;
- `bpmelemat` ⇒ Monta as matrizes elementares;
- `bpmmat` ⇒ Monta a matriz Global;
- `bpmpro` ⇒ Cálculo da propagação.;
- `bpmgauss` ⇒ Sinal de entrada "gaussiano ";
- `bpmquadrada` ⇒ Sinal de entrada quadrático.

Poderemos ver mais detalhes de seu funcionamento no Exemplo **4.4** do Capítulo **4**.

3.2.3 As Funções para 2D

Desenvolvidas para a solução de problemas do tipo:

$$-\frac{\partial}{\partial x}\left(\alpha_x \frac{\partial \phi}{\partial x}\right) - \frac{\partial}{\partial y}\left(\alpha_y \frac{\partial \phi}{\partial y}\right) + \beta\phi = f \tag{3.3}$$

As funções para 2D são por natureza mais complexas, porém procurou-se manter a filosofia básica de desenvolvimento adotada para as funções 1D.

A única função ainda não desenvolvida, especialmente devido a sua complexidade é a de um "malhador ". No entanto, podemos recorrer a um malhador externo, como por exemplo o já citado GenMesh ou então, nos casos mais simples, podemos entrar com a malha de forma manual.

A estrutura básica para a entrada de dados está dividida em três matrizes definidas como:

- `elem(N,4)` sendo que `N=número de elementos` e deve ser preenchida da seguinte forma `(elemento nó1 nó2 nó3)`
- `nodes(M,3)` onde `M=número de nós` e será preenchida como `(nó x y)`
- `bound(n,f)` onde `n` é o nó na fronteira ou região `f` . A dimensão de bound é dada pela maior coluna, e se $(n, f) = 0$, simplesmente indica que nesta fronteira ou região, o número de nós é menor do que em outras colunas, como no exemplo abaixo:

$$bound = \begin{matrix} 1 & 3 & 6 & 8 \\ 2 & 4 & 7 & 9 \\ 3 & 5 & 8 & 10 \\ 0 & 6 & 0 & 1 \end{matrix}$$

  - Fronteira 1 = `(1 2) e (2 3)`
  - Fronteira 2 = `(3 4), (4 5) e (5 6)`
  - Fronteira 3 = `(6 7) e (7 8)`
  - Fronteira 4 = `(8 9), (9 10) e (10 1)`

Na verdade, a matriz bound pode ser utilizada para armazenar, simultaneamente, os nós que estão em uma ou mais fronteiras e os nós que estão em uma ou mais regiões dis-

tintas, como será visto no Capítulo **4,** onde na Seção **4.5.4** esta matriz foi utilizada desta forma, e nos demais Exemplos servia apenas para armazenar os nós de fronteiras. Cabe aqui mencionar que o conceito de região determina, por exemplo, quais as características físicas do meio desta região ou, onde aplicar uma fonte de excitação.

De posse das entradas especificadas acima, podemos fazer chamada à função mefasble2d.m com a seguinte sintaxe:

`[K,b]=mefasble2d(elem,nodes,alphax,alphay,beta,f).`

Esta função gera a matriz global $[K]$ e o vetor global $\{b\}$ para o sistema:

$$[K]\{\phi\} = \{b\} \tag{3.4}$$

Onde $\alpha_x, \alpha_y, \beta$ e f são os parâmetros físicos do problema.

Esta função tem encadeada uma chamada à função `mefelem2d.m` e `mefk2d.m` onde `mefelem2d.m` monta a matriz elementar e o vetor elementar para o elemento da vez no cálculo, enquanto `mefk2d.m` efetua a adição da matriz elementar à matriz global $[K]$ e o vetor elementar ao vetor global $\{b\}$.

As chamadas:

- `[K,b]=mefdchlt2d(K,b,bound,nodes,essencial,p)`
- `[K,b]=mefnatural2d(K,b,bound,nodes,natural,gamma,q)`

Aplicam as condições de contorno nas fronteiras especificadas por `bound`.

Finalmente fazemos a chamada `sol=mefsolve(K,b);` que nos devolve o vetor solução $\{sol\}$ que pode ser visualizado através de `mefgrid2d.m` em conjunto com o arquivo M de comandos `mefview2d.m` onde `mefgrid2d.m` tem por objetivo mapear e interpolar o vetor $\{sol\}$, em função de uma malha com densidade ajustável. Veremos maiores detalhes na Seção seguinte.

3.2.4 Visualização Gráfica na `meftool`

O MATLAB tem um conjunto de funções gráficas de alta qualidade, proporcionando ao usuário uma grande variação de aplicabilidade tais como tratamento de imagens, cartografia, animações, entre outras.

Na grande maioria dos casos de resultados de simulações unidimensionais, a função `plot` acaba sendo suficiente. No entanto, muitas vezes se faz necessário a combinação de um grupo de funções para se obter resultados mais específicos. No BPM por exemplo, temos como resultado da simulação um feixe propagado sob a forma de um vetor. Nas funções gráficas do MATLAB não há como se visualizar este vetor adequadamente. Aí entra a combinação citada para se traçar a cada passo um vetor com o campo calculado, combinando-os depois em um gráfico de superfície. É exatamente isto que o seguinte trecho de programa do botão 'Surf View ' da interface gráfica do BPM faz:

```
'Callback',[...
'figure(2),'...
'campo0=zeros(size(campom));,'...
'[l,n]=size(campom);,'...
' for a= 1:l,'...
'campo0(a,:)=campom(a,:);,'...
'surf(xline,propaga,campo0),'...
'shading interp,'...
'view(-9,60),'...
'axis([xa1 xb1 ya yb za zb]);,'...
'axis off,'...
'drawnow,'...
'M(a)=getframe;,'...
'end,'...
'for a=1:2,'...
'movie(M);,'...
```

```
'end'], ...
```

Neste caso, `campom` é um vetor e `campo0` é uma matriz em cujas colunas está o campo calculado após cada passo.

Uma outra característica do BPM é que, como se trada de uma simulação de propagação de um feixe de ondas planas, utilizou-se um recurso do MATLAB destinado à animação gráfica. Este recurso é evocado da seguinte forma:

```
for a= 1:steps
 [Zi,x1,y1]=mefgrid2d(nodes,z(a),dxy)
 mefview2d(x1,y1,Zi,5,''BPM - Animação'')
 axis off
 drawnow
 M(a)=getframe
end.
movie(M);
```

Onde a dupla de comandos `getframe` e `movie` foram desenvolvidas especificamente para a animação dentro do MATLAB.

Para os demais propósitos da `meftool`, foram criadas as seguintes rotinas gráficas, todas baseadas nas funções básicas do MATLAB:

- `mefvmesh2d` ⇒ Permite visualizar a malha gerada no GenMesh
- `mefvres2d` ⇒ Traça o resultado, tendo a malha como parâmetro.
- `mefview2d` ⇒ Permite visualizar os resultados de cinco formas diferentes.
- `mefgrid2d` ⇒ Produz uma grade retangular para os gráficos de superfície utilizados na `mefview2d`

A função `mefvmesh2d` recebe como parâmetros as matrizes `elem` e `nodes`. Nela cada elemento é mapeado e traçado através da função básica do MATLAB chamada `patch`, cuja sintaxe é: `patch(xx,yy,'w')`, onde xx e yy são as coordenadas dos vértices de cada

elemento (nó) e w corresponde à cor de preenchimento do elemento e, neste caso é branco (white). A forma de se evocar a `mefvmesh2d` é `mefvmesh2d(elem,nodes)`.

Já a função `mefvres2d` foi desenvolvida com o objetivo de apresentar o resultado da simulação, levando-se em consideração a aparência da malha. O único inconveniente é que, quando se tem uma malha muito densa, o resultado visual tende a ficar muito escuro.

A chamada à esta função é feita na forma `mefvres2d(elem,nodes,sol)` onde a principal diferença com relação a `mefmesh2d` é o parâmetro *sol*, que é o vetor contendo a solução. Esta também faz uso da função básica `patch`, porém desta vez com uma segunda opção de sintaxe para visualização em 3D: `patch(xx,yy,zz,abs(zz))` onde xx, yy são como anteriormente, zz é um vetor contendo os resultados e, $abs(zz)$ servirá de parâmetro para a coloração do elemento, dado aos resultados obtidos em correspondência aos seus vértices (nós).

A função `mefview2d` é na verdade um grupamento de funções de visualização integrados, que podem ser selecionados de acordo com as necessidades do usuário, bastando para isso informar, via parâmetro, qual o tipo de gráfico se deseja visualizar e, qual a densidade da grade de visualização desejada.

A chamada a esta função se dá da seguinte forma: `mefview2d(x1,y1,Zi,tipo,título)`, onde o parâmetro `tipo` indica ao visualizador qual gráfico apresentar, de acordo com:

Opção	**Tipo de Gráfico**
1	Malha 3D
2	Pseudo cores com curvas de nível
3	Curvas de nível e diagrama de vetores (indicam a direção do gradiente)
4	Gráfico de superfície
5	Gráfico de pseudo cores

O parâmetro `título` é uma string que será apresentada na janela de visualização do gráfico.

Já os parâmetros $x1$, $y1$ e zi são provenientes da função `mefgrid2d` que tem, como objetivo, gerar uma malha retangular de densidade ajustável para a utilização nos gráficos.

A `mefgrid2d` é necessária porque, no MATLAB, para se produzir gráficos 3D ou de contorno baseados em um vetor, se faz necessário o uso de uma grade regular e retangular. As coordenadas desta grade no plano x,y devem estar contidas no domínio do problema a ser analisado, e, o vetor solução será utilizado para interpolação de uma matriz quadrada a ser traçada, com base nas coordenadas da malha original.

A chamada a `mefgrid2d` é na forma `[Zi,x1,y1]=mefgrid2d(nodes,sol,dxy)`, onde o vetor $[Zi, x1, y1]$ é o resultado obtido com a função `mefgrid2d`, sendo Zi uma matriz com os dados interpolados em relação a grade e, $x1$ e $y1$ as cordenadas da grade. Os parâmetros *nodes*, *sol* e *dxy* são respectivamente o vetor contendo as coordenadas dos nós, o vetor resultado e um vetor contendo a densidade da grade, do tipo $[dx, dy]$.

O processo de geração da grade é o seguinte:

1. Encontrar os extremos no eixo x ⇒ minX=min(x);maxX=max(x);
2. Encontrar os extremos no eixo y ⇒ minY=min(y);maxY=max(y);
3. Montar o vetor coordenada x, com densidade dx ⇒ x1=linspace(minX,maxX,dxy(1));
4. Montar o vetor coordenada y, com densidade dy ⇒ y1=linspace(minY,maxY,dxy(2));
5. Gerar as coordenadas da grade regular com base na malha original⇒ [Xi,Yi]=meshgrid(x1,y1)
6. Interpolar o vetor resultado com base na grade gerada ⇒ Zi=griddata(x,y,sol,Xi,Yi);

De posse do vetor $[Zi, x1, y1]$ podemos então chamar a `mefview2d`. Por exemplo, se optarmos pelo tipo de gráfico número 4, a seguinte sequência de comandos seria acionada:

1. `surf(x1,y1,Zi);` ⇒ Cria o gráfico de superfície
2. `shading interp;` ⇒ Gera o padrão de Pseudo cores
3. `colormap(hsv);` ⇒ Seleciona um mapa de cores para o shading interp
4. `title(titulo);` ⇒ Imprime o título

3.3 Interfaces Gráficas com o Usuário (GUI)

Para completar o conjunto de funções desenvolvidas para a `meftool`, temos as **interfaces gráficas com o usuário**.

Na verdade, estas interfaces foram elaboradas com a finalidade de apresentar as técnicas de programação orientada a objeto dentro dos recursos disponíveis no MATLAB, bem como proporcionar aos exemplos uma maior interoperabilidade.

Cada uma das interfaces foram montadas com o auxílio da ferramenta **Gui Layout**.

Procuramos utilizar todos os objetos à disposição, a saber:

- `Combo List;`
- `Text Box;`
- `Check Box;`
- `Option Botton;`
- `Painel;`
- `Frame.`

Os arquivos M de função destinados às intefaces gráficas tem uma pequena diferença em sua estrutura, se comparados aos demais tipos de arquivos M.

Esta diferença se dá porque nestes arquivos não há uma sequência de comandos, mas sim a definição dos objetos e suas propriedades. As ações são todas passadas à linha de comando, como parâmetros, por intermédio do único evento manipulável chamado `Callback`. Único manipulável porque, diferentemente das demais linguagens de programação orientada a objetos, no MATLAB não se tem controle sobre eventos tais como clique de mouse ou posicionamento de cursor. Um exemplo de como é feita a definição de um objeto é ilustrado abaixo:

```
nome_lógico_do_objeto = uicontrol('Parent',h0, ...
 'Units','points', ...
```

```
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'HorizontalAlignment','left', ...
'ListboxTop',0, ...
'Position',[3.75 129 37.5 12.75], ...
'Callback ',[ 'Lista de comandos '], ...
'String','Texto exibido no objeto', ...
'Style','botton', ...
'Tag','Nome_lógico_do_Objeto ', ...
'TooltipString','Texto exibido quando cursor está sobre o objeto');
```

Este é o código para a criação de um botão, cuja ação após o evento de clique do mouse é a **'lista de comandos '** apontados em `Callback`.

3.4 Considerações Finais

Consideramos a `meftool` como a mais relevante contribuição deste trabalho. Com ela é tanto possível aprender os conceitos básicos do método de elementos finitos, quanto realizar complexas análises e simulações. Dentro da `meftool`, o ponto mais importante é o visualizador gráfico contido nas funções `mefgrid2d` e `mefview2d` pois com elas podemos visualizar não só as saídas da `meftool` como qualquer outro resultado obtido com qualquer outra função ou aplicativo.

Estando a `meftool` inserida no ambiente do MATLAB, é perfeitamente possível interagir com outras funcionalidades do mesmo, tornando-a bastante poderosa e abrangente. No entanto, sendo totalmente dependente do ambiente interpretado do MATLAB, a `meftool` sofre por consequência uma redução em sua performance sendo que uma evolução natural será a compilação em `mex-files` ou transformá-la em uma biblioteca para C/C++ ou FORTRAN extrapolando em muito suas atuais limitações. Outras características importantes são a modularidade e a portabilidade, pois é muito simples agregar novas funções

e, sendo interpretada dentro do ambiente MATLAB, a mesma é independente do sistema operacional podendo ser executada sem nenhuma alteração em todas as plataformas que tenham o MATLAB instalado.

Como pontos fortes da `meftool`, podemos citar:

1. Extrema facilidade no manuseio;
2. Disponibilidade como biblioteca para outras aplicações em elementos finitos;
3. Possibilidade de interação via linha de comando ou interfaces gráficas;
4. Grande poder de manipulação dos resultados (análise gráfica e numérica);
5. Interação com outras funcionalidades do ambiente MATLAB;
6. Modularidade;
7. Portabilidade.

Capítulo 4

Exemplos

4.1 Introdução aos Exemplos

O objetivo deste capítulo é apresentarmos a `meftool` de forma prática e, desta forma, podermos avaliar seu funcionamento, potencial e características.

Na montagem dos exemplos, considerou-se tanto a possibilidade do uso da linha de comando, quanto o uso de interfaces gráficas. Isto nos permitirá demonstrar a versatilidade tanto do ambiente MATLAB quanto da `meftool`.

O Exemplo1 **4.2** não possue interface gráfica. Ele serve para ilustrar o uso de arquivos M de comando. Todos os demais estão disponíveis em interfaces gráficas padronizadas, podendo também ser inicializados a partir da linha de comandos. Uma exceção fica para o BPM - Exemplo **4.4**, devido ao fato deste se utilizar de animação.

Para acessar os exemplos via interface gráfica, basta digitar na linha de comandos `exemplos`, que surgirá a barra de botões flutuante ilustrada abaixo:

4.2 Linha de Transmissão com Perdas - 1D

Este exemplo foi tirado da Referência **[2]**. Trata-se de uma linha de transmissão de fios paralelos e com perdas, como ilustrado na figura 4-2:

O problema pode ser analisado por parâmetros distribuidos, da seguinte forma:

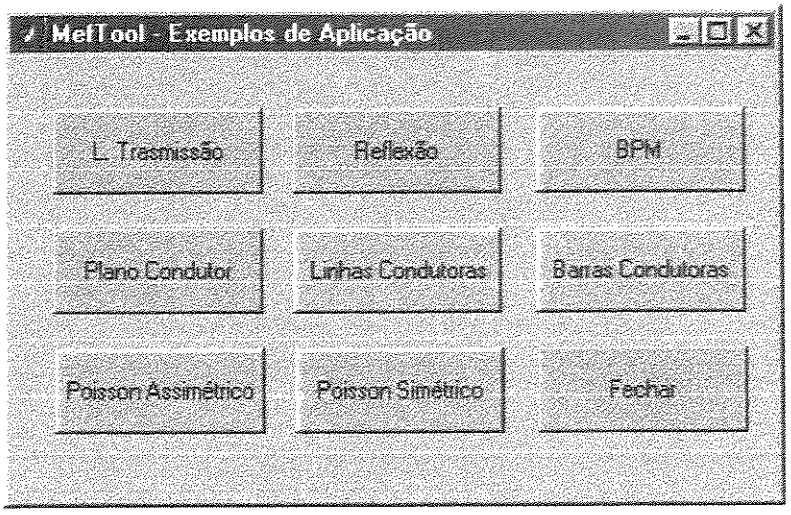

Fig. 4-1: Tela para seleção de exemplos

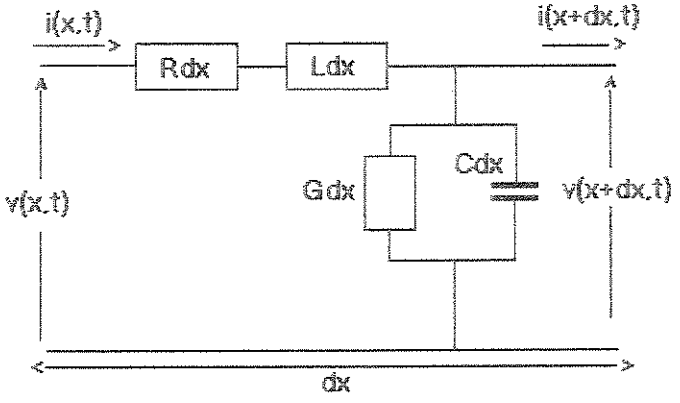

Fig. 4-2: Linha de Transmissão com Perdas

$$-\frac{dv(x)}{dx} = (R + j\omega L)\, i(x) \tag{4.1}$$

$$-\frac{di(x)}{dx} = (G + j\omega C)\, v(x) \tag{4.2}$$

Multiplicando-se e, considerando-se $r = R + j\varpi L$ e $g = G + j\varpi C$ (ou seja, valores complexos) como impedância característica, obtemos:

$$\frac{d^2v(x)}{dx^2} - rgv(x) \tag{4.3}$$

Sendo que R, L, G, C são respectivamente a resistência, indutância, condutância e capacitância da linha por unidade de comprimento e dx é o elemento diferencial do comprimento do fio.

Para este tipo de linha de transmissão podemos partir da seguinte relação para a definição dos parâmetros:

$$\frac{R}{L} = \frac{G}{C} \tag{4.4}$$

a) Constante de propagação:

$$\begin{aligned} \gamma &= \alpha + j\beta = \sqrt{(R + j\omega L)\left(\frac{RC}{L} + j\omega C\right)} \\ &= \sqrt{\frac{C}{L}}\,(R + j\omega L)\,; \end{aligned} \tag{4.5}$$

$$\alpha = R\sqrt{\frac{C}{L}} \tag{4.6}$$

$$\beta = \omega\sqrt{LC} \tag{4.7}$$

b) Velocidade de fase:

$$u_p = \frac{\omega}{\beta} = \frac{1}{\sqrt{LC}} \tag{4.8}$$

c) Impedância característica:

$$Z_0 = R_0 + jX_0 = \sqrt{\frac{R + j\omega L}{\left(\frac{RC}{L}\right) + j\omega C}} = \sqrt{\frac{L}{C}}; \tag{4.9}$$

$$R_0 = \sqrt{\frac{L}{C}} \tag{4.10}$$

$$X_0 = 0 \tag{4.11}$$

Por fim temos que a uma dada istância x, o percentual de atenuação pode ser expresso por: $(e^{-x\alpha})\,\%$

As condições de contorno para esse exemplo são dadas por:

$$v_{x=0} = V_0 \tag{4.12}$$

$$\frac{dv}{dx}_{x=D} = ri_{x=D} = 0 \tag{4.13}$$

Partindo-se da equação (**4.3**) podemos encontrar o funcional que descreve esta situação, de acordo com a equação (**4.17**).

$$F(v) = \frac{1}{2}\langle \pounds v, v\rangle - \frac{1}{2}\langle \pounds v, u\rangle + \frac{1}{2}\langle v, \pounds u\rangle - \frac{1}{2}\langle v, f\rangle + \frac{1}{2}\langle \pounds u, v\rangle \tag{4.14}$$

onde

$$\begin{aligned} v &= v' + u \\ \pounds v &= f = 0 \\ \pounds &= \frac{d^2}{dx^2} - rg \\ \langle v, u\rangle &= \int uv \end{aligned} \tag{4.15}$$

$$
\begin{aligned}
\langle \pounds v, v\rangle &= -\int_0^D \left(\frac{dv(x)}{dx}\right)^2 dx + \left[\left(\frac{dv(x)}{dx}\right) v(x)\right]_0^D \qquad (4.16)\\
-\langle \pounds v, u\rangle &= +\int_0^D \left(\frac{dv(x)}{dx}\frac{du}{dx}\right) dx - \left[\left(\frac{dv(x)}{dx}\right) u\right]_0^D\\
\langle \pounds u, v\rangle &= -\int_0^D \left(\frac{du}{dx}\right) v(x)dx + \left[\left(\frac{du}{dx}\right) v(x)\right]_0^D\\
\left[\left(\frac{dv(x)}{dx}\right) v(x)\right]_0^D &= v'(D)v(D) - v'(0)v(0) = 0 \text{ sendo } v(D_{\to\infty}) = 0 \text{ com}\\
-\left[\left(\frac{dv(x)}{dx}\right) u\right]_0^D &= -v'(D)u(D) + v'(0)u(0) = 0 \text{ mesma condição anterior}\\
\left[\left(\frac{du}{dx}\right) v(x)\right]_0^D &= u'(D)v(D) - u'(0)v(0) = 0 \text{ tomamos } u'(0) = 0
\end{aligned}
$$

Logo,

$$
F(v) = \frac{1}{2}\int_0^D \left[\left(\frac{dv(x)}{dx}\right)^2 + rg\left(v(x)\right)^2\right] dx \qquad (4.17)
$$

Após a execução do arquivo M de comandos `linha.m` (somente digitando-se `linha` na linha de comandos), obtivemos o seguinte resultado ilustrado abaixo, considerando-se $R = 50(\Omega), C = 0.1(nF), \alpha = 0.01(dB/m)$ e $V_0 = 10(v)$, comprimento da linha $D = 1000m$ e 3000 elementos obtemos:

$$
R_0 = \sqrt{\frac{L}{C}} = 50\,(\Omega)
$$

$$
\begin{aligned}
\alpha &= R\sqrt{\frac{C}{L}} = 0.01(dB/m) =\\
&= \frac{0.01}{8.69}(Np/m) = 1.15 \times 10^{-3}(Np/m)
\end{aligned}
$$

$$R = \alpha R_0 = \left(1.15 \times 10^{-3}\right) \times 50 = 0.057(\Omega/m)$$

$$L = CR_0^2 = 10^{-10} \times 50^2 = 0.25(\mu H/m)$$

$$G = \frac{RC}{L} = \frac{R}{R_0^2} = \frac{0.057}{50^2} = 22.8(\mu S/m)$$

Podemos então deduzir que após 1Km a perda será de:

$$\frac{V_1}{V_0} = e^{-1000\alpha} = e^{-1.15} = 0.317 \text{ ou } 31.7\%$$

A figura 4-3 apresenta o resultado desta simulação.

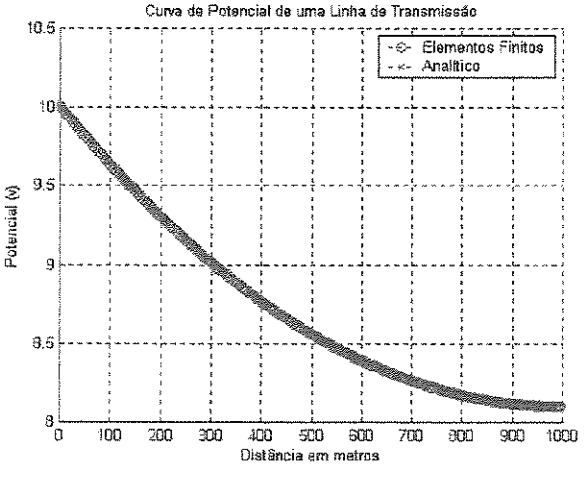

Fig. 4-3: Perda de potência x distância

Conteúdo do arquivo M de comando linha.m

```
%
% xb (vetor que contém as coordenadas das interfaces - [x0,x1,...,xn])
% xd (vetor que contém as divisões entre as interfaces)
% bx (arranjo que contém as características dos materiais - [x0,xn,r,g])
clear all
R=50;
G= 22.8E-6 ;
D=1000;
V0=10;
xb=[0 D];
bx=[0 D R G 0];
xd=3000;
% Início da meftool
[alpha,beta,f,coord,nel]=mefmesh1d(xb,bx,xd,0);
[K,b]=mefasble1d(coord,alpha,beta,f,nel);
[K,b]=mefdchlt1d(K,b,V0,nel,0,0);
zz=mefsolve(K,b);
%Fim da meftool
% Para o Cálculo da Solução Analítica
alpha=-1.15E-3;
for i=1:length(coord)-1
 % Montagem do Eixo x
 x(i)=coord(i);
 % Cálculo da Solução Analítica
 exact(i)= V0*exp(alpha*x(i));
end
plot(x,zz,'r:o',x,exact)
axis([0 10 -2 10]);
```

```
xlabel('Distância em metros')
ylabel('Potencial (v)')
title('Curva de Potencial de uma Linha de Transmissão')
legend('Elementos Finitos', 'Analítico')
grid on; box off
```

4.3 Problema de Espalhamento - 1D

Este problema, obtido da Referência **[1]**, considera uma onda plana incidindo em uma superfície com um ângulo θ pré-determinado.

A espessura do dielétrico é L, a permissividade relativa é ε_r e a permeabilidade relativa é μ_r.

Para a polarização E_z a onda incidente pode ser expressa por;

$$E_z^{inc}(x,y) = E_0 e^{jk_0 x\cos(\theta) - jk_0 y\sin(\theta)} \tag{4.18}$$

Onde E_0 é a grandeza que denota a magnitude do campo incidente e, θ é o âmgulo de incidência. Para satisfazer a condição de continuidade do campo na interface perpendicular ao eixo x, o campo total deve ter um fator comum dado por $e^{-jk_0 y\sin(\theta)}$. Dada esta observação, a equação escalar de Helmholtz que governa o campo elétrico E_z, pode ser reduzida, para o modo TM a:

$$\frac{d}{dx}\left(\frac{1}{\mu_r}\frac{dE_z}{dx}\right) + k_0^2\left(\varepsilon_r - \frac{1}{\mu_r}\sin^2\theta\right)E_z = 0 \tag{4.19}$$

A condição de contorno a ser aplicada para E_z é a condição de contorno de Dirichlet:

$$E_z|_{x=0} = 0 \tag{4.20}$$

Similarmente, para a polarização H_z a onda incidente pode ser expressa por:

$$H_z^{inc}(x,y) = H_0 e^{jk_0 x\cos(\theta) - jk_0 y\sin(\theta)} \tag{4.21}$$

A equação escalar de Helmholtz que governa o campo magnético H_z, pode ser reduzida, para o modo TE a:

$$\frac{d}{dx}\left(\frac{1}{\varepsilon_r}\frac{dH_z}{dx}\right) + k_0^2\left(\mu_r - \frac{1}{\varepsilon_r}\sin^2\theta\right)H_z = 0 \tag{4.22}$$

A condição de contorno a ser aplicada para H_z é a condição de contorno de Neumann:

$$\frac{dH_z}{dx}\,|_{x=0} = 0 \tag{4.23}$$

Queremos então obter o coeficiente de reflexão desta superfície.

Para a solução deste problema, desenvolvemos a interface gráfica ilustrada na figura 4-4, onde os parâmetros físicos são manipulados livremente, podendo dessa forma o usuário simular uma grande variedade de situações.

Pode o usuário optar pela polarização (E_z ou H_z), escolher o número de elementos (número de divisões do domínio), o valor de E_0 ou de H_0 (Campo incidente), variar o ângulo de incidência θ e ainda manipular μ_r ou ε_r podendo-se escolher entre valores constantes ou variáveis, de forma gaussiana ou senoidal.

Utilizando-se como parâmetros $L = 1$, Número de Elementos $= 50$, $\theta = 45^0$, $\mu_r = \varepsilon_r = 1$ para o modo E_z, a magnitude do campo incidente (neste caso E_0) como sendo

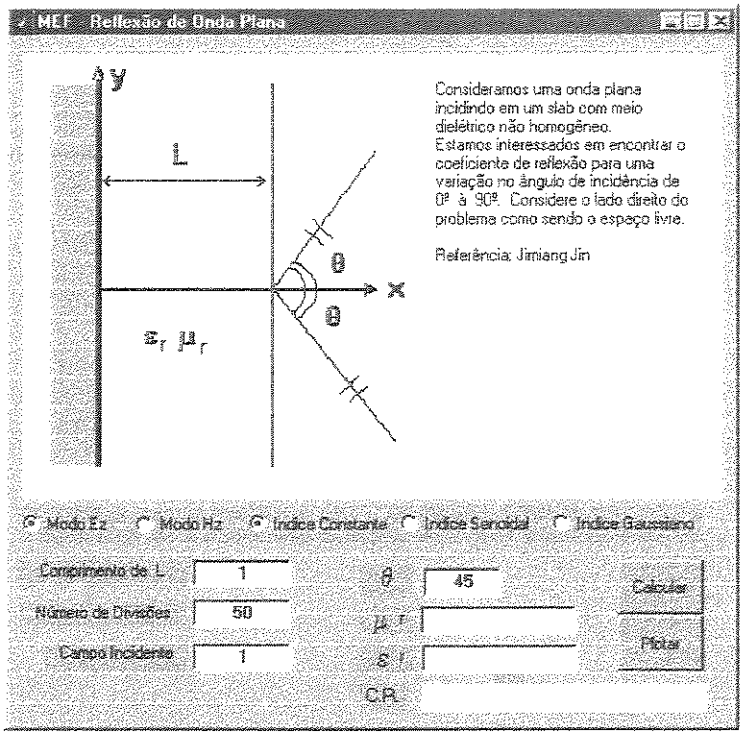

Fig. 4-4: Interface gráfica para o problema de espalhamento

igual à 1, obtivemos como resultado $C.R. = 0.769359$ que é o coeficiente de reflexão para este θ:

Se na linha de comando fizermos a chamada:

refl=reflec($L, \mu_r, \varepsilon_r, E_0$ ou H_0,Nº Elem,1 para E_z ou 0 para $H_z, \theta_{máximo}$,indice);

onde:

- $índice = 0 \Rightarrow \mu_r, \varepsilon_r$ = Constante;
- $índice = 1 \Rightarrow \mu_r, \varepsilon_r$ Variam de forma senoidal;
- $índice = 2 \Rightarrow \mu_r, \varepsilon_r$ Variam de forma gaussiana.

poderemos montar uma curva do coeficiente de reflexão em função do ângulo de incidência, cujo intervalo é $[0, \theta_{máximo}]$. Podemos ver na figura 4-5 a curva obtida para $L = 1$, $\mu_r = \varepsilon_r = 1$, Número de Elementos = 50, $E_0 = 1$, 1 para E_z e $\theta_{máximo} = 45^0$. ou seja:

`>>reflexao( 1,1,1,1,50,1,45,0)`

4.4 Beam Propagation Method - 1D

A pesquisa em circuitos de ótica integrada e dispositivos ópticos planares ficaram mais ativas nos últimos anos. Nesta área, um importante problema teórico é calcular como uma onda de luz é propagada em um circuito óptico que tenha índice de refração arbitrário.

Muitos métodos tem sido propostos com esta finalidade por diversos pesquisadores. Um destes métodos é o "Beam Propagation Method (BPM) ", proposto pela primeira vez em 1978 por J. A. Fleck Jr. e por Feit. M. D. [10]

Existem um grande número de versões do BPM que empregam diferentes tipos de aproximações numéricas. Aqui, apresentamos a aproximação para BPM por elementos finitos (FE-BPM)

Para elucidar a versatilidade da `meftool`, estaremos apresentando uma interface gráfica (figura 4-6), para análise da propagação de um feixe de luz ao longo do eixo z.

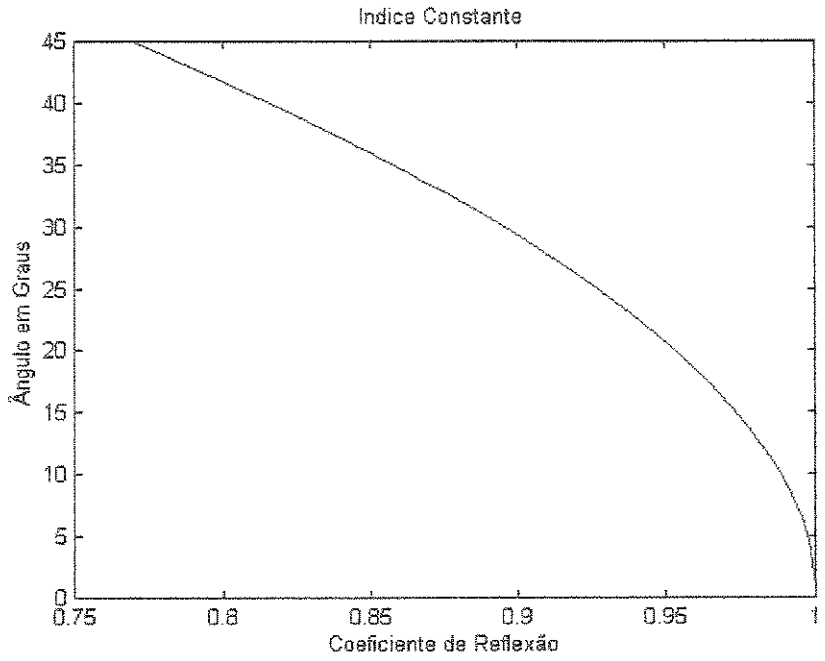

Fig. 4-5: Curva de coeficiente de reflexão (C.R.) x ângulo de incidência

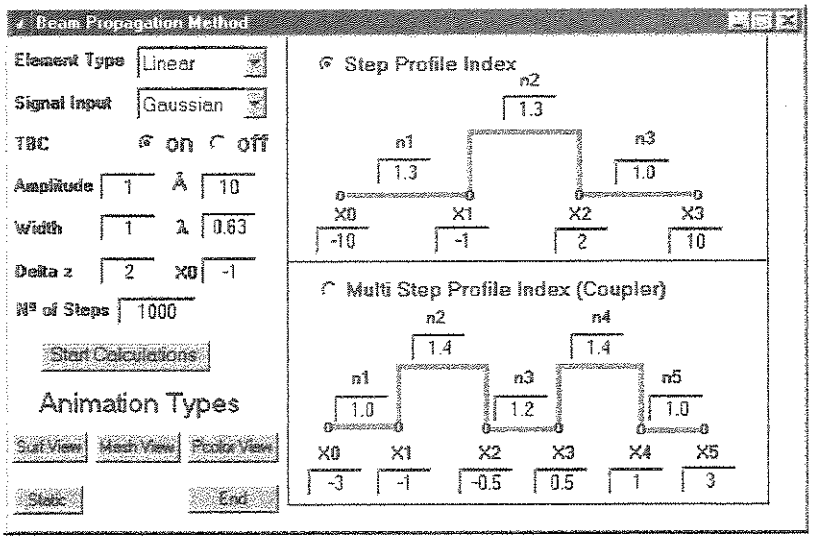

Fig. 4-6: Interface gráfica para o BPM

Como pode-se observar, é uma interface simples que permite a simulação da propagação de um feixe em um guia de ondas, sendo que o perfil do mesmo é determinado pelos padrões de índice degrau, onde, se for selecionado "Step Profile Index " teremos um perfil com três regiões e, se a seleção for "Multi Step Profile Index " teremos cinco regiões. Com esta interface, é possível o uso de elementos lineares ou quadráticos e, gerarmos um pulso gaussiano ou degrau como entrada. A opção TBC (transparent boundary conditions) é uma condição de contorno adequada e muito utilizada para o BPM, sendo recentemente substituída pela PML (Perfectly Matched Layer Boundary Conditions), apresentada em **[15]** . Uma boa referência para o BMP com TBC é dada em **[12]** e **[16].**

Nesta interface é considerado um guia de onda planar, onde y e z representam a direção transversal e de propagação respectivamente e, não ocorre variação na direção x, ou seja:

$$\frac{\partial}{\partial x} = 0 \tag{4.24}$$

O feixe ψ pode ser descrito como:

$$\psi(y,z) = \phi(y,z)e^{-jk_0 n_0 z} \tag{4.25}$$

Com o uso da aproximação de Fresnel, $\phi(y,z)$ pode ser descrita pela seguinte equação:

$$-j2k_0 n_0 \frac{\partial \phi}{\partial z} + \frac{\partial^2 \phi}{\partial y^2} + k_0^2 \left[n^2(y,z) - n_0^2\right] \phi = 0 \tag{4.26}$$

Nesta equação, k_0 é o número de onda no vácuo, $n(y,z)$ é a distribuição do índice de refração e, n_0 é uma constante cujo valor está perto de $n(y,z)$.

Nos extremos da janela de computational, podemos assumir a seguinte equação:

$$\left(\frac{\partial \phi}{\partial y} + \alpha\phi\right)_{y_{extremos}} = 0 \tag{4.27}$$

aqui $\alpha(z)$ é um parâmetro que é renovado pelo cálculo sucessivo de propagação de feixe.

A equação **(4.27)** representa a condição de contorno transparente (TBC) nos limites virtuais.

Após aplicarmos o método variacional apresentado no Apêndice e, aplicando-se o algorítimo de Crank-Nicholson para a propagação na direção z, chegamos a seguinte equação:

$$[A]\{\phi\}_{i+1} = [B]_i\{\phi\}_i \tag{4.28}$$

No MATLAB, podemos resolver este sistema, fazendo-se:

$$b = [B]_i\{\phi\}_i \tag{4.29}$$

$$[A]\{\phi\}_{i+1} = b \tag{4.30}$$

Um ponto importante a ser citado é que pode-se acompanhar a evolução dos cálculos através de uma animação.

Como resultado de uma simulação para o perfil de índice degrau com $n_1 = 1.1$ e $n_2 = n_3 = 1.0$, e com um ângulo de incidência de 75^o, que é maior que o ângulo crítico, podendo ser obtido da seguinte forma:

$$\theta_c = \sin^{-1}\left(\frac{n_2}{n1}\right) = \sin^{-1}\left(\frac{1}{1.1}\right) = 65.3800^o \tag{4.31}$$

Uma observação importante é que na janela de interface com o usuário, o ângulo $\hat{A}$ é o complemento do ângulo de incidência, que é medido a partir da normal à interface.

Nesta situação há reflexão quase total do feixe, como pode-se observar nas figuras 4-7 e 4-8.

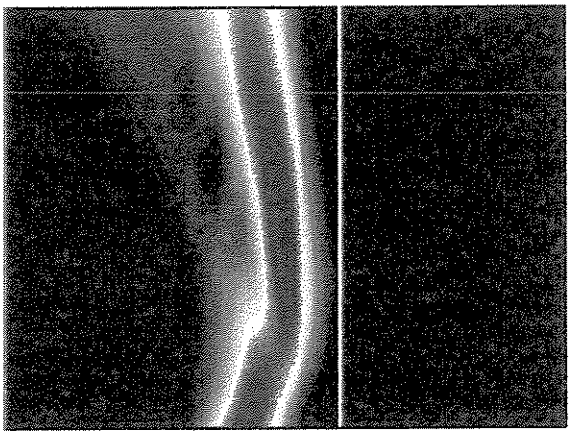

Fig. 4-7: Incidência do feixe acima do ângulo crítico - Pseudo Cores

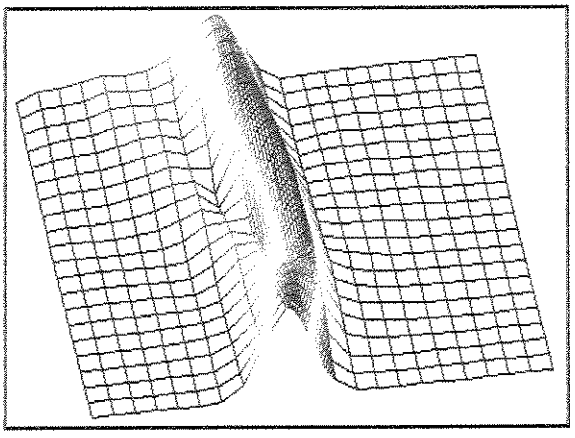

Fig. 4-8: Incidência do feixe acima do ângulo crítico - Gráfico 3D

.

.

Na próxima simulação, fizemos com que o ângulo de incidência fosse 20^o que é menor do que o ângulo crítico, obtendo-se as figuras 4-9 e 4-10.

.

.

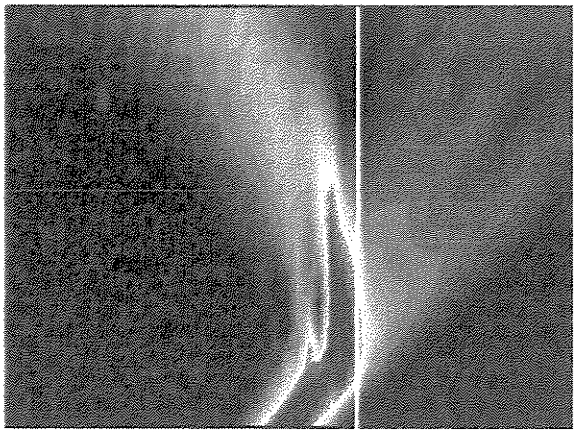

Fig. 4-9: Incidência do feixe abaixo do ângulo crítico - Pseudo Cores

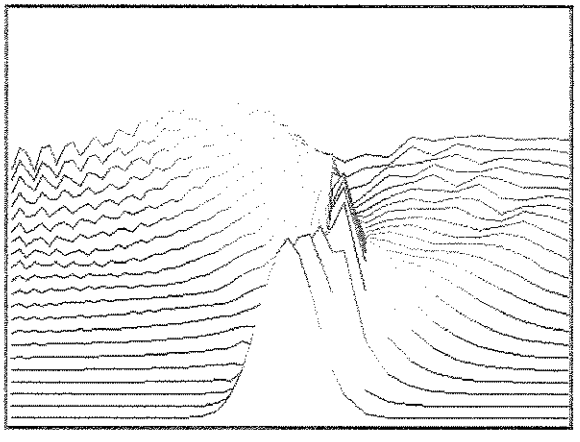

Fig. 4-10: Incidência do feixe abaixo do ângulo crítico - Gráfico 3D

Apresentamos a seguir uma simulação utilizando-se para a TBC a PML e, um guia

em "S " com as características apresentadas na figura 4-11.

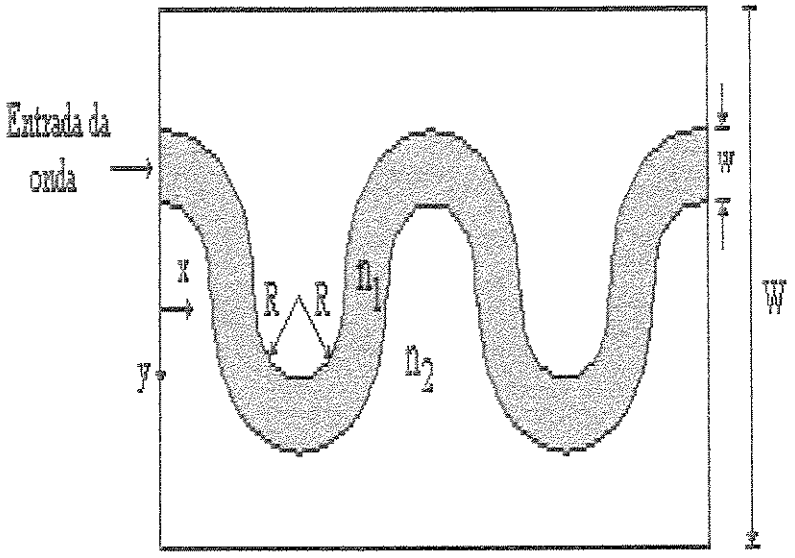

Fig. 4-11: Características de um guia em "S "

A evolução computacional do campo foi obtida com base na referência [17], onde o índice de refração n1=1.45 e a diferença entre os índices de refração dada por

$$\Delta = \left(\frac{n_1 - n_2}{n_1}\right) = 2.5\% \tag{4.32}$$

com raio de curvatura de 267,9 mm e distância de propagação de 2200 mm (figura 4-12) e 1000 mm (figura 4-13) respectivamente.

4.5 Equação de Poisson - 2D

Um problema clássico, em eletrostática, é o da distribuição de potencial. Este estudo apresenta vários subsídios importantes para a compreensão de fenômenos eletromagnéticos.

Se partirmos da fórma diferencial da lei de Gauss:

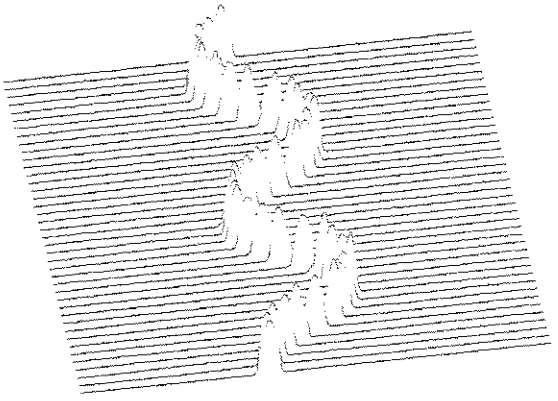

Fig. 4-12: Propagação em 2200 mm

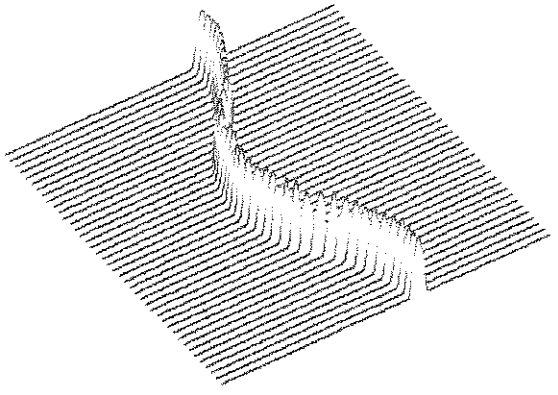

Fig. 4-13: Propagação em 1000 mm

$$\nabla \cdot \vec{D} = \rho \tag{4.33}$$

$$\vec{D} = \varepsilon \vec{E} \tag{4.34}$$

onde ρ é a densidade volumétrica de cargas.

E, considerando-se que em um campo puramente eletrostático, $\vec{E}$ pode ser expresso como o negativo do gradiente do potêncial ϕ, ou seja:

$$\vec{E} = -\nabla\phi \tag{4.35}$$

Se combinarmos as equações **(4.33)**, **(4.34)**, **(4.35)**, obtemos

$$\nabla \cdot (\varepsilon_r \nabla\phi) = -\frac{\rho}{\varepsilon_0} \tag{4.36}$$

ou,

$$-\frac{\partial}{\partial x}\left(\varepsilon_{xx}\frac{\partial\phi}{\partial x}\right) - \frac{\partial}{\partial y}\left(\varepsilon_{yy}\frac{\partial\phi}{\partial y}\right) = \frac{\rho}{\varepsilon_0} \tag{4.37}$$

Conhecida como equação de Poisson, onde ϕ é o potencial no domínio, $(\varepsilon_{xx}, \varepsilon_{yy})$ representam a constante dielétrica do material na direção x e y, respectivamente.

Se $\rho = 0$, a equação de Poisson se reduz na equação de Laplace,

$$-\frac{\partial}{\partial x}\left(\varepsilon_{xx}\frac{\partial\phi}{\partial x}\right) - \frac{\partial}{\partial y}\left(\varepsilon_{yy}\frac{\partial\phi}{\partial y}\right) = 0 \tag{4.38}$$

4.5.1 Plano Condutor

Vamos considerar uma região retangular condutora, como ilustrado na figura 4-14.

Se considerarmos uma tensão em cada lateral do retângulo, teremos um problema onde $\rho = 0$, e portanto, regido pela equação de Laplace 4.38.

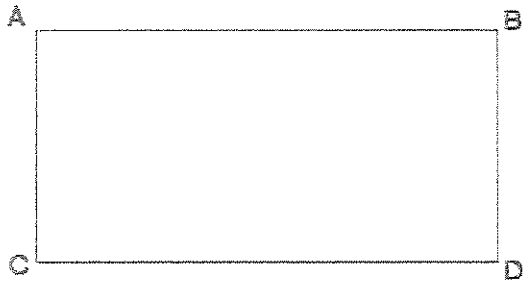

Fig. 4-14: Plano Metálico

A interface gráfica para este problema específico é apresentada na figura 4-15.

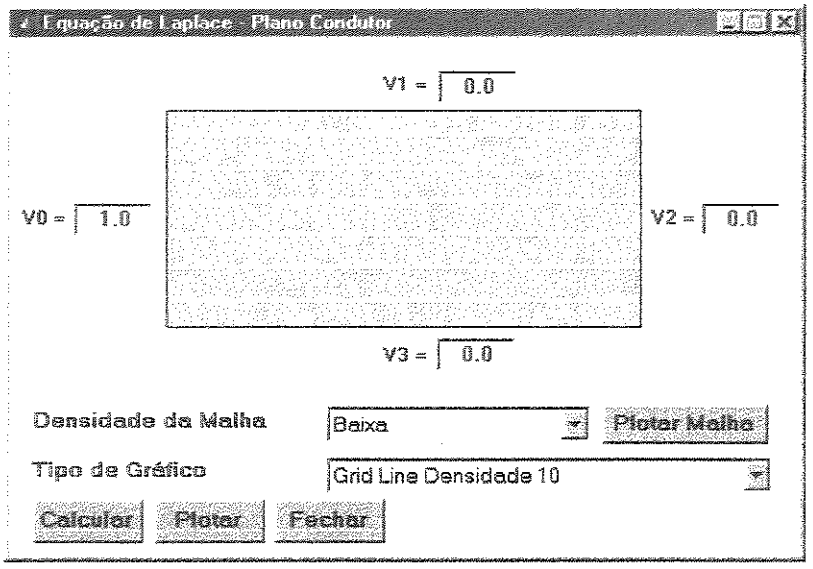

Fig. 4-15: Interface Gráfica - Plano Metálico

Observamos que é possível ao usuário alterar livremente os valores das tensões a serem

aplicadas em cada lado da região, podendo dessa forma obter uma grande variedade de simulações. Para esta interface estão definidas três densidades de malha representadas nas figuras 4-17, 4-16 e 4-18.

Baixa	⇒	17 nós
Média	⇒	89 nós
Alta	⇒	789 nós

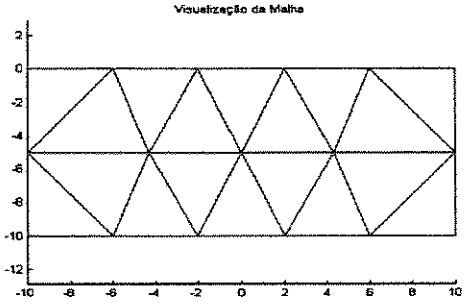

Fig. 4-16: Discretização Baixa = 17 nós

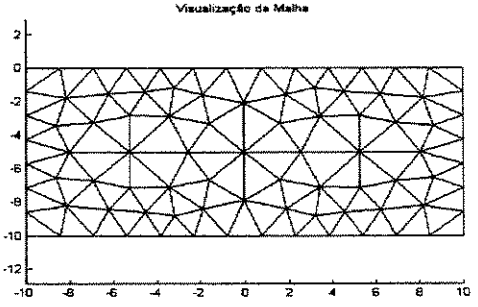

Fig. 4-17: Discretização Média = 89 nós

Uma outra opção para o usuário é a escolha do tipo de gráfico.

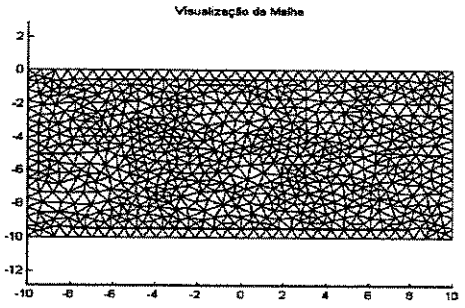

Fig. 4-18: Discretização Alta = 789 nós

Tipo de Gráfico
Grid Line Densidade 10
Grid Line Densidade 50
Gridline Densidade 100
Linhas de Campo Densidade 10
Linhas de Campo Densidade 50
Linhas de Campo Densidade 100
Vetores Densidade 10
Vetores Densidade 50
Vetores Densidade 100
Mesh Densidade 10
Mesh Densidade 50
Mesh Densidade 100
Pseudo Cores Shading Interp

Se considerarmos que $V_0 = 1V, V_1 = V_2 = V_3 = 0$, e com a malha de densidade média, obtemos os resultados apresentados na figura 4-19.

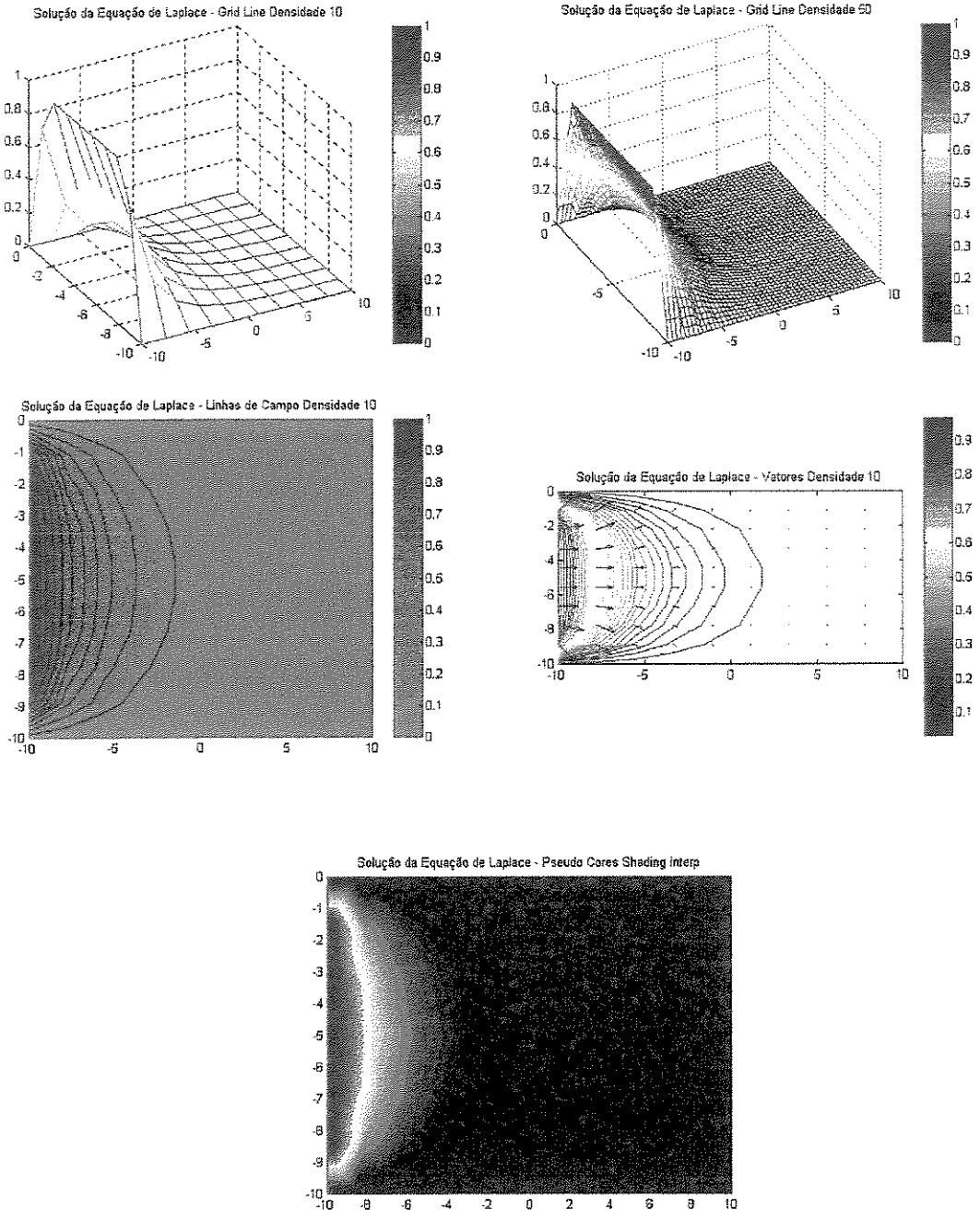

Fig. 4-19: Saídas da simulação para o Plano Condutor

4.5.2 Duas Linhas Condutoras

Usando a equação de Laplace, podemos também resolver o problema da distribuição do potencial entre duas linhas condutoras. Com o auxílio da interface mostrada na figura 4-20, podemos variar as tensões das cargas bem como seus sinais. Podemos também atribuir um potencial às fronteiras, de modo a aumentar a complexidade do problema. Os mesmos recursos utilizados no exemplo anterior, estão disponíveis nesta interface. Disponibilizamos para este exemplo, duas densidades de malha representadas nas figuras 4-21 e 4-22.

Média	⇒	669 nós
Alta	⇒	1008 nós

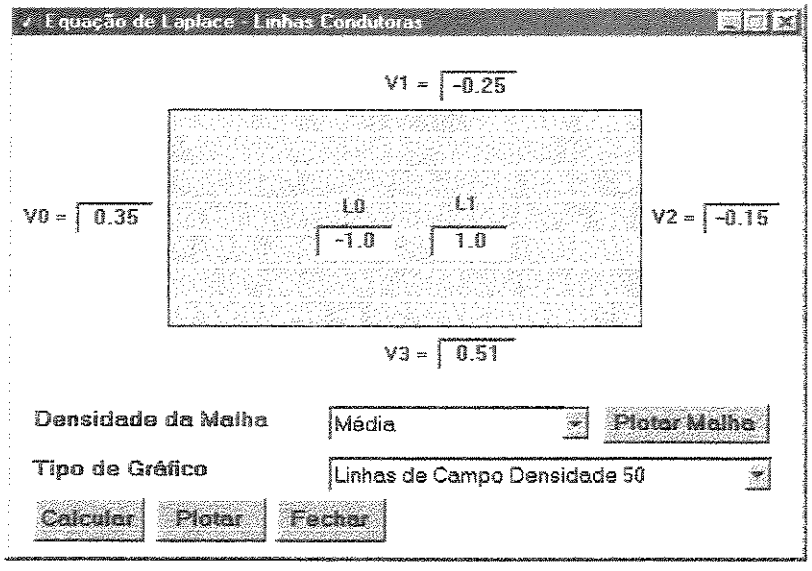

Fig. 4-20: Interface Gráfica - Cargas no Espaço

Para ilustrar, considerando-se os valores ilustrados na interface (figura 4-20) , obtivemos os resultados representados nas figuras 4-23, 4-24, 4-25 e 4-26.

A seguir apresentamos o trecho do programa que executa os cálculos. Podemos sub-

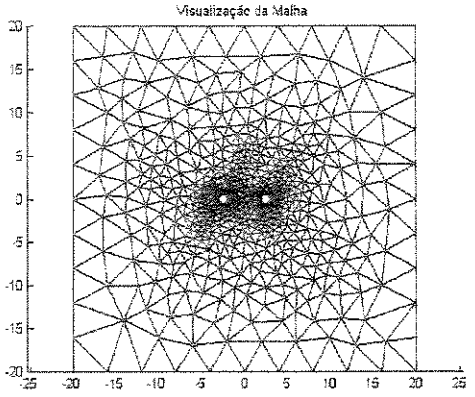

Fig. 4-21: Discretização Média - 669 nós

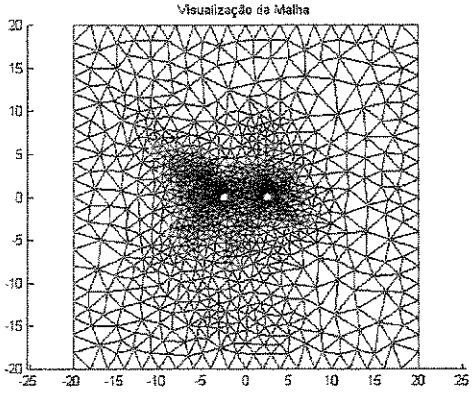

Fig. 4-22: Discretização Alta - 1008

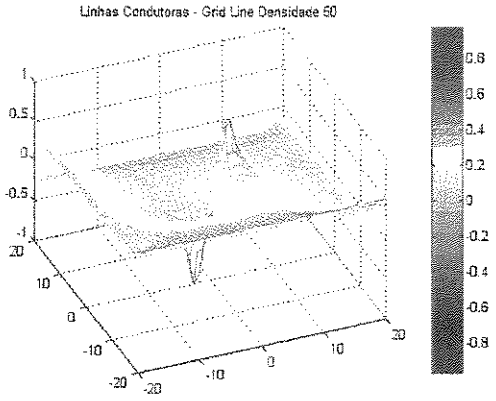

Fig. 4-23: Reresentação 3D da distribuição do Campo.

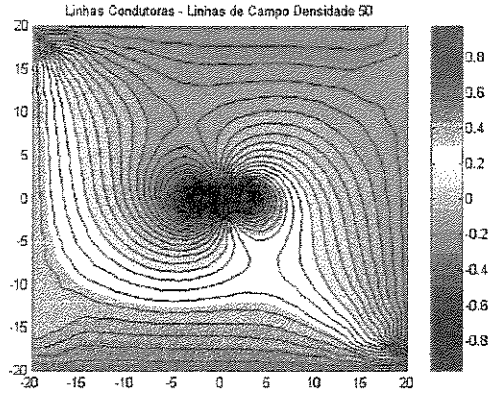

Fig. 4-24: Reresentação em Pseudo Cores + Curvas de Nível da distribuição do Campo.

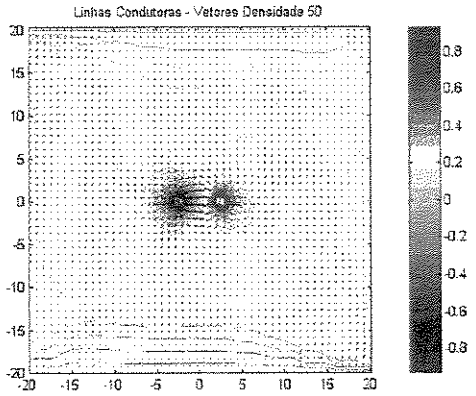

Fig. 4-25: Reresentação em, Curvas de Nível + Indicação do Gradiente, da distribuição do Campo.

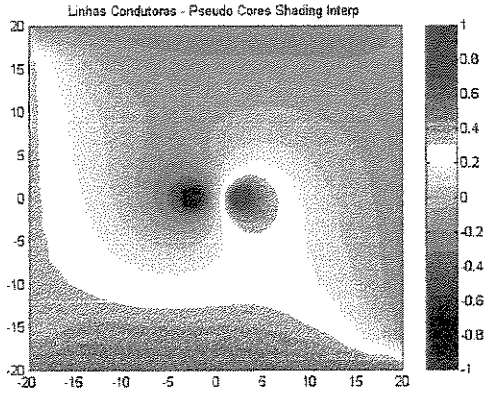

Fig. 4-26: Reresentação em Pseudo Cores da distribuição do Campo.

stituir a seqüência CalBack por digitação na linha de comandos.

```
'Callback',[...
'f=zeros(length(elem),1);,'...
'ff=[1 2 3 4 5 6];, '...  % Escolha das Fronteiras
'[K,b]=mefasble2d(elem,nodes,1,1,0,f);,'...
'p1=str2double(get(V0,''String''));,' ...
'p2=str2double(get(V1,''String''));,' ...
'p3=str2double(get(V2,''String''));,' ...
'p4=str2double(get(V3,''String''));,' ...
'p5=str2double(get(L0,''String''));,' ...
'p6=str2double(get(L1,''String''));,' ...
'pp= [p1,p2,p3,p4,p5,p6];,'...  % Atribuindo-se potencial as fronteiras
'[K,b]=mefdchlt2d(K,b,bound,nodes,ff,pp);,'...
'sol=mefsolve(K,b);,'], ...
```

4.5.3 Barras Condutoras

Similarmente ao problema das cargas no espaço, temos o problema de duas barras condutoras, submetidas a uma tensão. Desejamos saber qual a distribuição do potencial entre ambas.

Mais uma vez temos uma interface gráfica para tal problema (figura 4-27), e pelas mesmas razões apresentadas anteriormente, disponibilizamos duas malhas representadas nas figuras 4-28 e 4-29.

Média	$\Rightarrow$	483 nós
Alta	$\Rightarrow$	978 nós

Como se pode observar, com o auxílio da interface é possível alterar a tensão sobre cada barra. Para este exemplo utilizamos $B_1 = +1v$ e $B_2 = -1v$, obtendo os resultados apresentados nas figuras 4-30, 4-31, 4-32 e 4-33

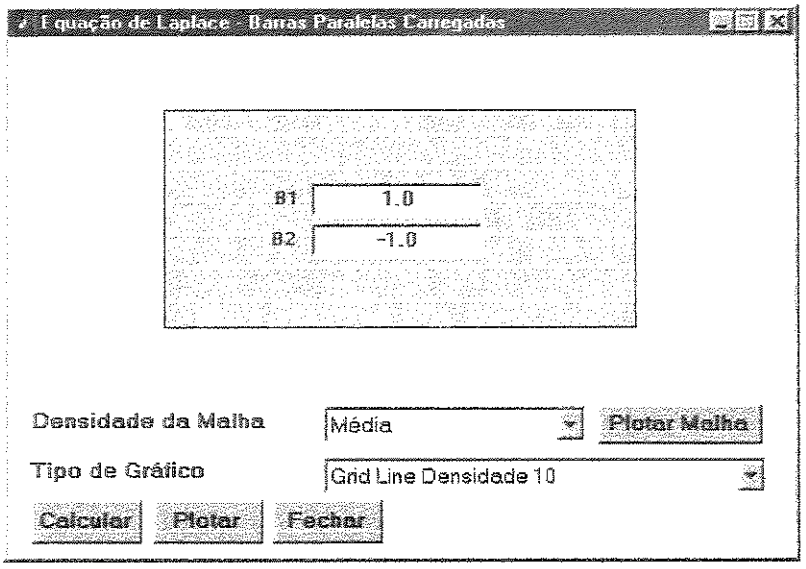

Fig. 4-27: Interface Gráfica - Barras Condutoras

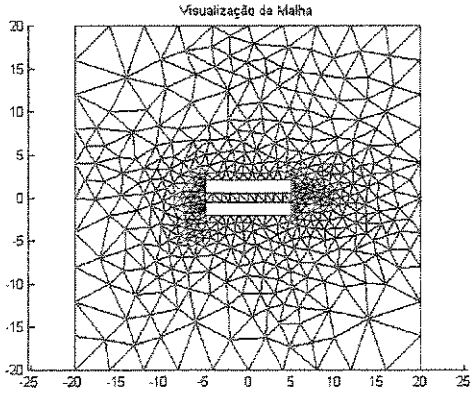

Fig. 4-28: Discretização Média - 483 nós

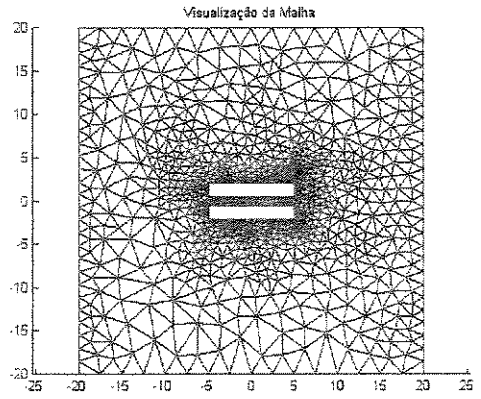

Fig. 4-29: Discretização Alta - 978 nós

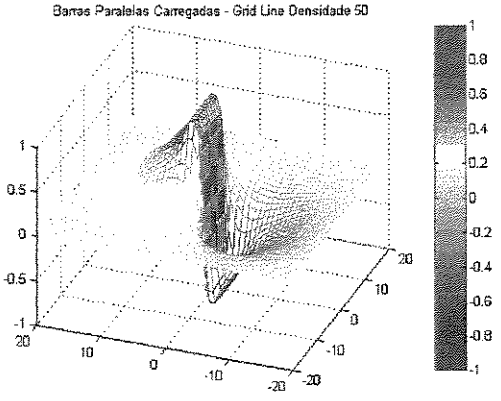

Fig. 4-30: Reresentação 3D da distribuição do Campo.

A seguir apresentamos o trecho do programa que executa os calculos. Podemos substituir a seqüência `CalBack` por digitação na linha de comandos.

```
'Callback',[...
'ff=[1 2 3];, '...
'f=zeros(length(elem),1);,'...
'[K,b]=mefasble2d(elem,nodes,1,1,0,f);,'...
```

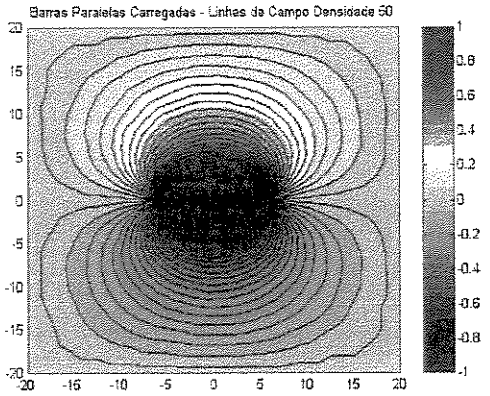

Fig. 4-31: Reresentação em Pseudo Cores + Curvas de Nível da distribuição do Campo.

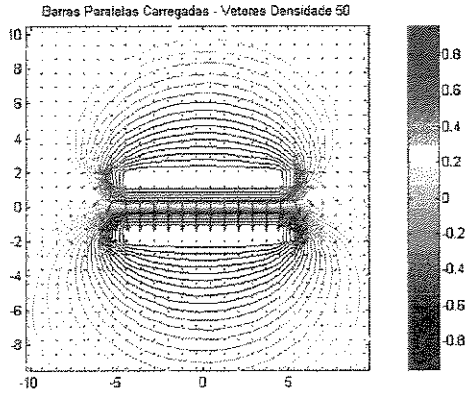

Fig. 4-32: Reresentação em, Curvas de Nível + Indicação do Gradiente, da distribuição do Campo.

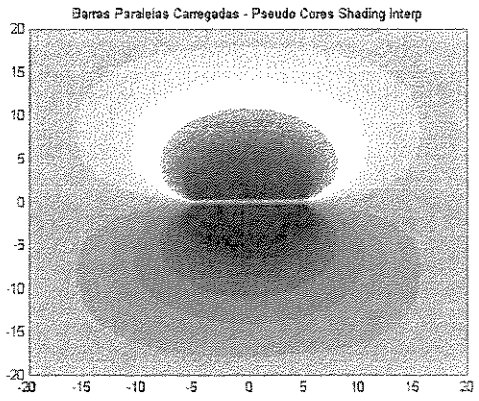

Fig. 4-33: Reresentação em Pseudo Cores da distribuição do Campo.

```
'p1=0;,' ...
'p2=str2double(get(Q0,''String''));,' ...
'p3=str2double(get(Q1,''String''));,' ...
'pp= [p1,p2,p3];,'...
'[K,b]=mefdchlt2d(K,b,bound,nodes,ff,pp);,'...
'sol=mefsolve(K,b);,'], ...
```

4.5.4 Distribuição de Potencial

Nos exemplos anteriores, haviamos trabalhado exclusivamente com a equação de Laplace. Esta se aplica em problemas eletrostáticos nos quais toda a carga reside sobre a superfície dos condutores. Ou seja: $\rho = 0$ para todo o domínio.

Agora, consideremos um problema eletrostático no qual parte da carga será dada por $\rho(x, y) \neq 0$, como uma função conhecida, e o resto das cargas reside na superfície dos condutores. Este problema requer a solução da equação de Poisson.

$$-\frac{\partial}{\partial x}\left(\varepsilon_x \frac{\partial \phi}{\partial x}\right) - \frac{\partial}{\partial y}\left(\varepsilon_y \frac{\partial \phi}{\partial y}\right) = \frac{\rho}{\varepsilon}$$

Para ilustrarmos de forma mais abrangente a solução de problemas deste tipo, desenvolvemos uma função chamada poiscom.m que associa todas as funções da meftool para 2D, de forma que dado um domínio arbitrário com n regiões com características diferentes ($\varepsilon_x, \varepsilon_y$ por região), pode-se aplicar uma fonte de excitação em qualquer uma destas regiões ($\rho \neq 0$) obtendo-se assim uma grante quantidade de combinações, englobando um vasto conjunto de situações.

A chamada a esta função é feita da seguinte forma:

`poiscomp(`ε_x R1,ε_y R1,ε_x R2,ε_y R2,ε_x R3,ε_y R3,ρ,`arquivo,potencial na fronteira)`

onde:

- ε_x R1,ε_y R1,ε_x R2,ε_y R2,ε_x R3,ε_y R3 $\Rightarrow$ Constantes dielétricas em três regiões distintas;
- $\rho \Rightarrow$ Densidade volumétrica de carga;
- `Arquivo` $\Rightarrow$ Arquivo com a malha, gerado pelo GemMesh;
- `potencial na fronteira` $\Rightarrow$ Potencial na fronteira mais externa do domínio.

Para fins de demonstração, desenvolvemos duas interfaces gráficas, ilustradas nas figuras 4-34 e 4-35, que levam em consideração três regiões distintas. Na primeira temos uma situação com simetria bem definida e, na segunda, temos a uma situação com quebra de simetria.

Utilizando-se a constate dielétrica do vidro ($\varepsilon_x, \varepsilon_y = 4$) na região 2 de ambas interfaces, obtivemos:

4.6 Considerações Finais

Buscamos neste Capítulo agrupar uma variada seleção de exemplos visando, principalmente, ilustrar o funcionamento e as características da `meftool`.

É importante salientar que todas as funções da `meftool` foram utilizadas, e que as interfaces gráficas, com exceção do BPM, são dedicadas apenas ao problema proposto

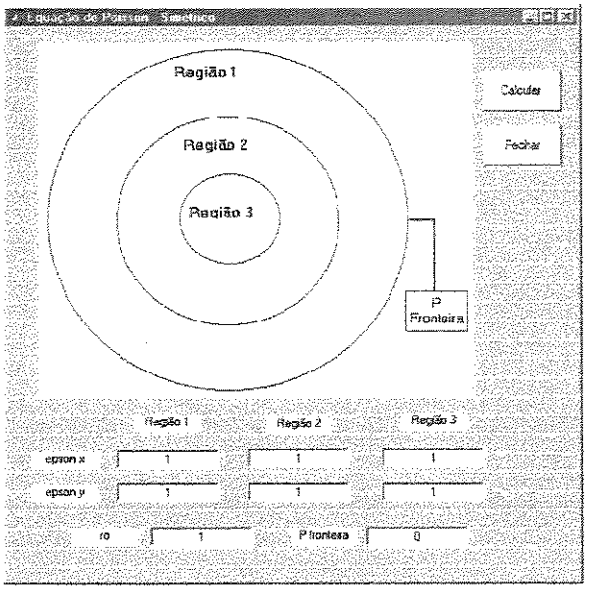

Fig. 4-34: Interface gráfica para um caso simétrico

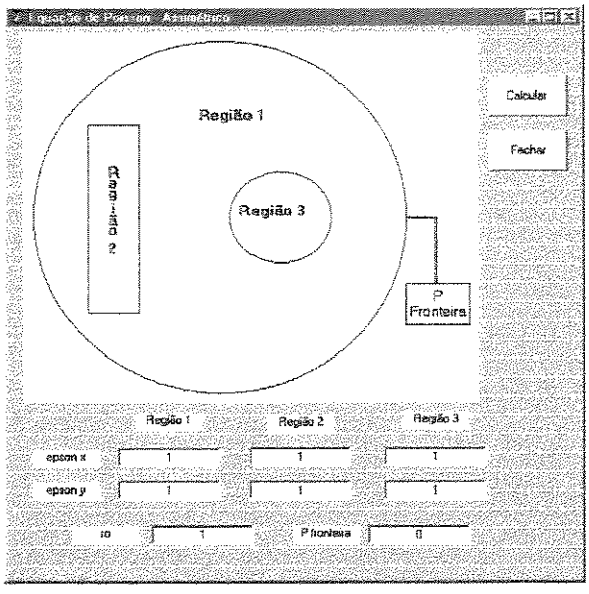

Fig. 4-35: Interface gráfica para um caso Assimétrico

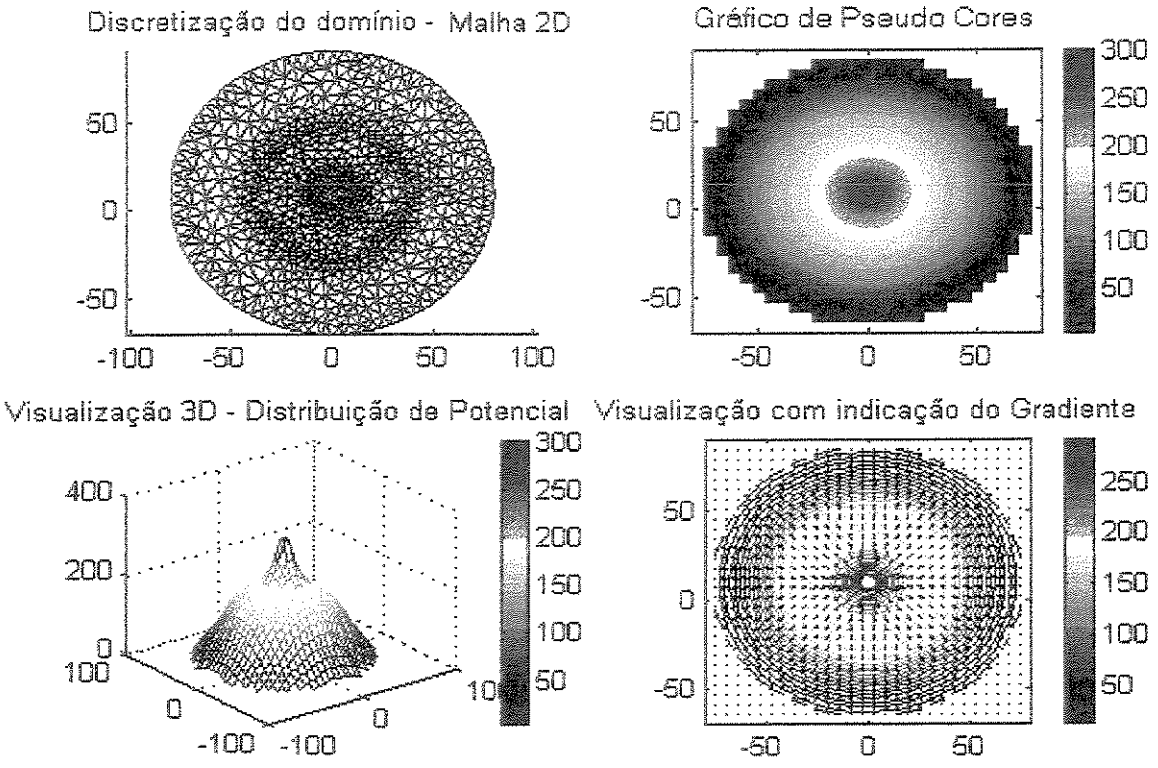

Fig. 4-36: Saida gráfica para o caso simétrico

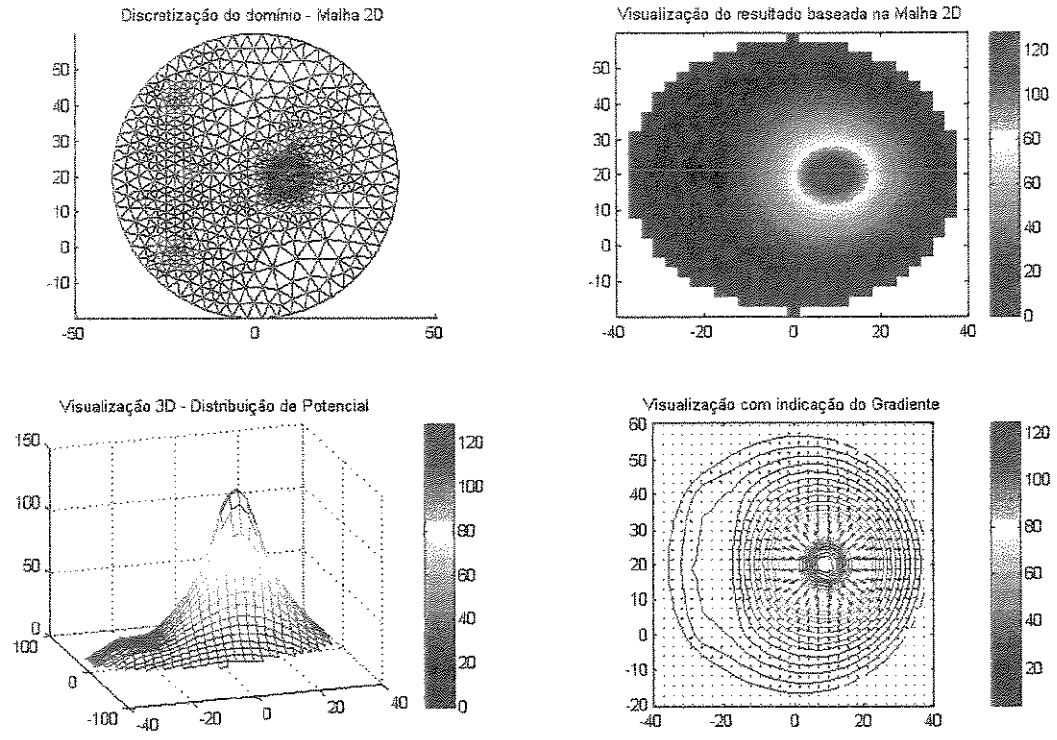

Fig. 4-37: Saída gráfica para um caso assimétrico.

no exemplo. No entanto a `meftool` vai muito além do escopo dos exemplos pois, como já mencionado, estamos apresentando um conjunto de funções que pode interagir diretamente com o ambiente MATLAB ou, de uma forma mais clara, uma vez instalada a `meftool` passa a fazer parte do ambiente

Sendo assim, pode-se utilizar a `meftool` de forma isolada, em conjunto com outras funções ou como componente de outros aplicativos.

Capítulo 5

Conclusão

Nos foi possível observar durante o desenvolvimento deste trabalho, que o MATLAB é fortemente recomendável para se implementar algorítimos de simulação pelo método de elementos finitos, devido especialmente a sua grande capacidade de álgebra matricial e visualização gráfica.

Um outro ponto extremamente relevante é que, o custo de desenvolvimento é muito baixo, uma vez que sendo o MATLAB um ambiente com muitas funcionalidades, permite ao usuário se concentrar principalmente no problema que deseja analisar ao invés de desenvolver todas as rotinas pertinentes.

A versão do MATLAB utilizada neste trabalho foi a 5.3, que inclui mais de 500 funções matemáticas, de engenharia e de análise de dados. Estas funcionalidades aliadas a uma poderosa linguagem como ferramenta de desenvolvimento de aplicações, permitiram-nos criar um ambiente para simulações pelo Método de Elementos Finitos, bastante versátil e funcional.

Sabemos que algumas lacunas ficaram por ser preenchidas, como por exemplo o desenvolvimento de um "CAD malhador ", ou a montagem de funções para análise 3D, elementos de aresta ou ainda problemas de auto-valor. No entanto consideramos agora o caminho aberto para que mais pessoas possam colaborar, principalmente se levarmos em conta o grau de modularidade da `meftool`.

Afora o supra citado, entendemos ainda como seqüência deste trabalho a geração de

bibliotecas para C/C++ ou FORTRAN.

Consideramos a `meftool` como a mais relevante contribuição deste trabalho. Com ela é tanto possível aprender os conceitos básicos do método de elementos finitos, quanto realizar complexas análises e simulações. Dentro da `meftool`, o ponto mais importante é o visualizador gráfico contido nas funções `mefgrid2d` e `mefview2d` pois, com elas, podemos visualizar não só as saídas da `meftool`, mas qualquer outro resultado obtido com qualquer outra função ou aplicativo.

Estando a `meftool` inserida no ambiente do MATLAB, é perfeitamente possível interagir com outras funcionalidades do mesmo, tornando-a bastante poderosa e abrangente. Outras características importantes da `meftool` são a modularidade e a portabilidade, pois é muito simples agregar novas funções e, sendo interpretada dentro do ambiente MATLAB, a mesma é independente do sistema operacional, podendo ser executada sem nenhuma alteração em todas as plataformas que tenham o MATLAB instalado.

Do ponto de vista da análise do MATLAB como ambiente de desenvolvimento, ficam as seguintes observações:

- O MATLAB é extremamente versátil no que diz respeito a desenvolver e testar algorítimos específicos, mas é desaconselhável como única ferramenta para desenvolvimento de interfaces com usuários;

- O conjunto de funções para geração de `mex-files` é de vital importância quando o objetivo é ganho de performance, pois ou as funções ficam compiladas e disponíveis para uso no próprio ambiente do MATLAB ou, convertemos tudo para fontes C/C++ ou FORTRAN para uso em aplicações "standalone " extrapolando em muito suas atuais limitações;

- A integração entre diferentes rotinas para visualização de resultados constituem um atrativo à parte, pois permite ao usuário criar um conjunto muito eficiente de pós processamento.

Por fim, recentemente chegou-nos ao conhecimento, o lançamento da versão 6.2 do MATLAB. De antemão soubemos que é possível compilar fontes C/C++ dentro do próprio

ambiente do MATLAB. Esta versão conta ainda com aprimoramento das rotinas gráficas e otimização de cálculos matriciais. É certo que estaremos adaptando a `meftool` a estas melhorias em momento oportuno.

Apêndice A

Método de Elementos Finitos

A.1 Apresentação e Formulações

A.1.1 Introdução ao MEF

O Método de Elementos Finitos (MEF) é uma técnica numérica para a obtenção de soluções aproximadas para problemas de valores de contorno em física e matemática e, consequentemente, em engenharia.

O método consiste basicamente em sub-dividir o domínio do problema em um número finito de sub-domínios conhecidos como elementos, nos quais atua uma função simples. Em geral, tais funções são lineares e as incógnitas do problema original, são os seus coeficientes. Desta forma, obteremos um sistema de equações que pode ser expresso e resolvido na forma matricial.

Uma característica muito importante do processo de subdivisão do domínio, também conhecido como malhagem é que, dependendo da forma da malha, a solução pode convergir com velocidade variada.

De forma resumida, podemos definir os seguintes passos básicos para a solução de um problema com o uso de elementos finitos :

1. Discretizar (sub-dividir) o domínio

2. Selecionar as funções de Interpolação

3. Formular o sistema de equações

4. Solucionar o sistema de equações

Veremos neste capítulo o modelamento do método para solução de problemas 1D e 2D.

A.1.2 Formulação 1D

Definição do problema

Consideraremos inicialmente o problema de valor de contorno que é definido pela seguinte equação diferencial

$$\begin{aligned} -\frac{d}{dx}\left(\alpha\frac{d\phi}{dx}\right)+\beta\phi &= f \\ x &\in (0,L) \end{aligned} \tag{A.1}$$

Onde ϕ é uma função desconhecida, α e β são parâmetros conhecidos, associados as propriedades físicas e f é uma fonte ou excitação.

As condições de contorno para ϕ são dadas por

$$\phi|_{x=0} = p \tag{A.2}$$

e

$$\left[\alpha\frac{d\phi}{dx}+\gamma\phi\right]_{x=L} = q \tag{A.3}$$

Onde p, γ e q são parâmetros conhecidos.

A equação **(A.2)** é conhecida como condição de Dirichilet ou essencial e a equação **(A.3)** é conhecida como condição mista. Nesta equação, quando $\gamma = 0$, temos a condição conhecida por condição de Neumann.

Análise por elementos finitos

O primeiro passo, é a dicretização do domínio, que para esta análise será feita da seguinte forma:

- Subdividiremos o domínio em M segmentos de comprimento l chamados elementos
- Os pontos extremos de cada elemento será chamado de nó, onde para o elemento em sí, teremos uma numeração local x_1 e x_2
- Cada elemento será batizado de e.

De acordo com a formulação abordada por Jianming Jin [1] temos que, para as funções de base, podemos usar :

$$\begin{aligned} N_1^e(x) &= \frac{x_2^e - x}{l^e} \\ N_2^e(x) &= \frac{x - x_1^e}{l^e} \end{aligned} \tag{A.4}$$

$$\phi^e(x) = \sum_{j=1}^{2} N_j^e(x)\phi_j^e \tag{A.5}$$

Onde e representa o elemento, l representa o comprimento do elemento, dado por $l^e = x_2^e - x_1^e$, sendo que x_1^e e x_2^e representam os pontos extremos do segmento (elemento).

Se observarmos a figura A-1 podemos concluir que $N_j^e(x_i^e) = \delta_{ij}$ onde $\delta_{ij} = 1$ para $i = j$ e $\delta_{ij} = 0$ para $i \neq j$.

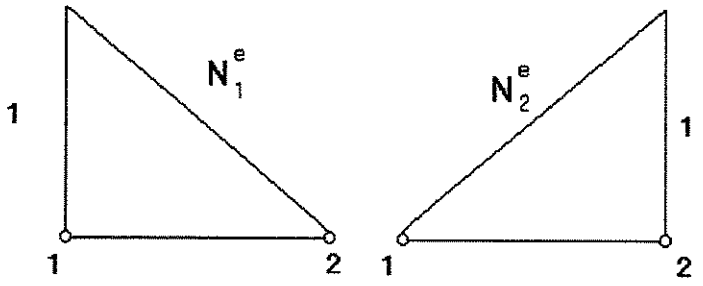

Fig. A-1: Funções de Interpolação para problemas 1D linear.

Já para a derivação das equações elementares, podemos utilizar o funcional para a equação **(A1)** escrito na seguinte forma:

$$F(\phi) = \sum_{e=1}^{M} F^e(\phi^e) \tag{A.6}$$

Onde

$$F^e(\phi^e) = \frac{1}{2}\int_{x_1^e}^{x_2^e} \left[\alpha\left(\frac{d\phi^e}{dx}\right)^2 + \beta\left(\phi^e\right)^2\right] dx - \int_{x_1^e}^{x_2^e} \phi^e f dx \tag{A.7}$$

Substituindo a equação **(A.5)** na equação **(A.7)** e derivando-se F^e com respeito a ϕ_i^e obtemos:

$$\frac{\partial F^e}{\partial \phi_i^e} = \sum_{j=1}^{2} \phi_j^e \int_{x_1^e}^{x_2^e} \left(\alpha \frac{dN_i^e}{dx}\frac{dN_j^e}{dx} + \beta N_i^e N j^e\right) dx - \int_{x_1^e}^{x_2^e} N_i^e f dx \tag{A.8}$$

Que pode ser escrito na forma matricial como:

$$\left\{\frac{\partial F^e}{\partial \phi^e}\right\} = [K^e]\{\phi^e\} - \{b^e\} \tag{A.9}$$

Sendo assim $\{\phi^e\} = [\phi_1^e, \phi_2^e]^T$ e

$$K_{ij}^e = \int_{x_1^e}^{x_2^e} \left(\alpha \frac{dN_i^e}{dx}\frac{dN_j^e}{dx} + \beta N_i^e N j^e\right) dx \tag{A.10}$$

$$b_i^e = \int_{x_1^e}^{x_2^e} N_i^e f dx \tag{A.11}$$

Pode-se notar que $[K^e]$ é simétrica e se α e β forem constantes, podemos escrever a matriz elementar na forma analítica como sendo:

$$K_{11}^e = K_{22}^e = \frac{\alpha^e}{l^e} + \beta\frac{l^e}{3} \tag{A.12}$$

$$K_{12}^e = K_{21}^e = -\frac{\alpha^e}{l^e} + \beta\frac{l^e}{6} \tag{A.13}$$

$$b_1^e = b_2^e = f^e\frac{l^e}{2} \tag{A.14}$$

A partir de então, podemos montar o sistema de equações, tendo em mente que o mesmo será obtido somando-se todos os elementos expostos na equação **(A.9)** e impondo-se a condição de estacionaridade, teremos:

$$\left\{\frac{\partial F}{\partial \phi}\right\} = \sum_{e=1}^{M} \left\{\overline{\frac{\partial F^e}{\partial \phi}}\right\} = \sum_{e=1}^{M} \left(\overline{\left[K^e\right]} \left\{\overline{\phi^e}\right\} - \left\{\overline{b^e}\right\} \right) = 0 \tag{A.15}$$

De onde finalmente obtemos:

$$K_{11} = K_{11}^{(1)} = \frac{\alpha^{(1)}}{l^{(1)}} + \beta^{(1)} \frac{l^{(1)}}{3} \tag{A.16}$$

$$K_{NN} = K_{22}^{(M)} = \frac{\alpha^{(M)}}{l^{(M)}} + \beta^{(M)} \frac{l^{(M)}}{3} \tag{A.17}$$

$$\begin{aligned} K_{ii} &= K_{22}^{(i-1)} + K_{11}^{(i)} = \frac{\alpha^{(i-1)}}{l^{(i-1)}} + \beta^{(i-1)} \frac{l^{(i-1)}}{3} + \frac{\alpha^{(i)}}{l^{(i)}} + \beta^{(i)} \frac{l^{(i)}}{3} \\ i &= 2, 3, ..., N-1 \end{aligned} \tag{A.18}$$

$$\begin{aligned} K_{i+1,j} &= K_{i,j+1} = K_{12}^{(i)} = \frac{\alpha^{(i)}}{l^{(i)}} + \beta^{(i)} \frac{l^{(i)}}{6} \\ i &= 1, 2, 3, ..., N-1 \end{aligned} \tag{A.19}$$

$$b_1 = b_1^{(1)} = f^{(1)} \frac{l^{(1)}}{2} \tag{A.20}$$

$$b_N = b_2^{(M)} = f^{(M)} \frac{l^{(M)}}{2} \tag{A.21}$$

$$\begin{aligned} b_i &= b_2^{(i-1)} + b_1^{(i)} = f^{(i-1)}\frac{l^{(i-1)}}{2} + f^{(i)}\frac{l^{(i)}}{2} \\ i &= 2,3,4,...,N-1 \end{aligned} \tag{A.22}$$

Dadas estas equações, falta-nos apenas incorporar as condições de contorno.

Iniciaremos aplicando a condição explicitada na equação **(A.3)** modificando as equações **(A.17)** e **(A.21)** somando-se γ e q respectivamente, obtendo-se:

$$K_{NN} = \frac{\alpha^{(M)}}{l^{(M)}} + \beta^{(M)}\frac{l^{(M)}}{3} + \gamma \tag{A.23}$$

$$b_N = f^{(M)}\frac{l^{(M)}}{2} + q \tag{A.24}$$

E, para aplicarmos a condição explicitada na equação **(A.2)** temos o seguinte procedimento:

$$\begin{aligned} b_i &\leftarrow b_i + K_{i,1}p \\ K_{i,1} &= 0 \\ i &= 2,3,4,...,N \end{aligned} \tag{A.25}$$

O próximo passo agora, será o de resolver o sistema de equações.

Um outro detalhe muito importante, é o fato de que a análise descrita acima, é referente a elementos lineares apenas, pois para elementos de ordem superior, devemos ajustar as equações.

A.1.3 Formulação 2D

Definição do Problema

Analogamente a análise 1D, iniciaremos a formulação para 2D, à partir do problema de valor de contorno, definido pela seguinte equação diferencial:

$$-\frac{\partial}{\partial x}\left(\alpha_x \frac{\partial \phi}{\partial x}\right) - \frac{\partial}{\partial y}\left(\alpha_y \frac{\partial \phi}{\partial y}\right) + \beta\phi = f \tag{A.26}$$

Onde ϕ é uma função desconhecida, α_x, α_y, β são parâmetros associados às propriedades físicas do domínio podendo ser complexos e, f é uma fonte ou função de excitação. As equações de Laplace, Poison e Helmholtz são casos especiais da equação **(A.26)**.

Como condições de contorno para este problema temos a condição essencial ou de Dirichlet dada por:

$$\begin{aligned} \phi &= p \\ &\text{em } \Gamma_1 \end{aligned} \tag{A.27}$$

E a condição mista é dada por:

$$\begin{aligned} \left(\alpha_x \frac{\partial \phi}{\partial x}\widehat{x} + \alpha_y \frac{\partial \phi}{\partial y}\widehat{y}\right) \cdot \widehat{n} + \gamma\phi &= q \\ &\text{em } \Gamma_2 \end{aligned} \tag{A.28}$$

Onde Γ_1 e Γ_2 são as fronteiras a serem analisadas, p, q e γ são constantes físicas definidas pelas características do problema.

Quando $\gamma = 0$ dizemos que a equação **(A.28)** é a condição homogênea de Neumann.

Análise por Elementos Finitos

Da mesma forma como procedemos na análise 1D, nosso primeiro passo será o de discretizar o domínio, que, como se trata de um domínio bi-dimensional, deveremos utilizar elementos bi-dimensionais. Este processo chama-se *malhagem*, e o elemento mais comum é o triangular.

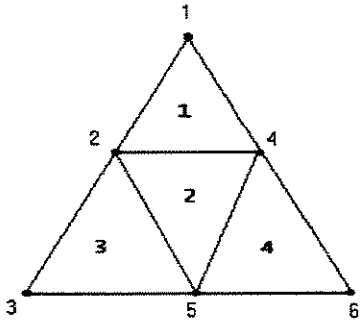

Fig. A-2: Subdivisão de um domínio em quatro elementos triangulares (negrito) com seis nós (normal)

Baseando-se na figura A-2 podemos construir um arranjo de $3 \times M$ denominado $n(i,e)$, onde $i = 1,2,3$ e $e = 1,2,3,...,M$ com M denotando o número total de elementos (neste caso $M = 4$), com $n(i,e)$ sendo o número total de nós, i a numeração local dos nós e e o número do elemento. Por convenção, enumeramos os nós locais de um elemento no sentido anti-horário.

e	$n(1,e)$	$n(2,e)$	$n(3,e)$
1	2	4	1
2	5	4	2
3	3	5	2
4	5	6	4

(Tabela A1)

Analogamente à formulação uni-dimensional, podemos aproximar a função incógnita ϕ para cada elemento, que neste caso serão utilizados elementos triangulares.

Podemos então representar ϕ como sendo:

$$\phi^e(x,y) = a^e + b^e x + c^e y \tag{A.29}$$

Onde a^e, b^e e c^e são os coeficientes a serem determinados, e é o número do elemento e (x,y) representam as coordenadas do nó.

Podemos especificar ϕ para cada nó local do elemento (1,2,3) da seguinte forma:

$$\begin{aligned} \phi_1^e &= a^e + b^e x_1^e + c^e y_1^e \\ \phi_2^e &= a^e + b^e x_2^e + c^e y_2^e \\ \phi_3^e &= a^e + b^e x_3^e + c^e y_3^e \end{aligned} \tag{A.30}$$

Resolvendo-se para os coeficientes a^e, b^e e c^e em termos de ϕ_j^e e substituindo na equação **(A.29)** obtemos:

$$\phi^e(x,y) = \sum_{j=1}^{3} N_j^e(x,y)\phi_j^e \tag{A.31}$$

Onde $N_j^e(x,y)$ representa a função de interpolação, e é dado por:

$$\begin{aligned} N_j^e(x,y) &= \frac{1}{2\Delta^e}\left(a_j^e + b^e x_j^e + c^e y_j^e\right) \\ j &= 1,2,3 \end{aligned} \tag{A.32}$$

com

$$
\begin{array}{lll}
a_{1=}^e x_2^e y_3^e - y_2^e x_3^e \; ; & b_1^e = y_2^e - y_3^e \; ; & c_1^e = x_3^e - x_2^e \\
a_{2=}^e x_3^e y_1^e - y_3^e x_1^e \; ; & b_2^e = y_3^e - y_1^e \; ; & c_2^e = x_1^e - x_3^e \\
a_{3=}^e x_1^e y_2^e - y_1^e x_2^e \; ; & b_3^e = y_1^e - y_2^e \; ; & c_3^e = x_2^e - x_1^e
\end{array} \tag{A.33}
$$

e

$$
\Delta^e = \frac{1}{2}\begin{vmatrix} 1 & x_1^e & y_1^e \\ 1 & x_2^e & y_2^e \\ 1 & x_3^e & y_3^e \end{vmatrix} = \frac{1}{2}\left(b_1^e c_2^e - b_2^e c_1^e\right) \tag{A.34}
$$

$$
\Rightarrow \quad \text{Área do e-ésimo elemento}
$$

No exposto acima, x_j^e e y_j^e com $(j = 1, 2, 3)$ denotam as coordenadas do j-ésimo nó do e-ésimo elemento. Uma observação importante é que a função de interpolação tem a seguinte propriedade:

$$
N_i^e\left(x_j^e, y_j^e\right) = \delta_{ij} = \left\{ \begin{array}{l} 1 \leftrightarrow i = j \\ 0 \leftrightarrow i \neq j \end{array} \right\} \tag{A.35}
$$

Temos agora os componentes necessários para a composição das matrizes elementares, e para isso podemos recorrer mais uma vez ao método abordado para 1D, tendo como ponto de partida o funcional:

$$
F(\phi) = \sum_{e=1}^{M} F^e(\phi^e) \tag{A.36}
$$

Sendo que desta vez F^e será dado por:

$$F^e(\phi^e) = \frac{1}{2}\int\int_{\Omega^e}\left[\alpha_x\left(\frac{\partial\phi^e}{\partial x}\right)^2 + \alpha_y\left(\frac{\partial\phi^e}{\partial y}\right)^2 + \beta\left(\phi^e\right)^2\right]d\Omega - \int\int_{\Omega^e} f\phi^e d\Omega \qquad \text{(A.37)}$$

Aqui, Ω^e representa o domínio do e-ésimo elemento.

Introduzindo-se a equação **(A.31)** diferenciando-se F^e com respeito a ϕ_j^e obtemos:

$$\begin{aligned}\frac{\partial F^e}{\partial \phi_j^e} &= \sum_{j=1}^{3}\int\int_{\Omega^e}\left[\alpha_x\frac{\partial N_i^e}{\partial x}\frac{\partial N_j^e}{\partial x} + \alpha_y\frac{\partial N_i^e}{\partial y}\frac{\partial N_j^e}{\partial y} + \beta N_i^e N_j^e\right]d\Omega \\ &\quad - \int\int_{\Omega^e} f N_i^e d\Omega \qquad \text{(A.38)} \\ i &= 1,2,3\end{aligned}$$

Que pode ser escrita na forma matricial como segue

$$\left\{\frac{\partial F^e}{\partial \phi_j^e}\right\} = \left[K^e\right]\left\{\phi^e\right\} - \left\{b^e\right\} \qquad \text{(A.39)}$$

Onde

$$\left\{\frac{\partial F^e}{\partial \phi_j^e}\right\} = \left[\frac{\partial F^e}{\partial \phi_1^e}, \frac{\partial F^e}{\partial \phi_2^e}, \frac{\partial F^e}{\partial \phi_3^e}\right]^T \qquad \text{(A.40)}$$

e

$$\left\{\phi^e\right\} = \left[\phi_1^e, \phi_2^e, \phi_3^e\right]^T \qquad \text{(A.41)}$$

Sendo assim, a matriz elementar $\left[K_{ij}^e\right]$ será dada por:

$$\begin{aligned} [K_{ij}^e] &= \int\int_{\Omega^e} \left[\alpha_x \frac{\partial N_i^e}{\partial x} \frac{\partial N_j^e}{\partial x} + \alpha_y \frac{\partial N_i^e}{\partial y} \frac{\partial N_j^e}{\partial y} + \beta N_i^e N_j^e \right] dxdy \qquad \text{(A.42)} \\ i &= 1,2,3 \end{aligned}$$

e o vetor $\{b_i^e\}$ será dado por:

$$\begin{aligned} \{b_i^e\} &= \int\int_{\Omega^e} f N_i^e dxdy \qquad \text{(A.43)} \\ i &= 1,2,3 \end{aligned}$$

Que na forma analítica poderão serem representadas por:

$$K_{ij}^e = \frac{1}{4\Delta^e} \left(\alpha_x^e b_i^e b_j^e + \alpha_y^e c_i^e c_j^e \right) + \frac{\Delta^e}{12} \beta^e \left(1 + \delta_{ij} \right) \qquad \text{(A.44)}$$

e

$$b_i^e = \frac{\Delta^e}{3} f^e \qquad \text{(A.45)}$$

Podemos agora montar o sistema de equações, e para isso basta proceder da seguinte forma:

- Baseando-se na matriz conectividade apresentada na tabela **(Tabela A1)** onde $n(i,e)$ representa um nó global, com i representando um nó local do elemento e montaremos a matriz global $[K]$ como segue:

$$K_{n(i,e),n(j,e)} = K_{n(i,e),n(j,e)} + K_{i,j}^e \qquad \text{(A.46)}$$

- E para o vetor global $\{b\}$ procedemos também como segue:

$$b_{n(i,e)} = b_{n(i,e)} + b_i^e \tag{A.47}$$

Podemos finalmente incorporar as condições de contorno exigidas pelo problema. Começaremos pela condição natural apresentada pela equação **(A.28)**.

Para tanto, vamos considerar os lados dos elementos que compõem a fronteira a ser analisada. Cada um destes M_s segmentos, conterá dois nós locais, e cada um destes nós terá sua denominação global definida como $ns(i,s)$. Seguindo-se o exemplo apresentado na figura **(A-2)**, podemos construir a seguinte matriz de conectividade:

s	$ns(1,s)$	$ns(2,s)$
1	6	4
2	4	1
3	1	2
4	2	3

(A.48)

Para cada segmento, montaremos uma matriz elementar $[K^s]$ e um vetor elementar $\{b^s\}$ definidos por:

$$K_{ij}^s = \gamma^s \frac{l^s}{6}(1 + \delta_{ij}) \tag{A.49}$$

$$b_i^s = q^s \frac{l^2}{2} \tag{A.50}$$

Onde γ^s e q^s são constantes para cada segmento e são definidos pelas características

físicas do problema, l^s é o comprimento de cada segmento dado por :

$$l^s = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{A.51}$$

Esta formulação é detalhadamente demonstrada por Jianming Jin[**1**].

Para incorporar K^s_{ij} e b^s_i à matriz global, devemos simplesmente executar a seguinte rotina:

$$K_{ns(i,s),ns(j,s)} = K_{ns(i,s),ns(j,s)} + K^s_{i,j} \tag{A.52}$$

e

$$b_{ns(i,s)} = b_{ns(i,s)} + b^s_i \tag{A.53}$$

Finalmente para incorporarmos a condição essencial ou Dirichlet definida pela equação **(A.27)**, aplicamos o seguinte algorítmo:

$b_{nd(i)} = p(i)$
$K_{nd(i),nd(j)} = 1$
$K_{nd(i),j} = 0$ para $j \neq nd(i)$
e
$b_j \leftarrow b_j - K_{j,nd(i)}p(i)$
$K_{j,nd(i)} = 0$ para $j \neq nd(i)$

Desta forma, apresentamos um resumo das formulações para elementos finitos 1D e 2D apresentadas em detalhes por Jianming Jin **[1]**.

É importante frisar que existe ainda a possibilidade do uso de elementos de ordem

superior, e no caso de problemas 2D, podemos fazer uso de elementos quadriláteros e que, o sucesso nos resultados depende em muito do aspecto da malha aplicada.

Referências

[1] Jimiang Jin - The Finite Elements Method in Electromagnetics - Wiley-Iterscience Publication, 1993

[2] Peter P. Silvester and Ronald L. Ferrari - Finite Elements for electrical engineers 3^a edition - Cambridge Press, 1996

[3] Duane Hanselman and Bruce Lttlefield - Mastering MATLAB - Prentice-Hall, 1995

[4] Duane Hanselman and Bruce Lttlefield - MATLAB 5 Guia do Usuário - MAKRON Books, 1999

[5] George R. Buchanan - Finite Element Analysis 2^a edition - McGraw-Hill, 1995

[6] P. Silvester - Campos Eletromagnético Modernos - Editora Polígono, 1988

[7] David K. Cheng - Field and Wave Electromagnetics - Addison-Wesley Publishing Company, 1989

[8] Robert E. Collin - Engenharia de Microondas - Guanabara Dois, 1979

[9] Aloisio Ernesto Assan - Método dos Elementos Finitos Primeiros Passos, Editora da Unicamp, 1999

[10] Eikichi Yamashita - Analysis Methods for Electromagnetic Wave Problems - Artech House, 1993

[11] Kaijiro HyodoTese de Mestrado - Estudo do Método de Elementos Finitos Aplicado a Problemas de Eletromagnetismo - TM-02/98 UFPA/CT/CMEE

[12] Yasuyuki Arai, Akihiro Maruta e Masanori Matsuhara -Transparent Boundary for the Finite Element Beam Propagation Method , Optics Letters May 15, 1993/Vol.18,N° 10

[13] Calfem: A finite element *toolbox* to MATLAB - Version 3.2 - Division of Structural Mechanics LTH, Lund University & Division of Solid Mechanics LTH, Lund University, 1996

[14] Young W. Kwoun and Hyochoong Bang - The Finite Element Method using MATLAB, CRC Press - Naval Postgraduete School & Korea Aerospace Resarch Institute, 1996

[15] Masanore. Kosshiba - Finite Element Beam Propagation Method With Perfectly Matched Layer Boundary Conditions - IEEE Tans. Magnet, Vol 35, Mar. 1999

[16] H. E. Hernández Figueroa and C. E. Rubio Mercedes - "Transparent Boundary for the Finite Element Simulation of Temporal Soliton Propagation ". IEEE Transactions on Magnetics, Vol. 34, N°5, pp. 3228-3331, September 1998

[17] H. E. Hernández Figueroa e J. Patrocinio - Relatório Interno de Estudos Especiais - DMO/FEE/Unicamp - Dez/2000