LUCAS BLATTNER MARTINHO

Determinação, no Regime Senoidal, de Impedância e de Campos Eletromagnéticos em Sistemas de Aterramento pelo Método dos Elementos Finitos.

Relatório final de projeto de formatura apresentado à Escola Politécnica da Universidade de São Paulo para a obtenção do grau de Engenheiro

Orientadora:
Prof[a]. Dr[a]. Viviane Cristine Silva

Coordenadores:
Prof. Dr. Eduardo César Senger
Prof. Dr. Lourenço Matakas Jr.

São Paulo
2005

AGRADECIMENTOS

Primeiramente, agradeço à Professora Viviane Cristine Silva pela constante prontidão, paciência e solicitude na orientação deste trabalho.

Agradeço também ao Professor José Roberto Cardoso pela oportunidade única que tive de participar, ao longo dos três últimos anos do meu curso de graduação, das atividades do Laboratório de Eletromagnetismo Aplicado da Escola Politécnica da Universidade de São Paulo.

Finalmente, gostaria também de agradecer aos demais colegas e professores do Laboratório de Eletromagnetismo Aplicado por terem me oferecido, durante todo o período em que estive entre eles, um ambiente de trabalho marcado pelo respeito, pela amizade e pelo companheirismo.

ABSTRACT

An analysis of grounding systems subjected to high frequency current is presented. The vector finite element technique is the numerical method used in this study. Complementary computational tools for data input and output were developed. This complete set of programs is then applied to the study of two particular grounding system configurations. The results are compared with those reported by other authors, with an overall good agreement.

RESUMO

Uma análise de sistemas de aterramento submetidos a correntes em alta freqüência é apresentada. A técnica de elementos finitos vetoriais é o método numérico utilizado neste estudo. Ferramentas computacionais complementares de entrada e saída de dados foram desenvolvidas. Este conjunto completo programas é então aplicado ao estudo de duas configurações particulares de sistema de aterramento. Os resultados são comparados com aqueles produzidos por outros autores, com uma boa concordância geral verificada.

"Ja! Diesem Sinne bin ich ganz ergeben,
Das ist der Weisheit letzter Schluß:
Nur der verdient sich Freiheit wie das Leben,
Der täglich sie erobern muß.
Und so verbringt, umrungen von Gefahr,
Hier Kindheit, Mann und Greis sein tüchtig Jahr.
Solch ein Gewimmel möcht' ich sehn,
Auf freiem Grund mit freiem Volke stehn.
Zum Augenblicke dürft' ich sagen:
Verweile doch, du bist so schön!
Es kann die Spur von meinen Erdetagen
Nicht in Äonen untergehn. -
Im Vorgefühl von solchem hohen Glück
Genieß' ich jetzt den höchsten Augenblick."

Johann Wolfgang von Goethe
Faust: der Tragödie zweiter Teil
Fünfter Akt, *Großer Vorhof des Palasts*

SUMÁRIO

1. Introdução

A principal função de um sistema de aterramento é aquela de manter dentro de níveis aceitáveis de segurança os potenciais elétricos resultantes da circulação de correntes provenientes de faltas ou de impulsos atmosféricos em uma instalação elétrica. A violação destes níveis resulta numa condição de perigo aos seres humanos que circulam nas imediações do sistema de aterramento, uma vez que potenciais acima de limites bem estabelecidos podem vir a produzir correntes suficientemente elevadas e capazes de causar fibrilação ventricular numa eventual vítima. Desta forma, deseja-se através de um aterramento a obtenção da menor impedância possível entre o ponto de conexão à terra da instalação e certo ponto remoto de potencial nulo localizado sob o solo, minimizando assim as elevações de potencial observadas pela instalação na eventualidade da circulação de correntes de falta.

O estudo e a determinação das características dos sistemas de aterramento em operação na freqüência industrial tais como resistência, elevação de potencial e outros parâmetros têm sido suficientemente abordados na literatura [1][2]. Entretanto, em condições transitórias, como quando uma malha de aterramento está sujeita a um surto de corrente (provindo de uma descarga atmosférica, por exemplo), o cálculo de sua impedância, bem como da sua máxima elevação de potencial, oferecem um desafio ao projetista. Tais casos se diferenciam daqueles de faltas na freqüência industrial pelo fato de as correntes injetadas no sistema de aterramento nestas situações possuírem um conteúdo harmônico diversificado, especialmente no que diz respeito às altas freqüências do espectro.

O escoamento de correntes dotadas de componentes em alta freqüência por um sistema de aterramento resulta ainda na possibilidade de interferência eletromagnética na instrumentação e nos equipamentos presentes na vizinhança. Deste modo, o estudo da imposição deste tipo de solicitação a sistemas de aterramento e a determinação dos campos elétrico e magnético resultantes possuem um interesse e uma motivação adicionais, pois estas grandezas se tornam importantes na avaliação destes fenômenos de compatibilidade eletromagnética associados. Assim, nestas condições, deve-se recorrer a métodos numéricos para que estas grandezas de interesse sejam determinadas. Os métodos numéricos aplicáveis, por sua vez, classificam-se em duas categorias: aqueles que utilizam uma abordagem baseada na teoria de circuitos elétricos e aqueles que empregam a teoria eletromagnética, nos quais se parte para uma tentativa de resolução das equações de Maxwell que regem o fenômeno em questão.

Neste trabalho, foi feita a determinação de grandezas de interesse em sistemas de aterramento como a impedância complexa, campos eletromagnéticos e a distribuição de potenciais no nível do solo para algumas configurações particulares de aterramento submetidas a injeções de harmônicos de corrente em uma larga faixa de freqüências (até o limite superior de 1MHz). Por ser a mais genérica, a abordagem mencionada baseada na teoria eletromagnética foi a adotada, e o método numérico escolhido foi o método dos elementos finitos. Trata-se de um método consagrado na modelagem de uma vasta gama de fenômenos e dispositivos eletromagnéticos pela sua capacidade de levar em conta heterogeneidades, anisotropias, não-linearidades e geometrias complexas.

2. Objetivos

De acordo com o que se apresentará com detalhe na próxima seção, a caracterização das grandezas representativas dos sistemas de aterramento feita neste trabalho foi realizada empregando-se uma variante do método dos elementos finitos apropriada à consideração de problemas envolvendo altas freqüências. Tal variante encontrava-se implementada na forma de um programa desenvolvido no Laboratório de Eletromagnetismo Aplicado da Escola Politécnica da Universidade de São Paulo (LMAG) denominado HFG3D, que por sua vez carecia de ferramentas computacionais complementares para a tomada de dados de entrada e para a saída e visualização dos resultados calculados. Levando estes fatos adicionais em conta, os objetivos pretendidos com este trabalho encontram-se enumerados a seguir:

1) Desenvolver as ferramentas computacionais complementares de entrada de dados e de visualização de resultados necessárias à utilização do programa HFG3D no estudo de problemas de aterramento.

2) Aplicar o programa HFG3D em conjunto com as ferramentas computacionais desenvolvidas a configurações particulares de sistemas de aterramento selecionadas para a caracterização destas, considerando os casos em que estas são solicitadas por injeções de correntes harmônicas em alta freqüência. Os aterramentos particulares selecionados para este estudo correspondem ao caso de uma haste metálica verticalmente enterrada e ao de uma malha de aterramento quadriculada enterrada a uma certa profundidade do solo. Estes sistemas de aterramento serão oportunamente detalhados e descritos no desenvolvimento deste texto.

3) Comparar os resultados obtidos com resultados experimentais e com outras metodologias disponíveis na literatura [3] [4] [5], com a finalidade de se validar tanto as ferramentas computacionais auxiliares desenvolvidas quanto a abordagem numérica empregada.

Maiores detalhes a respeito dos programas auxiliares ao HFG3D mencionados serão apresentados adiante neste texto, quando então a natureza destes e a razão da necessidade por eles poderão ser esclarecidas com maior propriedade. Quanto à referida caracterização dos sistemas de aterramento desejada, esta consistiu na determinação do comportamento da impedância complexa de aterramento em função da freqüência para um intervalo de variação capaz de representar o conteúdo harmônico da corrente injetada por um raio . Também consistiu na determinação e na representação gráfica (através de gráficos de superfícies tridimensionais) para o nível do solo do comportamento e da distribuição de grandezas como o potencial elétrico e os campos eletromagnéticos resultantes da injeção destas correntes harmônicas em alta freqüência.

3. O Método dos Elementos Finitos aplicado a problemas em alta freqüência

O método dos elementos finitos é, genericamente, uma técnica numérica de resolução de equações diferenciais. Em particular, para aplicações de Engenharia Elétrica, estas equações diferenciais correspondem às equações de Maxwell, e formulações convencionais deste método numérico dedicadas a problemas de eletromagnetismo podem ser interpretadas como sendo a resolução deste conjunto de equações para cada um dos elementos componentes do domínio em estudo [6]. Tais elementos são obtidos a partir de uma discretização introduzida por uma malha dita de elementos finitos, e os resultados fornecidos por estes procedimentos numéricos são fundamentalmente os valores das grandezas eletromagnéticas (potencial elétrico e vetor potencial magnético) nos pontos que representam vértices desta malha. A determinação destas grandezas num ponto qualquer do domínio é realizada através de interpolação, e o método numérico assim formulado é dito "método de elementos finitos nodal". É a formulação usualmente disponível em programas comerciais como, por exemplo, o FLUX3D [7].

Os casos de estudo abordados neste trabalho correspondem, conforme já se colocou, à análise do comportamento de sistemas de aterramento submetidos a injeções de corrente em alta freqüência. Ocorre, no entanto, que a utilização de técnicas de elementos finitos nodais em aplicações de três dimensões envolvendo freqüências elevadas resulta, invariavelmente, numa deterioração da qualidade dos resultados obtidos. Isto ocorre em função do surgimento das chamadas "soluções espúrias", que são repostas sem significado físico que passam a figurar entre os valores das incógnitas calculados numericamente neste contexto [8] [9]. A título de exemplo, um caso afetado por estas soluções espúrias pode ser observado na figura 1. Trata-se do problema da determinação da distribuição do campo elétrico nas imediações de duas superfícies cilíndricas condutoras, uma delas maior e externa e a outra interna. A figura 1.a apresenta a solução prevista para este problema, enquanto a figura 1.b exibe uma solução numérica obtida pela técnica de elementos finitos nodal e que sofre do problema das soluções espúrias mencionado. Neste caso particular, pode-se observar que as soluções espúrias se manifestam através de um campo elétrico calculado com deformações em diversas regiões.

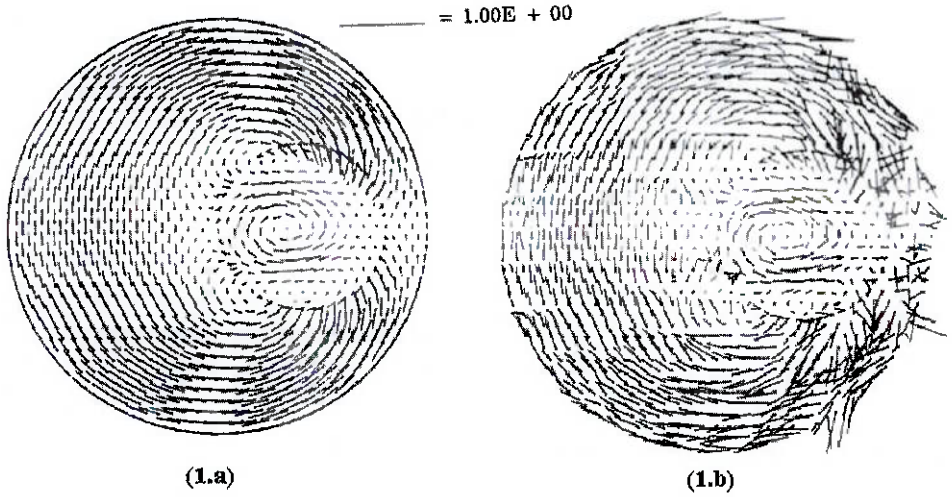

Figura 1.
A distribuição de campo elétrico esperada (1.a) e a solução numérica afetada por soluções espúrias (1.b) para o problema de dois cilindros

Uma maneira de se contornar estas dificuldades que surgem nos problemas de alta freqüência é fazer o uso de formulações alternativas do método dos elementos finitos. A variante desta técnica numérica conhecida como "método dos elementos finitos vetorial" (ou "de aresta"), por exemplo, é uma possibilidade. Nela, as incógnitas do problema a ser resolvido deixam de ser os valores de potenciais elétrico e magnético nos vértices dos elementos da discretização e passam a ser integrais de linha dos campos ao longo das arestas destes elementos [8] [9]. O resultado final desta mudança de enfoque é um método numérico apropriado para a consideração de casos de aplicação em altas freqüências e que não sofre do problema das soluções espúrias mencionado.

Diante de tudo isso, para o estudo de caracterização dos sistemas de aterramento aqui realizado foi adotada então esta abordagem de elementos finitos de aresta. Utilizou-se para este fim o já mencionado programa HFG3D, desenvolvido no Laboratório de Eletromagnetismo Aplicado da Escola Politécnica da Universidade de São Paulo e que realiza uma variante do método dos elementos finitos de aresta dedicada a problemas de aterramento.

4. Pré-processamento, Processamento e Pós-processamento para o programa HFG3D

O programa HFG3D corresponde, na realidade, a um módulo de resolução numérico de problemas formulados segundo a técnica de elementos finitos vetorial. Trata-se de um programa que toma como informações de entrada uma geometria já discretizada em elementos finitos e uma condição de contorno, e a partir destes dados monta um sistema linear de equações representativo do problema que por sua vez é resolvido iterativamente pelo método ICCG (*Incomplete Cholesky Conjugate Gradients*). Por condição de contorno entende-se a imposição prévia de alguma condição física a alguns locais da geometria discretizada, sendo que isto se faz necessário para se levantar a indeterminação do sistema linear obtido que é inerente à técnica de elementos finitos. Condições de contorno típicas para problemas de aterramento correspondem à necessidade de o potencial elétrico ser nulo a grandes distâncias no interior do solo e à aproximação de se exigir campos eletromagnéticos sem componente normal no plano do solo, por exemplo.

Observa-se, então, que o HFG3D não possui incorporado entre as suas funcionalidades recursos sofisticados para a entrada e para a saída de dados, necessitando, portanto, de programas auxiliares para a realização destas tarefas. Afinal, conforme o já afirmado, para o seu uso exige-se que uma discretização em elementos finitos do problema considerado já tenha sido previamente produzida através de outros meios e que a massa de dados correspondente aos resultados calculados pelo programa seja de alguma forma apresentada de maneira amigável ao usuário. O desenvolvimento destes programas auxiliares corresponde à criação das ferramentas computacionais complementares mencionada entre os objetivos deste trabalho, de modo que a simulação numérica dos problemas de aterramento analisados neste trabalho foi realizada na realidade através do uso coordenado de uma série de programas.

Desta forma, para efeito de se obter um maior esclarecimento a respeito dos diversos programas que foram utilizados e também sobre as ferramentas computacionais adicionais que precisaram ser desenvolvidas, são descritas a seguir as três etapas básicas correspondentes à resolução numérica de cada problema de aterramento (pré-processamento, processamento e pós-processamento), conforme elas foram realizadas neste trabalho. Foram discriminadas as atividades envolvidas em cada uma destas etapas e se deu ênfase ao relacionamento entre os programas utilizados e ao fluxo de informações através de arquivos de dados. Esta dinâmica de troca de dados e de divisão de tarefas entre programas até que será na seqüência relatada foi ainda esquematicamente representada no diagrama da figura 2.

4.1. Pré-processamento

Por pré-processamento entende-se a geração do modelo geométrico do problema, a discretização deste modelo através de uma malha de elementos finitos e a especificação das condições de contorno necessárias para a resolução numérica do problema. O modelo geométrico e a discretização foram criados, respectivamente, com as ferramentas de CAD e com o algoritmo de geração de malhas de elementos hexaédricos pertencentes ao programa FLUX3D. Este programa por sua vez permite exportar estas informações representativas da geometria e da malha de elementos finitos associada para um arquivo texto escrito num formato específico, denominado PF3. O conteúdo de tal arquivo consiste, basicamente, numa listagem numerada dos pontos que compõem a malha de elementos finitos, nas coordenadas destes pontos e em uma numeração de todos os elementos hexaédricos resultantes, expressa em termos da numeração dos pontos que correspondem aos oito vértices de cada um deles.

A especificação das condições de contorno foi realizada através da geração de um arquivo de dados necessário ao programa HFG3D, denominado bc.dat. Tal arquivo foi construído com o programa MATLAB [10] a partir de um procedimento numérico contido num programa já disponível.

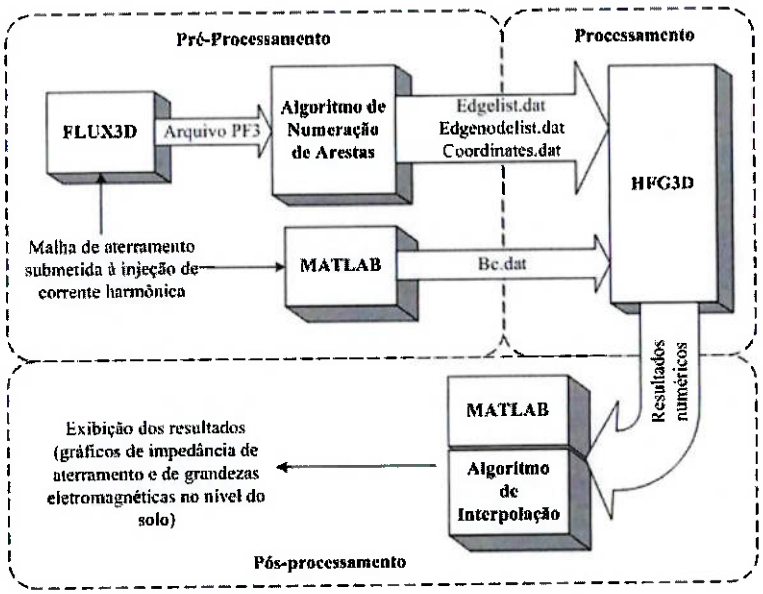

Figura 2.
Relacionamento entre os programas utilizados e fluxo de dados

4.2. Processamento

A etapa de processamento representa a resolução numérica do problema propriamente dita, sendo o responsável por ela o programa HFG3D conforme o já apontado. A tomada de dados para este programa é feita através de quatro arquivos de entrada. O primeiro deles, contendo condições de contorno, é o arquivo bc.dat já mencionado. O segundo destes arquivos (coordinates.dat) deve conter as coordenadas e a numeração de cada um dos pontos da malha de discretização a ser utilizada, e é facilmente obtido a partir do arquivo PF3 anteriormente gerado. Os outros dois arquivos restantes (edgelist.dat e edgenodelist.dat) devem apresentar informações das arestas componentes de cada elemento hexaédrico da discretização (como a sua numeração e também a numeração dos dois pontos da malha que determinam cada uma delas).

Estas informações de arestas contidas nestes arquivos são necessárias justamente por se estar utilizando uma formulação vetorial do método dos elementos finitos no HFG3D. A obtenção destes dois arquivos não é imediatamente realizada a partir da massa de dados disponível no arquivo PF3 anteriormente produzido, uma vez que o programa FLUX3D utiliza a formulação nodal do método dos elementos finitos conforme já se mencionou em passagem anterior. Tornou-se necessário então a criação de um programa intermediário para a realização de um algoritmo de numeração de arestas.

O algoritmo correspondente a esta tarefa de numerar arestas possui um papel bastante importante no trabalho desenvolvido, justamente porque executa a tarefa essencial de converter os dados de geometria e de discretização produzidos pelo FLUX3D para um formato adequado à formulação do método de elementos finitos de aresta implementada no

programa HFG3D. Maiores detalhes a respeito do programa que o realiza serão apresentados adiante.

4.3. Pós-processamento

Para os problemas estudados neste trabalho e de acordo com os objetivos estabelecidos, o pós-processamento corresponde à apresentação dos resultados de interesse para cada uma das configurações de aterramento selecionadas. Portanto, trata-se da tarefa de, a partir dos resultados calculados pelo programa HFG3D, produzir-se gráficos do comportamento da impedância de aterramento em função da freqüência e da distribuição dos campos e dos potenciais no nível do solo. Para esta finalidade, foram utilizadas ferramentas gráficas de visualização disponíveis no programa MATLAB.

O caso particular da representação gráfica de campos e potenciais no nível do solo exige o cálculo das grandezas consideradas em uma grande quantidade de pontos deste plano. Por este motivo, e de acordo com o que se apresentou na breve descrição da técnica de elementos finitos anteriormente realizada, houve a necessidade de se produzir um algoritmo de interpolação para o cálculo da grandeza em questão em pontos adicionais do domínio.

5. O programa de numeração de arestas

Conforme o já mencionado, a informação contida no arquivo PF3 gerado pelo FLUX3D representa, basicamente, uma lista dos pontos que compõem a malha de elementos finitos do problema, as suas coordenadas e uma numeração associada a estes nós e aos elementos hexáedricos resultantes. O programa que executa o algoritmo de numeração de arestas a partir dos dados deste arquivo (de acordo com a figura 2) foi inicialmente desenvolvido no ambiente do MATLAB, utilizando-se assim a linguagem de programação própria deste programa.

Preliminarmente, foi construído com o programa FLUX3D um problema teste, correspondendo a uma geometria pequena e simples. Mais especificamente, a geometria simplificada escolhida consistiu numa composição de oito elementos hexaédricos (representada na figura 3), e a partir do seu traçado foi obtido o arquivo de dados no formato PF3 representativo do problema e da malha de elementos finitos associada. O objetivo da criação de um caso teste foi justamente o de utilizá-lo durante esta etapa de desenvolvimento para a validação e para a verificação do programa que viria a ser produzido, uma vez que os arquivos PF3 gerados a partir de casos reais de aplicação são bastante extensos (no caso real de uma malha de aterramento, a quantidade de elementos hexaédricos resultante da discretização em elementos finitos facilmente chega às dezenas de milhares, ao invés dos apenas oito elementos do caso simplificado produzido). Assim, o desenvolvimento do programa realizado imediatamente a partir das geometrias de interesse resultaria inevitavelmente em dificuldades para a depuração do código e para a correção de problemas.

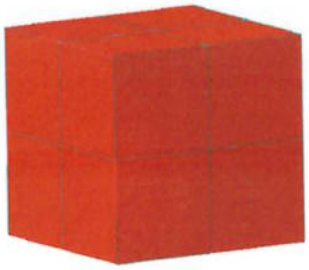

Figura 3.
Modelo geométrico para o caso teste

Obteve-se então uma primeira versão funcional do programa de numeração de arestas codificado com a linguagem de programação do MATLAB, e esta foi então aplicada ao caso teste para se obter os arquivos edgelist.dat, edgenodelist.dat e coordinates.dat já mencionados. Estes por sua vez foram obtidos com sucesso, porém a aplicação do mesmo programa aos casos reais de sistema de aterramento que se apresentarão adiante explicitou a existência de algumas dificuldades que somente neste momento puderam ser evidenciadas. Primeiramente, verificou-se que o tempo de processamento para a conversão do arquivo PF3 (que para o caso teste com apenas 8 elementos hexaédricos foi de apenas poucos instantes) era demasiadamente extenso já para o caso real e de interesse de uma haste verticalmente enterrada, a configuração de aterramento mais simples entre as duas que serão adiante consideradas neste trabalho. Para ela, na qual a quantidade total de elementos hexaédricos resultante da discretização adotada é de pouco mais do que 11000, o tempo total para a conversão alcançou a marca proibitiva de quase dois dias de processamento nos computadores

disponíveis no Laboratório de Eletromagnetismo Aplicado. Além disso, terminado este extenso período de processamento e finalmente construídos os três arquivos convertidos para a entrada de dados no HFG3D, verificou-se que estes continham alguns problemas construtivos que não se manifestaram durante os testes com o caso simplificado contendo apenas oito elementos.

Diante destes fatos, chegou-se a conclusão de que a insistência no uso do MATLAB para a realização do algoritmo de numeração de arestas seria infrutífera, pois os longos tempos de execução do programa verificados inviabilizariam o processo de depuração necessário à correção dos problemas construtivos que se encontraram nos arquivos obtidos para as geometrias reais de aterramento selecionadas. Optou-se então pela reescrita do programa de numeração de arestas utilizando-se agora ao invés do MATLAB a linguagem de programação C [11]. Esta simples transposição do código redigido para a linguagem C resultou na redução do tempo total de execução do algoritmo de dois dias para algumas dezenas de minutos. Isto se verificou pelo fato de a linguagem C fazer o uso de compiladores, ao contrário da linguagem de programação do MATLAB que emprega comandos interpretados que resultam em programas de execução intrinsecamente mais demorada.

Desta forma, foi criada uma segunda versão do programa de numeração de arestas, desta vez produzida na linguagem C. Para esta finalidade foi empregado o compilador Bloodshed Dev-C++ 4.9.9.2 [12], distribuído como software livre pela Internet nos termos da *GNU General Public License*. Através dele, foi possível realizar a depuração desta nova versão do programa em tempos aceitáveis e os problemas construtivos que afetavam os arquivos criados para as geometrias de aterramento de interesse puderam então ser detectados e resolvidos. O código fonte do programa de numeração de arestas assim desenvolvido encontra-se integralmente reproduzido no Apêndice A.

Tendo em vista então tudo o que até aqui foi exposto, a seguir encontra-se descrito o funcionamento do programa de numeração de arestas criado, sintetizado em alguns itens que correspondem aos trabalhos em seqüência realizados durante a execução do código. Ao longo desta descrição, o leitor será remetido para exemplos em anexo dos arquivos de dados envolvidos, todos eles correspondentes ao caso teste de oito hexaedros já apresentado. A extensão dos mesmos arquivos referentes aos problemas de aterramento reais adiante considerados é tal que impede a reprodução destes neste trabalho.

1) Leitura do arquivo PF3. O programa toma o arquivo PF3 com as informações geométricas da malha e armazena o seu conteúdo. A estrutura do arquivo PF3 é tal que ele possui duas seções principais. A primeira descreve a topologia dos elementos da malha de elementos finitos, sendo que nela cada duas linhas correspondem a um elemento. O primeiro número da primeira linha de certo elemento representa a numeração deste, criada pelo algoritmo de geração de malhas do FLUX3D. O segundo número da primeira linha codifica o tipo do elemento, sendo que sempre vale sete para os elementos hexaédricos. Os oito números da segunda linha de certo elemento armazenam a numeração global dos nós que integram este elemento, que também foi criada pelo algoritmo gerador de malhas do FLUX3D. À seqüência com que estes oito nós se apresentam nesta linha está associada uma numeração local dos nós do elemento. A segunda seção do arquivo PF3 armazena uma listagem das coordenadas espaciais cartesianas dos nós da malha de elementos finitos representada, sendo que estes são referenciados através da numeração global de nós já mencionada. O Apêndice B contém o arquivo PF3 correspondente ao caso teste.

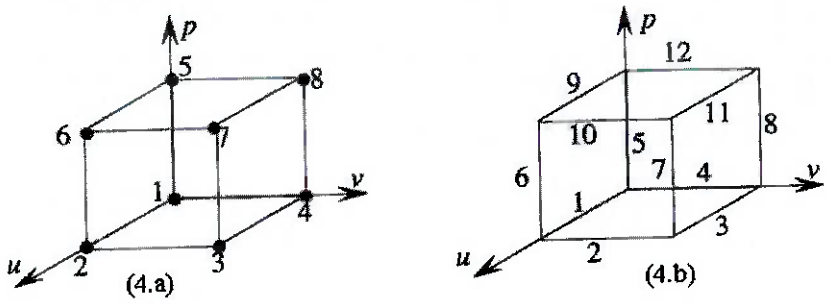

Figura 4.
Obtenção da numeração local de arestas a partir da numeração local de nós num elemento hexaédrico

2) Numeração das arestas. É produzida uma numeração das arestas constituintes da malha de elementos finitos a partir da numeração dos seus nós, da numeração dos elementos e das coordenadas dos nós disponíveis no arquivo PF3. Enfatiza-se aqui mais uma vez que esta numeração adicional das arestas não realizada pelo FLUX3D é necessária ao HFG3D devido às peculiaridades na formulação do método numérico empregado por este último, que é o método dos elementos finitos de aresta. A referida numeração de arestas foi realizada de acordo com o procedimento proposto por Jin [9], e pode ser mais bem compreendida a partir da figura 4 [8]. Para cada elemento hexaédrico da malha, a numeração local de vértices (nós) já mencionada anteriormente é feita de acordo com a figura 4.a. dentro do arquivo PF3. Através de rotinas de busca e de ordenação, o programa desenvolvido produz uma numeração local também para as arestas (até então inexistente indisponível no arquivo PF3) seguindo o padrão contido na figura 4.b. A partir então desta numeração local de arestas realizada internamente a cada elemento, cria-se também uma numeração global para todas as arestas da malha de elementos finitos que será utilizada nos arquivos de saída gerados e descritos a seguir.

3) Geração dos arquivos de saída. O programa armazena os resultados obtidos nos três arquivos de saída já mencionados, denominados edgelist.dat, edgenodelist.dat e coordinates.dat. Estes apresentam o formato adequado para a entrada de dados no HFG3D. O arquivo edgelist.dat criado contém uma linha para cada uma das arestas da malha de elementos finitos, sendo que cada linha por sua vez armazena, em seqüência, a numeração gerada na etapa anterior para a aresta correspondente e a numeração dos dois nós formadores desta aresta (de acordo com a numeração global de nós feita no arquivo PF3). O arquivo edgenodelist.dat possui uma linha para cada elemento hexaédrico, cada uma delas com 21 números. O primeiro número é a numeração do elemento corrente extraída do arquivo PF3. Os oito números seguintes representam as numerações dos oito nós correspondentes aos oito vértices do presente elemento hexaédrico também de acordo com a numeração global disponível no arquivo PF3. Os doze números finais representam as numerações das arestas constituintes do elemento corrente, obtidas na etapa anterior e também armazenada no arquivo edgelist.dat. O arquivo coordinates.dat é simplesmente uma transcrição da segunda seção do arquivo PF3, mencionada no primeiro item. Os Apêndices C, D e E contêm, respectivamente, reproduções dos arquivos edgelist.dat, edgenodelist.dat e coordinates.dat gerados pelo programa de numeração de arestas para o caso teste.

6. O Programa de Interpolação

Uma vez desenvolvida esta primeira ferramenta computacional complementar necessária à entrada de dados no HFG3D, este programa poderia ser utilizado para o estudo dos sistemas de aterramento proposto entre os objetivos pretendidos tão logo fossem construídos com o FLUX3D os modelos geométricos e as discretizações em elementos finitos correspondentes às configurações de aterramento selecionadas para a análise. Bastaria tomar-se o arquivo PF3 associado e se construir, a partir dele, os arquivos de entrada já descritos através da execução do programa de numeração de arestas desenvolvido. No ato da sua execução, o programa HFG3D solicita como entradas adicionais as características do solo (resistividade, permeabilidade magnética e permissividade elétrica), a intensidade e a freqüência do fasor de corrente harmônica a ser injetado no aterramento e também a coordenada do ponto no plano do solo em que se dará a injeção desta corrente. Faltaria ainda, porém, realizar a apresentação dos resultados produzidos pelo HFG3D no padrão e no formato desejado e proposto entre os objetivos. Trata-se das atividades de pós-processamento anteriormente mencionadas.

O Apêndice F apresenta um exemplo do formato de saída do programa HFG3D, que escreve um arquivo texto com os resultados por ele calculados através da técnica de elementos finitos vetorial. A consulta a este apêndice mostra ao leitor que o programa calcula a impedância complexa equivalente do aterramento verificada no ato da injeção do harmônico de corrente selecionado, além de fornecer uma extensa listagem de números, dispostos em muitas linhas e em duas colunas. Cada linha está associada a um dos nós da discretização em elementos finitos adotada, sendo que a primeira coluna armazena a parte real em volts do fasor do potencial elétrico calculado para o nó associado a esta linha (resultante da injeção de corrente realizada). A segunda coluna, por sua vez, armazena a parte imaginária em volts do mesmo.

A análise do arquivo de saída do programa HFG3D em anexo deixa clara então a necessidade de se construir ferramentas complementares para a exibição dos resultados de forma gráfica e amigável. Os gráficos desejados de módulo e fase da impedância complexa de aterramento em função da freqüência puderam ser obtidos através do uso de recursos simples do MATLAB, tendo sido apenas necessário a realização de muitas simulações (com o objetivo de se varrer um amplo intervalo de freqüências). Adotou-se para o limite superior de freqüência considerado neste trabalho o teto de 1MHz, pelo fato de freqüências menores ou iguais a este valor já serem capazes de modelar adequadamente um impulso atmosférico e também por inexistirem na literatura resultados para comparação em freqüências ainda mais elevadas.

A tarefa que também figura entre os objetivos deste trabalho de se representar graficamente o comportamento de grandezas eletromagnéticas no plano do solo apresentou, no entanto, alguns complicadores. Conforme o Apêndice F, observa-se que o programa HFG3D fornece valores calculados de potenciais elétricos em diversos pontos do domínio estudado, sendo então obrigatório que se realize a partir desta grandeza particular a obtenção das outras em que eventualmente se tenha interesse, como por exemplo o campo elétrico associado. Além disso, e de acordo com o que já se afirmou em passagem anterior, a representação gráfica de grandezas no plano do solo exige o cálculo destas em uma grande quantidade de pontos, bastante superior àquela resultante dos pontos da malha de elementos finitos adotada que coincidentemente já se situem sobre o plano do solo (e para os quais o programa HFG3D fornece o potencial elétrico calculado explicitamente). Fez-se necessária então a criação de uma segunda ferramenta computacional complementar, desta vez para o cálculo do campo elétrico associado ao potencial elétrico já disponível e também para a

interpolação de novos valores calculados para o plano do solo entre as grandezas mencionadas.

O programa de interpolação em questão também foi escrito na linguagem de programação C, utilizando-se mais uma vez o compilador Bloodshed Dev-C++ 4.9.9.2. A seguir, são descritas as principais etapas envolvidas na realização da interpolação por ele executada.

1) Representação matemática do plano do solo. O nível do solo nas imediações do sistema de aterramento é representado matricialmente. São definidos pontos regularmente espaçados no plano do solo e imagina-se que cada posição de uma matriz quadrada armazena a abscissa e que cada posição de uma segunda matriz quadrada armazena a ordenada de cada um destes pontos. Ambas estas coordenadas são dadas em relação a um sistema cartesiano centralizado sobre a região do plano do solo representada.

2) Posicionamento dos pontos do plano do solo na malha de discretização e busca pelo elemento hexaédrico a que pertencem. Através de um procedimento de busca, determina-se a qual dos elementos hexaédricos da malha de discretização adjacentes à superfície do solo pertence cada um dos pontos armazenados nas referidas matrizes.

3) Interpolação do potencial elétrico e cálculo do campo elétrico associado. Uma vez localizado o elemento ao qual certo ponto pertence, o fasor do potencial elétrico para este ponto é determinado através de interpolação a partir dos fasores de potencial elétrico calculados pelo HFG3D para os quatro vértices deste elemento situados na face do hexaedro tangente ao plano do solo. Esta interpolação é realizada então de acordo com o proposto por Jin [9] e através de expressões matemáticas bem conhecidas na literatura referente às técnicas de elementos finitos. Através de expressões igualmente conhecidas, determina-se também o campo elétrico correspondente.

Ao término da execução deste programa, o potencial elétrico e o campo elétrico encontram-se determinados para cada um dos pontos representados matricialmente na etapa inicial do programa. O programa de interpolação foi concebido de tal forma que ele é capaz de determinar o potencial e o campo elétrico para 1681 pontos regularmente distribuídos ao longo de uma superfície quadrada de 6400m^2 sobre a superfície do solo acima do sistema de aterramento.

A saída do programa é escrita de tal maneira que os resultados obtidos são armazenados em arquivos de formato já apropriado para serem utilizados pelo MATLAB na produção dos gráficos desejados (arquivos do tipo "m-file"). O código fonte correspondente a este algoritmo de interpolação também se encontra integralmente reproduzido no Apêndice G. Sem muita dificuldade, a análise deste código e dos comentários feitos junto dele permite identificar as expressões de interpolação empregadas e que aqui foram omitidas.

7. Casos de Aplicação

O desenvolvimento dos programas complementares descritos, juntamente com o uso coordenado destes com as demais ferramentas de software já citadas, permitiu então que fosse possível se realizar o estudo de configurações de sistemas de aterramento apresentado a seguir com o método dos elementos finitos de aresta. Os casos de aplicação selecionados foram, de acordo com o apontado entre os objetivos do trabalho, o de um aterramento através de uma haste metálica verticalmente enterrada e o de uma malha de aterramento.

Antes de se apresentar cada um destes casos de aplicação em todo o seu detalhe, alguns comentários gerais a respeito da obtenção dos modelos geométricos dos sistemas de aterramento selecionados se fazem necessários. Em muitos problemas de eletromagnetismo, como por exemplo este aqui considerado da injeção de corrente no solo, os campos eletromagnéticos estabelecidos estendem-se ao longo de distâncias que virtualmente tendem ao infinito. Como a representação geométrica destes problemas num computador para a resolução numérica obviamente não pode se prolongar infinitamente pelo simples fato de os recursos computacionais serem limitados, surge então a necessidade de se truncar o domínio em estudo. Tal truncamento sem dúvida nenhuma deve ser realizado com algum critério, pois ele por sua vez não deve comprometer os resultados calculados pela técnica numérica empregada.

O critério aqui adotado para o truncamento do domínio e conseqüentemente para o dimensionamento da região em torno do sistema de aterramento a ser representada geometricamente foi o seguinte. A profundidade de penetração δ de um campo eletromagnético estabelecido num certo meio com resistividade ρ, permeabilidade magnética μ_o e que pulsa com uma freqüência f pode ser calculada através da expressão

$$\delta = \sqrt{\frac{\rho}{\pi f \mu_o}}$$

A aplicação desta expressão aos problemas aqui estudados leva à conclusão de que injeções de corrente em baixa freqüência resultam em campos eletromagnéticos com elevada profundidade de penetração, ou seja, que se atenuam significativamente somente após terem se espalhado por grandes distâncias. Injeções de corrente em alta freqüência por sua vez atenuam-se em curtas distâncias. Assim, o cálculo da mínima e da máxima profundidade de penetração do campo a partir, respectivamente, da máxima e da mínima freqüência das correntes injetadas no aterramento permite que se obtenha parâmetros de distância úteis no dimensionamento do domínio geométrico em torno do sistema de aterramento que precisa ser integralmente representado. A máxima distância de penetração calculada (correspondente à mínima freqüência) fornece uma medida para a máxima dimensão do domínio geométrico a ser representado em torno do sistema de aterramento. A mínima distância de penetração calculada (correspondente à máxima freqüência), por sua vez, fornece uma medida da distância em torno do sistema de aterramento dentro da qual a discretização em elementos finitos realizada precisa ser mais refinada.

Os modelos geométricos para os sistemas de aterramento aqui estudados e a discretização em elementos finitos foram portanto construídos levando-se em conta estas observações. Conforme se afirmou também anteriormente, utilizou-se para esta finalidade o programa FLUX3D. Finalmente, nas próximas subseções cada um dos dois casos de aplicação é apresentado, juntamente com os resultados obtidos e com a comparação destes com a literatura.

7.1 Primeiro caso de aplicação – Haste metálica de aterramento

No trabalho desenvolvido por Bourg, Sacepe e Debu [3], é apresentada uma modelagem matemática para o problema de uma haste metálica em solo homogêneo. Neste modelo a haste é tratada como uma linha de transmissão, e o equacionamento resultante para o comportamento da impedância complexa de aterramento para uma ampla faixa de freqüências de injeção de corrente é confrontado com medições experimentais.

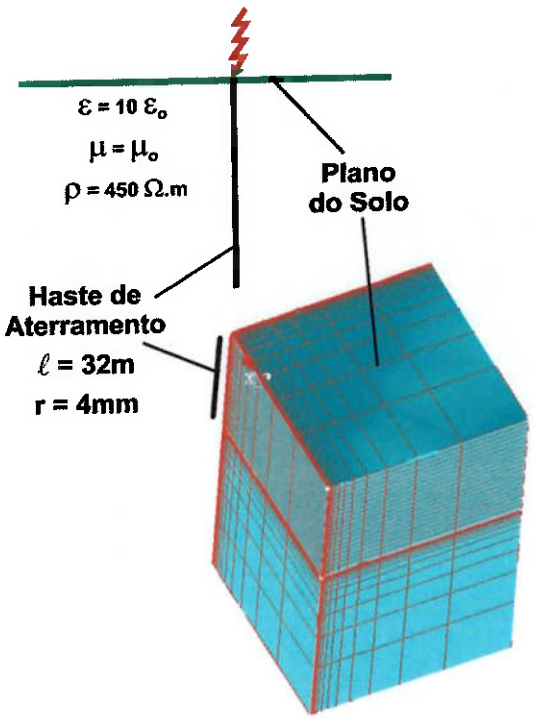

Figura 5.
O aterramento através de haste considerado e a discretização em elementos finitos hexaédricos correspondente

A figura 5 representa um dos casos de aterramento através de haste metálica apresentado em [3], no qual o solo foi admitido homogêneo, com permissividade elétrica $\varepsilon = 10.\varepsilon_o$, permeabilidade magnética μ_o e resistividade $\rho = 450\Omega.m$. A haste foi tomada como possuindo 32m de comprimento, possuindo 4mm de raio e sendo perfeitamente condutora. Este mesmo problema foi então analisado com o HFG3D e com as ferramentas auxiliares já mencionadas.

De acordo então com tudo o que até aqui foi exposto, a discretização da geometria correspondente ao problema da haste foi feita com o programa FLUX3D (figura 5). Dada a simetria do problema, tornou-se suficiente que apenas um quarto da região em torno da haste fosse representada. O cálculo da impedância de aterramento foi feito com funcionalidades do programa HFG3D empregando o método dos elementos finitos de aresta.

A apresentação dos resultados obtidos em várias freqüências foi feita com o MATLAB de acordo com o gráfico da figura 6. Nele foram também simultaneamente representados os resultados experimentais e o modelo teórico de [3]. O registro experimental foi extraído diretamente da referida publicação, e infelizmente não se encontra disponível com maior resolução ou com a qualidade que se desejaria. Mesmo assim, o confronto de todos estes valores graficamente apresentados permite concluir então que há um bom acordo entre a

técnica de elementos finitos de aresta e os resultados experimentais e teóricos previstos segundo outras metodologias.

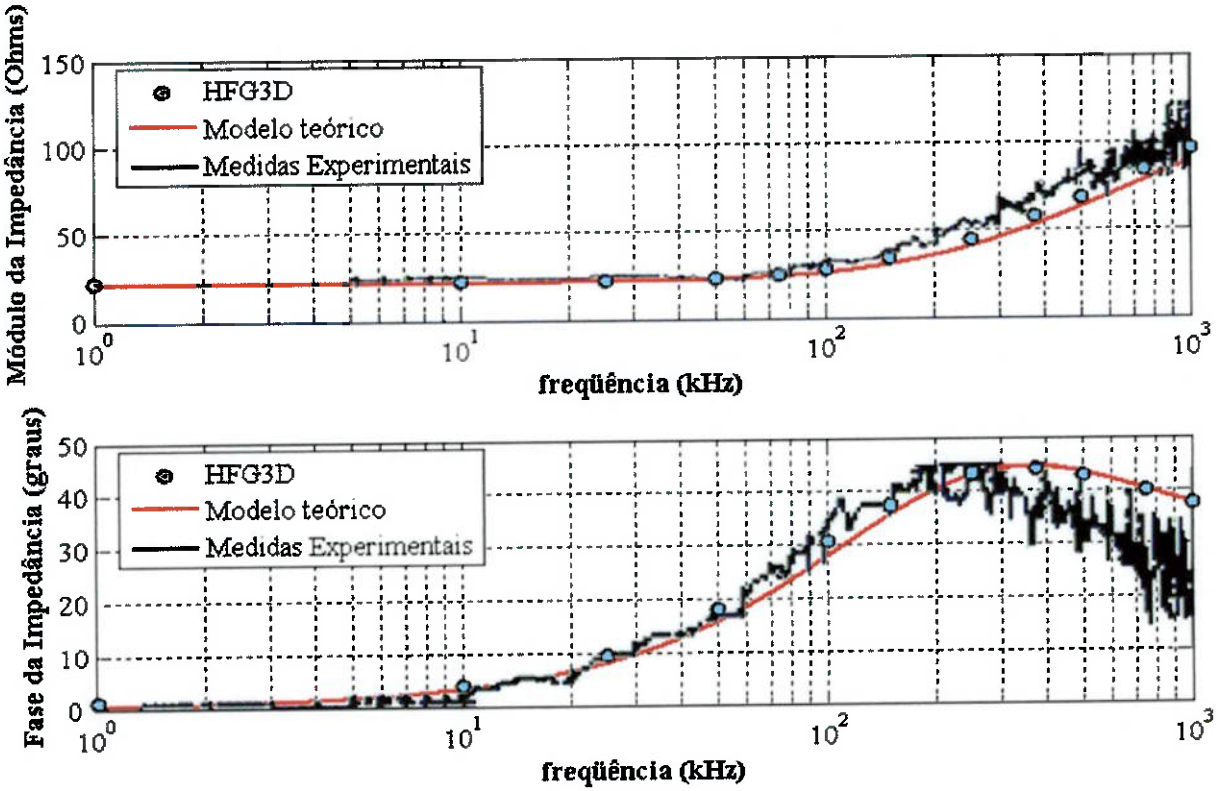

Figura 6.
Impedância de aterramento para a haste metálica considerada calculada com o programa HFG3D e comparação com modelo teórico e medidas experimentais

7.2 Segundo caso de aplicação – Malha de aterramento

Condutores metálicos conectados entre si e compondo uma grade quadriculada representam uma configuração comum de sistema de aterramento em subestações elétricas e em instalações industriais. Dada a geometria complexa destes aterramentos em malha, o uso de métodos numéricos corresponde à principal ferramenta para a previsão do comportamento de tais sistemas.

Grcev obteve em seus trabalhos descrições da variação com a freqüência da impedância de aterramento para uma grande variedade de geometrias de aterramento em malha submetidas a diferentes condições de solo [4]. Xiong, por sua vez, determinou o comportamento no nível do solo de diversas grandezas eletromagnéticas, representando-as em gráficos tridimensionais [5]. Ambos os autores utilizaram em seus respectivos trabalhos abordagens baseadas na teoria eletromagnética, mas com métodos numéricos diferentes do método dos elementos finitos.

Os dois autores consideraram também nos trabalhos mencionados o sistema de aterramento particular detalhado a seguir e representado na figura 7. Este foi então aqui utilizado como um segundo caso de aplicação. Trata-se de uma malha de aterramento quadriculada de 60m de comprimento e 60m de largura, com as subdivisões realizadas a cada

10m. Os condutores são de cobre e com diâmetro de 1,4cm. Este reticulado foi então enterrado a 0,5m de profundidade em um solo homogêneo com permissividade elétrica $\varepsilon = 9.\varepsilon_o$, permeabilidade magnética μ_o e resistividade $\rho = 1000\Omega.m$. Fixando-se um sistema de coordenadas cartesiano no plano do solo com origem num ponto centralizado sobre a malha de aterramento, as injeções de correntes harmônicas necessárias às simulações realizadas se fizeram no ponto de coordenadas x = 30m e y = 30m, localizado assim sobre um dos cantos da malha.

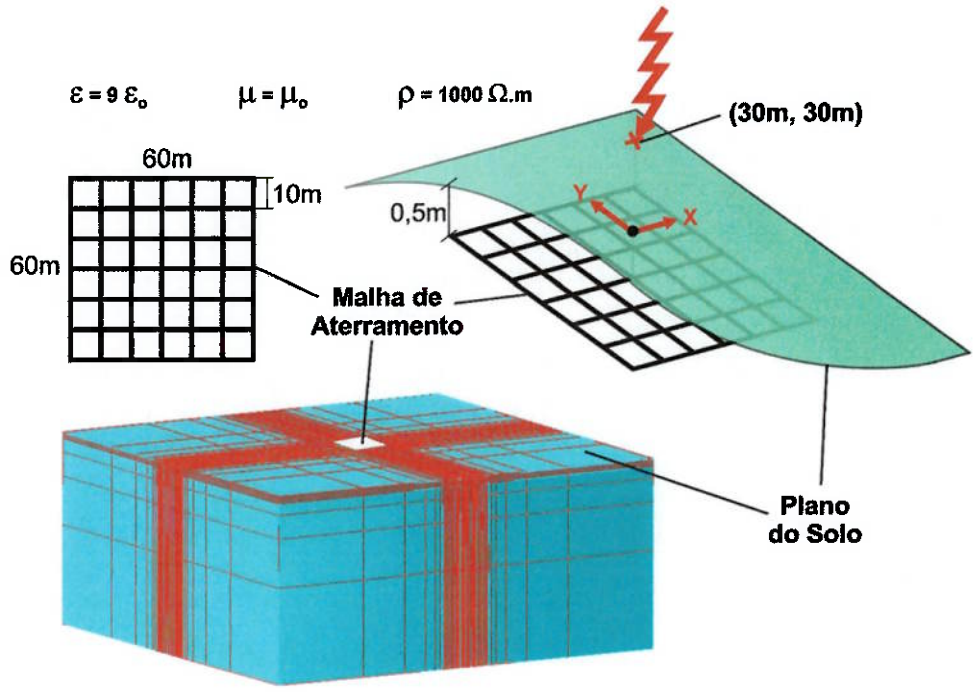

Figura 7.
A malha de aterramento considerada e a discretização em elementos finitos hexaédricos correspondente

A figura 7 contém também a representação da geometria deste problema feita com o programa FLUX3D e a correspondente discretização em elementos finitos hexaédricos. O gráfico da figura 8 compara os valores de impedância de aterramento obtidos com o programa HFG3D através da metodologia exposta neste trabalho com os resultados obtidos por Grcev com o seu método numérico alternativo [4].

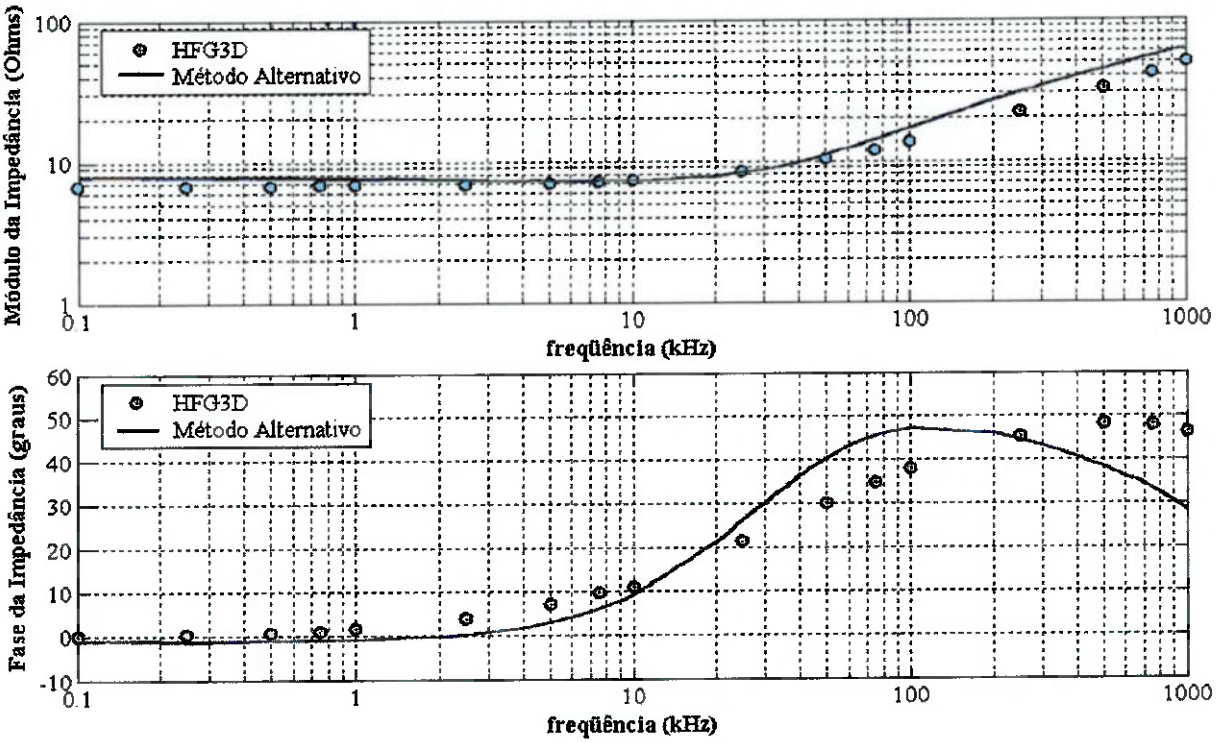

Figura 8.
Impedância de aterramento para a malha considerada calculada com o programa HFG3D e comparação com método numérico alternativo

A análise do gráfico da figura 8 mostra um relativo acordo entre a técnica de elementos finitos de aresta e o método numérico empregado em [4]. Embora os valores de módulo da impedância determinados por ambas as metodologias apresentem uma boa correlação para qualquer uma das freqüências consideradas, os valores de fase possuem comportamentos diferentes nas freqüências mais elevadas (superiores a 10 kHz). Apesar desta parcial divergência, conclusões a respeito da maior adequação de uma metodologia ou de outra não podem ser tiradas em função da indisponibilidade na literatura de medições experimentais confiáveis para configurações de aterramento semelhantes.

As figuras 9 e 10 contêm representações da magnitude do potencial elétrico e do módulo do campo elétrico no plano do solo decorrentes da injeção de uma corrente de (1000 + j.0) A na freqüência de 0,5MHz no referido canto da malha de aterramento. Estes gráficos foram determinados a partir dos valores de potencial elétrico calculados pelo programa HFG3D e com a aplicação do programa de interpolação anteriormente apresentado. Figuras nestes mesmos moldes encontram-se publicadas no trabalho de Xiong citado anteriormente, e a comparação dos resultados aqui obtidos com aqueles lá disponíveis mostra um acordo entre o aspecto das superfícies obtidas e entre as ordens de grandeza dos valores representados. Uma comparação deste gênero é feita na figura 11.

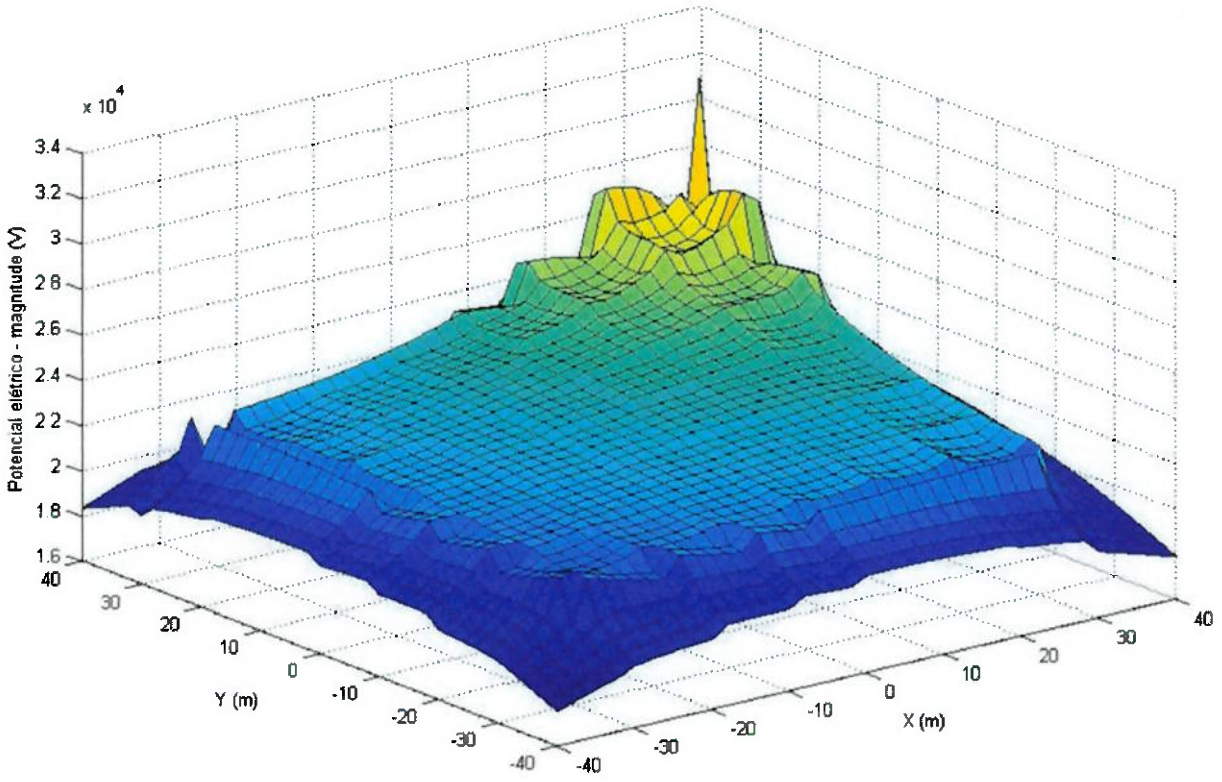

Figura 9.
Magnitude do potencial elétrico no plano do solo para uma injeção de corrente de 1000A em 0,5MHz no canto da malha de aterramento

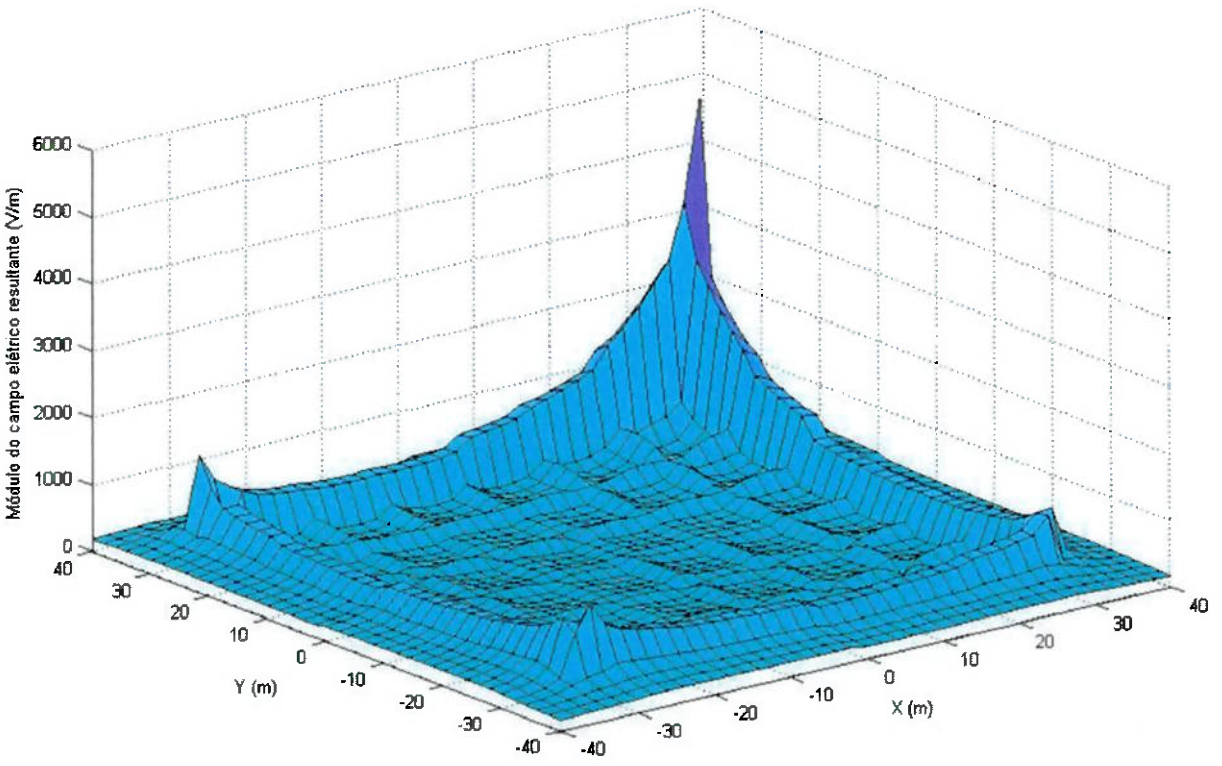

Figura 10.
Módulo do campo elétrico resultante no plano do solo para uma injeção de corrente de 1000A em 0,5MHz no canto da malha de aterramento

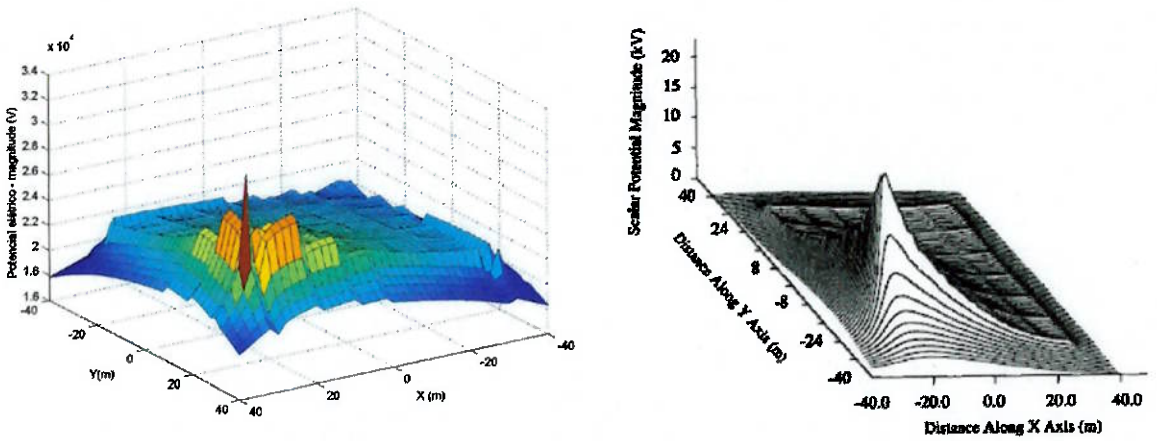

Figura 11.
Comparação do potencial elétrico determinado no plano do solo para uma injeção de corrente de 1000A em 0,5MHz no canto da malha com um resultado similar disponível na literatura (figura da direita) [5]

A figura 12 dada a seguir é um mapa de cores da magnitude da elevação de potencial nas imediações do sistema de aterramento. Corresponde a uma representação bidimensional do gráfico tridimensional da figura 9.

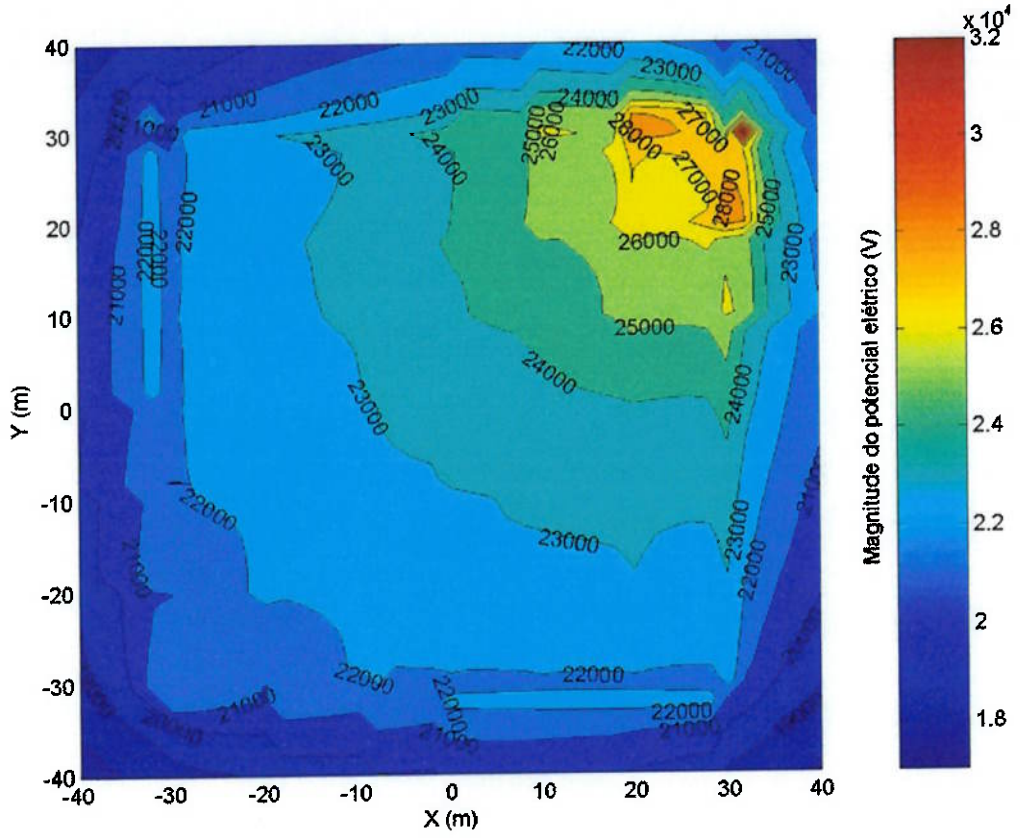

Figura 11.
Mapa de cores para a magnitude do potencial elétrico no plano do solo para uma injeção de corrente de 1000A em 0,5MHz no canto da malha de aterramento

8. Conclusão

Com a realização do trabalho aqui apresentado, foi desenvolvido um conjunto de ferramentas computacionais auxiliares para o programa HFG3D, tanto para a entrada de dados quanto para a apresentação de resultados. A utilidade e a consistência de tais ferramentas foram testadas e verificadas mediante a utilização destas nos dois problemas de aplicação propostos (o caso da haste de aterramento e o caso da malha).

A comparação dos resultados obtidos pela metodologia aqui proposta com outros disponíveis na literatura e o bom acordo em geral verificado entre estes permitiram não apenas a validação dos programas auxiliares desenvolvidos, mas também a validação da própria técnica de elementos finitos de aresta quando aplicada a problemas de alta freqüência. Demonstrou-se especialmente a aplicabilidade deste procedimento numérico ao estudo de sistemas de aterramento solicitados por injeções de corrente em alta freqüência, situação esta que conforme o exposto pode ser utilizada na modelagem da incidência de surtos atmosféricos sobre um sistema de aterramento.

9. Referências Bibliográficas

[1] MELIOPOULOS, A. P. S., Power System Grounding and Transients: an Introduction. N. York and Basel, Marcel Dekker Inc., 1988.

[2] KINDERMANN, G.; CAMPAGNOLO, J. M., Aterramento Elétrico. P. Alegre, Sagra–DC Luzzatto, 1995.

[3] BOURG, S.; SACEPE, B.; DEBU. T., *Deep Earth Eectrodes in Highly Resistive Ground: Frequency Behaviour*, IEEE International Symposium on Electromagnetic Compatibility, 1995, 14-18 August , p. 584-589.

[4] GRCEV, L. D.; HEIMBACH, M., *Frequency Dependent and Transient Characteristics of Substation Grounding System"*. IEEE Transactions on Power Delivery, v.12, n.1, p. 172-178, 1997.

[5] XIONG, W.; DAWALIBI, F. P., *Transient Performance of Substation Grounding Systems Subjected to Lightning and Similar Surge Currents*. IEEE Transactions on Power Delivery, v.9, n.3, p. 1412-420, 1994.

[6] CARDOSO, J. R., Introdução ao Método dos Elementos Finitos para Engenheiros Eletricistas.

[7] FLUX3D – CAD package for electromagnetic field computation, User's guide, CEDRAT.

[8] IDA, N.; BASTOS, J. P. A., Electromagnetics and Calculation of Fields. N. York, Springer-Verlag, 1997.

[9] JIN, J., The finite element method in electromagnetics. Wiley, Segunda edição, 2002.

[10] MATLAB Versão 7.0, Guia do Usuário, Mathworks.

[11] KERNIGHAN, B. W.; RITCHIE, D. M., C: A Linguagem de Programação. Editora Campus, 1986.

[12] Bloodshed Dev-C++ 4.9.9.2 – Compilador C e C++, http://www.bloodshed.net

APÊNDICE A - Código fonte do programa de numeração de arestas

```
#include <stdio.h>


#define MAXLINHA 100
#define MAXCOLUNA 12
#define NOMBREDELEMENTS 18
#define NOMBREDEMACROELEMENTS 24
#define DTOPOLOGIEDESELEMENTS 37
#define COORDONNEESDESNOEUDS 22


int main()
{

   /* abertura do arquivo PF3 de entrada */
   FILE *fp;
   fp = fopen("geometriateste.PF3", "r");


   /* abertura do arquivo coordinates.dat para saída */
   FILE *fp2; //apagar isto depois
   fp2 = fopen("coordinates.dat", "w");

   /* abertura do arquivo edgelist.dat para saída */
   FILE *fp3;
   fp3 = fopen("edgelist.dat", "w");

   /* abertura do arquivo edgenodelist.dat para saída */
   FILE *fp4; //apagar isto depois
   fp4 = fopen("edgenodelist.dat", "w");

    /*Saída para Debug*/
   FILE *fp5; //apagar isto depois
   fp5 = fopen("ids.txt", "w");

   /*Saída para Debug*/
   FILE *fp6; //apagar isto depois
   fp6 = fopen("orden.txt", "w");

   /*Saída para Debug*/
   FILE *fp7; //apagar isto depois
   fp7 = fopen("elig.txt", "w");

   /*Saída para verificação*/
   FILE *fp8; //apagar isto depois
   fp8 = fopen("verifica.txt", "w");
```

```
int nelementos; // número de elementos hexaédricos
int nnos; // número de nós da malha
int narestas; // número de arestas da malha
int i,j,k,l,m; // contadores
char s[MAXLINHA]; // armazena caracteres nas buscas por strings
int teste; // variável de teste
int ultimoid; // variável utilizada na ordenação das arestas
int temp[12]; // armazenam temporariamente valores durante ordenação
int *topologia; // armazena a descrição da topologia dos elementos extraída do PF3
float *coordenadas; //armazena as coordenadas dos nós extraídas do PF3
int *edgelist; // armazena a lista de arestas numeradas geradas com o algoritmo do Jin
int *arestas; // utilizada em etapa intermediária da escrita de edgelist
int *edgenodelist; // armazena nós e arestas de cada elemento


/*************************************************************************
 *    Leitura do arquivo PF3 e armazenamento do seu conteúdo             *
 *************************************************************************/


/* Busca pelo string "NOMBRE  D'ELEMENTS" */
teste=1;
for(i=0; teste!=0; i++)
{

      /* o tamanho máximo de "s" é o do string "NOMBRE  D'ELEMENTS": apos */
      if(i==NOMBREDELEMENTS)
      {
            for(j = 0; j<=NOMBREDELEMENTS-2; j++)
            {
                s[j]=s[j+1];
            }

            i--;
      }


      s[i] = getc(fp);
      s[i+1] = '\0';
      teste = strcmp("NOMBRE  D'ELEMENTS",s);
}


/* Cópia do próximo número após "NOMBRE D'ELEMENTS". Trata-se do número de
elementos volumétricos da malha.*/
fscanf(fp, "%d", &nelementos);


/* Busca pelo string "NOMBRE DE MACRO-ELEMENTS" */
teste=1;
```

```
for(i=0; teste!=0; i++)
{

        if(i==NOMBREDEMACROELEMENTS)
        {
                for(j = 0; j<=NOMBREDEMACROELEMENTS-2; j++)
                {
                    s[j]=s[j+1];
                }

                i--;
        }


        s[i] = getc(fp);
        s[i+1] = '\0';
        teste = strcmp("NOMBRE DE MACRO-ELEMENTS",s);
}
```

/* Cópia do próximo número após "NOMBRE DE MACRO-ELEMENTS". Trata-se do número de nós da malha.*/

```
fscanf(fp, "%d", &nnos);
```

/* Busca pelo string "DESCRIPTEUR DE TOPOLOGIE DES ELEMENTS" */

```
teste=1;
for(i=0; teste!=0; i++)
{

        if(i==DTOPOLOGIEDESELEMENTS)
        {
                for(j = 0; j<=DTOPOLOGIEDESELEMENTS-2; j++)
                {
                    s[j]=s[j+1];
                }

                i--;
        }


        s[i] = getc(fp);
        s[i+1] = '\0';
        teste = strcmp("DESCRIPTEUR DE TOPOLOGIE DES ELEMENTS",s);
}
```

/* Cópia do trecho do arquivo pf3 abaixo de "DESCRIPTEUR DE TOPOLOGIE DES ELEMENTS" */

// "topologia" será utilizado como uma matriz com 20 colunas e 'nelementos' linhas

```
// as 12 primeiras colunas armazenam a 1a. linha do elemento hexaédrico do PF3
// as 8 últimas colunas armazenam a 2a. linha (nós numerados glabalmente) do elemento

topologia = (int*)malloc(sizeof(int)*2*nelementos*20);

for(i=0; i<=((12+8)*nelementos)-1; i++)
{
      fscanf(fp, "%d", &topologia[i]);
}


/* Busca pelo string "COORDONNEES DES NOEUDS" */
teste=1;
for(i=0; teste!=0; i++)
{

      if(i==COORDONNEESDESNOEUDS)
      {
            for(j = 0; j<=COORDONNEESDESNOEUDS-2; j++)
            {
                s[j]=s[j+1];
            }

            i--;
      }


      s[i] = getc(fp);
      s[i+1] = '\0';
      teste = strcmp("COORDONNEES DES NOEUDS",s);
}


/* Cópia do trecho do arquivo pf3 abaixo de "COORDONNEES DES NOEUDS" */

// "coordenadas" será utilizado como uma matriz com 4 colunas e 'nnos' linhas
// a 1a. coluna armazenam a numeracão global do nó contida no arquivo PF3
// as 3 últimas colunas armaznam as coordenadas do nó correspondente

coordenadas = (float*)malloc(sizeof(float)*nnos*4);

for(i=0; i<=(4*nnos)-1; i++)
{
      fscanf(fp, "%f", &coordenadas[i]);
}



printf("Leitura e armazenamento do arquivo concluida...\n");
```

```
/*************************************************************************
*     Implementação do algoritmo de numeração de arestas do Jin          *
*************************************************************************/

/*1a. parte do algoritmo para geração da numeração global de arestas - Livro Jin*/

// "arestas" será uma matriz com 7 colunas e 12*nelementos colunas
// cada elemento (linha) de "topologia" gera 12 linhas em "arestas", seguindo a lógica de ordenação de arestas necessária
// é gerado um 'identificador' de arestas armazenado na primeira coluna de "arestas"
// as 4 últimas colunas de "arestas" armazenarão os elementos de que a aresta faz parte (3a. parte do algoritmo)

arestas = (int*)malloc(sizeof(int)*nelementos*7*12);


/* zera todo o conteúdo de "arestas" */
for(i=0; i<=(7*12*nelementos)-1; i++)
{
      arestas[i]=0;
}

i=0;
j=0;
while(topologia[20*i + 1 ] == 7)// verifica que se trata de elemento hexaédrico
{
      // 1a. aresta
      arestas[7*j + 0] = topologia[20*i + 12]*topologia[20*i + 13];
      arestas[7*j + 3] = topologia[20*i + 0];
      if( topologia[20*i + 12]<=topologia[20*i + 13] )
      {
        arestas[7*j + 1] = topologia[20*i + 12];
        arestas[7*j + 2] = topologia[20*i + 13];
      }
      else
      {
        arestas[7*j + 1] = topologia[20*i + 13];
        arestas[7*j + 2] = topologia[20*i + 12];
      }

      // 2a. aresta
      arestas[7*(j+1) + 0] = topologia[20*i + 13]*topologia[20*i + 14];
      arestas[7*(j+1) + 3] = topologia[20*i + 0];
      if( topologia[20*i + 13]<=topologia[20*i + 14] )
      {
        arestas[7*(j+1) + 1] = topologia[20*i + 13];
        arestas[7*(j+1) + 2] = topologia[20*i + 14];
      }
      else
```

```
{
   arestas[7*(j+1) + 1] = topologia[20*i + 14];
   arestas[7*(j+1) + 2] = topologia[20*i + 13];
}

// 3a. aresta
arestas[7*(j+2) + 0] = topologia[20*i + 15]*topologia[20*i + 14];
arestas[7*(j+2) + 3] = topologia[20*i + 0];
if( topologia[20*i + 15]<=topologia[20*i + 14] )
{
   arestas[7*(j+2) + 1] = topologia[20*i + 15];
   arestas[7*(j+2) + 2] = topologia[20*i + 14];
}
else
{
   arestas[7*(j+2) + 1] = topologia[20*i + 14];
   arestas[7*(j+2) + 2] = topologia[20*i + 15];
}

// 4a. aresta
arestas[7*(j+3) + 0] = topologia[20*i + 12]*topologia[20*i + 15];
arestas[7*(j+3) + 3] = topologia[20*i + 0];
if( topologia[20*i + 12]<=topologia[20*i + 15] )
{
   arestas[7*(j+3) + 1] = topologia[20*i + 12];
   arestas[7*(j+3) + 2] = topologia[20*i + 15];
}
else
{
   arestas[7*(j+3) + 1] = topologia[20*i + 15];
   arestas[7*(j+3) + 2] = topologia[20*i + 12];
}

// 5a. aresta
arestas[7*(j+4) + 0] = topologia[20*i + 12]*topologia[20*i + 16];
arestas[7*(j+4) + 3] = topologia[20*i + 0];
if( topologia[20*i + 12]<=topologia[20*i + 16] )
{
   arestas[7*(j+4) + 1] = topologia[20*i + 12];
   arestas[7*(j+4) + 2] = topologia[20*i + 16];
}
else
{
   arestas[7*(j+4) + 1] = topologia[20*i + 16];
   arestas[7*(j+4) + 2] = topologia[20*i + 12];
}

// 6a. aresta
arestas[7*(j+5) + 0] = topologia[20*i + 13]*topologia[20*i + 17];
arestas[7*(j+5) + 3] = topologia[20*i + 0];
```

```
if( topologia[20*i + 13]<=topologia[20*i + 17] )
{
   arestas[7*(j+5) + 1] = topologia[20*i + 13];
   arestas[7*(j+5) + 2] = topologia[20*i + 17];
}
else
{
   arestas[7*(j+5) + 1] = topologia[20*i + 17];
   arestas[7*(j+5) + 2] = topologia[20*i + 13];
}

// 7a. aresta
arestas[7*(j+6) + 0] = topologia[20*i + 14]*topologia[20*i + 18];
arestas[7*(j+6) + 3] = topologia[20*i + 0];
if( topologia[20*i + 14]<=topologia[20*i + 18] )
{
   arestas[7*(j+6) + 1] = topologia[20*i + 14];
   arestas[7*(j+6) + 2] = topologia[20*i + 18];
}
else
{
   arestas[7*(j+6) + 1] = topologia[20*i + 18];
   arestas[7*(j+6) + 2] = topologia[20*i + 14];
}

// 8a. aresta
arestas[7*(j+7) + 0] = topologia[20*i + 15]*topologia[20*i + 19];
arestas[7*(j+7) + 3] = topologia[20*i + 0];
if( topologia[20*i + 15]<=topologia[20*i + 19] )
{
   arestas[7*(j+7) + 1] = topologia[20*i + 15];
   arestas[7*(j+7) + 2] = topologia[20*i + 19];
}
else
{
   arestas[7*(j+7) + 1] = topologia[20*i + 19];
   arestas[7*(j+7) + 2] = topologia[20*i + 15];
}

// 9a. aresta
arestas[7*(j+8) + 0] = topologia[20*i + 16]*topologia[20*i + 17];
arestas[7*(j+8) + 3] = topologia[20*i + 0];
if( topologia[20*i + 16]<=topologia[20*i + 17] )
{
   arestas[7*(j+8) + 1] = topologia[20*i + 16];
   arestas[7*(j+8) + 2] = topologia[20*i + 17];
}
else
{
   arestas[7*(j+8) + 1] = topologia[20*i + 17];
```

```
    arestas[7*(j+8) + 2] = topologia[20*i + 16];
  }

  // 10a. aresta
  arestas[7*(j+9) + 0] = topologia[20*i + 17]*topologia[20*i + 18];
  arestas[7*(j+9) + 3] = topologia[20*i + 0];
  if( topologia[20*i + 17]<=topologia[20*i + 18] )
  {
    arestas[7*(j+9) + 1] = topologia[20*i + 17];
    arestas[7*(j+9) + 2] = topologia[20*i + 18];
  }
  else
  {
    arestas[7*(j+9) + 1] = topologia[20*i + 18];
    arestas[7*(j+9) + 2] = topologia[20*i + 17];
  }

  // 11a. aresta
  arestas[7*(j+10) + 0] = topologia[20*i + 19]*topologia[20*i + 18];
  arestas[7*(j+10) + 3] = topologia[20*i + 0];
  if( topologia[20*i + 19]<=topologia[20*i + 18] )
  {
    arestas[7*(j+10) + 1] = topologia[20*i + 19];
    arestas[7*(j+10) + 2] = topologia[20*i + 18];
  }
  else
  {
    arestas[7*(j+10) + 1] = topologia[20*i + 18];
    arestas[7*(j+10) + 2] = topologia[20*i + 19];
  }

  // 12a. aresta
  arestas[7*(j+11) + 0] = topologia[20*i + 16]*topologia[20*i + 19];
  arestas[7*(j+11) + 3] = topologia[20*i + 0];
  if( topologia[20*i + 16]<=topologia[20*i + 19] )
  {
    arestas[7*(j+11) + 1] = topologia[20*i + 16];
    arestas[7*(j+11) + 2] = topologia[20*i + 19];
  }
  else
  {
    arestas[7*(j+11) + 1] = topologia[20*i + 19];
    arestas[7*(j+11) + 2] = topologia[20*i + 16];
  }

  i++;
  j=j+12;

}
```

```
printf("1a. parte do algoritmo concluida...\n");


/*imprime para debug*/
for(i=0;i<=(12*nelementos)-1; i++)
{
   for(j=0;j<=7-1; j++)
   {
            fprintf(fp5, "%d\t", arestas[7*i +j]);
   }
   fprintf(fp5, "\n");
}
fprintf(fp5, "\n\n");


/* 2a. parte do algoritmo para geração da numeração global de arestas - Livro Jin */


/* Ordenação da matriz "arestas" a partir do identificador armazenado na primeira coluna */
/* ordenação tipo 'bubblesort' */

for(i=0; i<=(12*nelementos)-1; i++)
{
      for(j=(i+1); j<=(12*nelementos)-1; j++)
      {
            if(arestas[ 7*i + 0 ]> arestas[ 7*j + 0 ])
            {
                  temp[0] = arestas[ 7*j + 0 ];
                  temp[1] = arestas[ 7*j + 1 ];
                  temp[2] = arestas[ 7*j + 2 ];
                  temp[3] = arestas[ 7*j + 3 ];

                  arestas[ 7*j + 0 ] = arestas[ 7*i + 0 ];
                  arestas[ 7*j + 1 ] = arestas[ 7*i + 1 ];
                  arestas[ 7*j + 2 ] = arestas[ 7*i + 2 ];
                  arestas[ 7*j + 3 ] = arestas[ 7*i + 3 ];

                  arestas[ 7*i + 0 ] = temp[0];
                  arestas[ 7*i + 1 ] = temp[1];
                  arestas[ 7*i + 2 ] = temp[2];
                  arestas[ 7*i + 3 ] = temp[3];

            }
      }
}

/*imprime para debug*/
for(i=0;i<=(12*nelementos)-1; i++)
{
   for(j=0;j<=7-1; j++)
```

```
    {
            fprintf(fp6, "%d\t", arestas[7*i +j]);
    }
    fprintf(fp6, "\n");
}
fprintf(fp6, "\n\n");


printf("2a. parte do algoritmo concluida...\n");

/* 3a. parte do algoritmo para geração da numeração global de arestas - Livro Jin
  Ordenação da matriz "arestas" */


/* numeração da primeira aresta e inicialização de numeradores/contadores */
//ultimoid = arestas[7*0 + 0];
//arestas[7*0 + 0] = 1 ;
//armazena a numeração a ser atribuída para a próxima aresta


for(i=0; i<=(12*nelementos)-1; i=i+k)// percorre "arestas" até a última linha
{
      printf("%d\n", i);
      if(arestas[7*i + 0] != 0)
      {
            k=1;
            while(arestas[7*(i+k) +0]==arestas[7*i+0])
            {
                  k++;
            }

            for(j=0; j<=(k-2); j++)
            {
                  for(l=(j+1); l<=k-1; l++)
                  {
                        if((arestas[7*(i+j) +1] == arestas[7*(i+l)+1] ) &&
(arestas[7*(i+j) +2] == arestas[7*(i+l)+2] ))
                        {
                              arestas[7*(i+l) + 0]=0;
                        }
                  }
            }
      }


}


j=1;
for(i=0;i<=(12*nelementos)-1; i++)
```

```
{
                if(arestas[7*i+0]!=0)
                {
                        arestas[7*i+0] =j;
                        j++;
                }
}


/*imprime para debug*/
for(i=0; i<=(12*nelementos)-1; i++)
{
   for(j=0;j<=7-1; j++)
   {
            fprintf(fp7, "%d\t", arestas[7*i +j]);
   }
   fprintf(fp7, "\n");
}
fprintf(fp7, "\n\n");


printf("\nimprimiu debug\n");

/*encontra o número de arestas da malha*/
for(i=0; i<=(12*nelementos-1); i++)
{
      if(arestas[7*i + 0]!=0)
      narestas = arestas[7*i + 0];
}

printf("\nachou numero arestas\n");


// "edgelist será uma matriz com 7 colunas por 'narestas' linhas
// o seu conteúdo é o mesmo da matriz "arestas", mas com as linhas zeradas no
procedimento de numeração eliminadas

edgelist = (int*)malloc(sizeof(int)*7*narestas);


for(i=0; i<=(12*nelementos)-1; i++)
{
      if(arestas[7*i + 0] != 0)
      {
            k = arestas[7*i + 0]-1;
            for(j=0; j<=7-1;j++)
            {
                  edgelist[7*k + j] = arestas[7*i+j];
            }
      }
```

```
}

printf("\nescreveu edgelist\n");

printf("3a. parte do algoritmo concluida...\n");

/* 4a. parte do algoritmo para geração da numeração global de arestas - Livro Jin */
edgenodelist = (int*)malloc(sizeof(int)*21*nelementos);


for(i=0; i<=nelementos-1;i++)
{
        edgenodelist[21*i+0] = topologia[20*i+0];
        for(j=1; j<=8;j++)
        {
                edgenodelist[21*i+j] = topologia[20*i+ (j+11)];
        }


        /* atribui a numeraçao global correspondente à 1a. aresta do elemento "i"*/
        for(j=0; j<=narestas-1;j++)
        {
                if( ( (topologia[20*i + 12]==edgelist[7*j + 1]) &&(topologia[20*i + 13]==
edgelist[7*j + 2]) )||
                  ( (topologia[20*i + 13]==edgelist[7*j + 1]) &&(topologia[20*i + 12]==
edgelist[7*j + 2]) ) )
                {
                  edgenodelist[21*i+9] = edgelist[7*j + 0];
                }
        }

        /* atribui a numeraçao global correspondente à 2a. aresta do elemento "i"*/
        for(j=0; j<=narestas-1;j++)
        {
                if( ( (topologia[20*i + 13]==edgelist[7*j + 1]) &&(topologia[20*i + 14]==
edgelist[7*j + 2]) )||
                  ( (topologia[20*i + 14]==edgelist[7*j + 1]) &&(topologia[20*i + 13]==
edgelist[7*j + 2]) ) )
                {
                  edgenodelist[21*i+10] = edgelist[7*j + 0];
                }
        }

        /* atribui a numeraçao global correspondente à 3a. aresta do elemento "i"*/
        for(j=0; j<=narestas-1;j++)
        {
                if( ( (topologia[20*i + 15]==edgelist[7*j + 1]) &&(topologia[20*i + 14]==
edgelist[7*j + 2]) )||
                  ( (topologia[20*i + 14]==edgelist[7*j + 1]) &&(topologia[20*i + 15]==
edgelist[7*j + 2]) ) )
```

```
            {
              edgenodelist[21*i+11] = edgelist[7*j + 0];
            }
        }

        /* atribui a numeraçao global correspondente à 4a. aresta do elemento "i"*/
        for(j=0; j<=narestas-1;j++)
        {
            if( ( (topologia[20*i + 12]==edgelist[7*j + 1]) &&(topologia[20*i + 15]==
edgelist[7*j + 2]) )||
              ( (topologia[20*i + 15]==edgelist[7*j + 1]) &&(topologia[20*i + 12]==
edgelist[7*j + 2]) ) )
            {
              edgenodelist[21*i+12] = edgelist[7*j + 0];
            }
        }

        /* atribui a numeraçao global correspondente à 5a. aresta do elemento "i"*/
        for(j=0; j<=narestas-1;j++)
        {
            if( ( (topologia[20*i + 12]==edgelist[7*j + 1]) &&(topologia[20*i + 16]==
edgelist[7*j + 2]) )||
              ( (topologia[20*i + 16]==edgelist[7*j + 1]) &&(topologia[20*i + 12]==
edgelist[7*j + 2]) ) )
            {
              edgenodelist[21*i+13] = edgelist[7*j + 0];
            }
        }

        /* atribui a numeraçao global correspondente à 6a. aresta do elemento "i"*/
        for(j=0; j<=narestas-1;j++)
        {
            if( ( (topologia[20*i + 13]==edgelist[7*j + 1]) &&(topologia[20*i + 17]==
edgelist[7*j + 2]) )||
              ( (topologia[20*i + 17]==edgelist[7*j + 1]) &&(topologia[20*i + 13]==
edgelist[7*j + 2]) ) )
            {
              edgenodelist[21*i+14] = edgelist[7*j + 0];
            }
        }

        /* atribui a numeraçao global correspondente à 7a. aresta do elemento "i"*/
        for(j=0; j<=narestas-1;j++)
        {
            if( ( (topologia[20*i + 14]==edgelist[7*j + 1]) &&(topologia[20*i + 18]==
edgelist[7*j + 2]) )||
              ( (topologia[20*i + 18]==edgelist[7*j + 1]) &&(topologia[20*i + 14]==
edgelist[7*j + 2]) ) )
            {
              edgenodelist[21*i+15] = edgelist[7*j + 0];
```

```
            }
        }

        /* atribui a numeraçao global correspondente à 8a. aresta do elemento "i"*/
        for(j=0; j<=narestas-1;j++)
        {
            if( ( (topologia[20*i + 15]==edgelist[7*j + 1]) &&(topologia[20*i + 19]==
edgelist[7*j + 2]) )||
              ( (topologia[20*i + 19]==edgelist[7*j + 1]) &&(topologia[20*i + 15]==
edgelist[7*j + 2]) ) )
            {
              edgenodelist[21*i+16] = edgelist[7*j + 0];
            }
        }

        /* atribui a numeraçao global correspondente à 9a. aresta do elemento "i"*/
        for(j=0; j<=narestas-1;j++)
        {
            if( ( (topologia[20*i + 16]==edgelist[7*j + 1]) &&(topologia[20*i + 17]==
edgelist[7*j + 2]) )||
              ( (topologia[20*i + 17]==edgelist[7*j + 1]) &&(topologia[20*i + 16]==
edgelist[7*j + 2]) ) )
            {
              edgenodelist[21*i+17] = edgelist[7*j + 0];
            }
        }

        /* atribui a numeraçao global correspondente à 10a. aresta do elemento "i"*/
        for(j=0; j<=narestas-1;j++)
        {
            if( ( (topologia[20*i + 17]==edgelist[7*j + 1]) &&(topologia[20*i + 18]==
edgelist[7*j + 2]) )||
              ( (topologia[20*i + 18]==edgelist[7*j + 1]) &&(topologia[20*i + 17]==
edgelist[7*j + 2]) ) )
            {
              edgenodelist[21*i+18] = edgelist[7*j + 0];
            }
        }

        /* atribui a numeraçao global correspondente à 11a. aresta do elemento "i"*/
        for(j=0; j<=narestas-1;j++)
        {
            if( ( (topologia[20*i + 19]==edgelist[7*j + 1]) &&(topologia[20*i + 18]==
edgelist[7*j + 2]) )||
              ( (topologia[20*i + 18]==edgelist[7*j + 1]) &&(topologia[20*i + 19]==
edgelist[7*j + 2]) ) )
            {
              edgenodelist[21*i+19] = edgelist[7*j + 0];
            }
        }
```

```
        /* atribui a numeraçao global correspondente à 12a. aresta do elemento "i"*/
        for(j=0; j<=narestas-1;j++)
        {
              if( ( (topologia[20*i + 16]==edgelist[7*j + 1]) &&(topologia[20*i + 19]==
edgelist[7*j + 2]) )||
                ( (topologia[20*i + 19]==edgelist[7*j + 1]) &&(topologia[20*i + 16]==
edgelist[7*j + 2]) ) )
              {
                edgenodelist[21*i+20] = edgelist[7*j + 0];
              }
        }
  }

  printf("4a. parte do algoritmo concluida...\n");

  /************************************************************
   *          Geração dos arquivos de saída                   *
   ************************************************************/

  /*Escreve o arquivo "coordinates.dat"*/
  fprintf(fp2, "%d\n", nnos);
  for(i=0; i<=nnos-1 ;i++)
  {
              fprintf(fp2, "%.0f\t", coordenadas[4*i + 0]);
              for(j=1;j<=4-1;j++)
              {
                        fprintf(fp2, "%f\t", coordenadas[4*i + j]);
              }
              fprintf(fp2, "\n");
  }

  /*Escreve o arquivo "edgelist.dat"*/
  fprintf(fp3, "%d\n", narestas);
  for(i=0; i<=narestas-1 ;i++)
  {
              for(j=0;j<=3-1;j++)
              {
                        fprintf(fp3, "%d\t", edgelist[7*i + j]);
              }
              fprintf(fp3, "\n");
  }

  /*Escreve o arquivo "edgenodelist.dat"*/
  fprintf(fp4, "%d\n", nelementos);
```

```
    for(i=0; i<=nelementos-1 ;i++)
    {
                for(j=0;j<=21-1;j++)
                {
                        fprintf(fp4, "%d\t", edgenodelist[21*i + j]);
                }
                fprintf(fp4, "\n");
    }

    /*Escreve arquivo para verificações*/
    fprintf(fp8, "%d ", narestas);
    for(i=0; i<=(7*narestas)-1; i++)
    {
          fprintf(fp8, "%d ", edgelist[i]);
    }



    printf("geracao dos arquivos de saida concluida...\n");

    fclose(fp);
    fclose(fp2);
    fclose(fp3);
    fclose(fp4);
    fclose(fp5);
    fclose(fp6);
    fclose(fp7);

    printf("%d %d %d\n", nelementos, nnos, narestas);
    system("PAUSE");


    return 0;
}
```

APÊNDICE B - Transcrição do arquivo PF3 correspondente ao caso teste

```
Fichier cree par F3DXPE 1.0. Date: 05/09/05 11:06:51
       3              NOMBRE DE DIMENSIONS DU DECOUPAGE
       8              NOMBRE  D'ELEMENTS
       8              NOMBRE  D'ELEMENTS VOLUMIQUES
       0              NOMBRE  D'ELEMENTS SURFACIQUES
       0              NOMBRE  D'ELEMENTS LINEIQUES
       0              NOMBRE  D'ELEMENTS PONCTUELS
       0              NOMBRE DE MACRO-ELEMENTS
      27              NOMBRE DE POINTS
       1              NOMBRE DE REGIONS
       1              NOMBRE DE REGIONS VOLUMIQUES
       0              NOMBRE DE REGIONS SURFACIQUES
       0              NOMBRE DE REGIONS LINEIQUES
       0              NOMBRE DE REGIONS PONCTUELLES
       0              NOMBRE DE REGIONS MACRO-ELEMENTAIRES
      20              NOMBRE DE NOEUDS DANS 1 ELEMENT (MAX)
      20              NOMBRE DE POINTS D'INTEGRATION / ELEMENT (MAX)
 NOMS DES REGIONS
     REGIONS VOLUMIQUES

REGIAO
------
       0      4      1
 DESCRIPTEUR DE TOPOLOGIE DES ELEMENTS
           1       7    2202       1       4       0      15       8       0       0       0       0
          16       7      19      24      23      17      26      27
           2       7    2202       1       4       0      15       8       0       0       0       0
          23      17      26      27      14       6      15      22
           3       7    2202       1       4       0      15       8       0       0       0       0
          24      19       8      18      27      26      20      25
           4       7    2202       1       4       0      15       8       0       0       0       0
           3      16      24      12      10      23      27      21
           5       7    2202       1       4       0      15       8       0       0       0       0
          27      26      20      25      22      15       5      11
           6       7    2202       1       4       0      15       8       0       0       0       0
          10      23      27      21       4      14      22       9
           7       7    2202       1       4       0      15       8       0       0       0       0
          12      24      18       2      21      27      25      13
           8       7    2202       1       4       0      15       8       0       0       0       0
          21      27      25      13       9      22      11       1
 COORDONNEES DES NOEUDS
       1       0.0000000       0.0000000       0.0000000
       2   0.2000000E-02       0.0000000       0.0000000
       3   0.2000000E-02   0.2000000E-02       0.0000000
       4       0.0000000   0.2000000E-02       0.0000000
       5       0.0000000       0.0000000   0.2000000E-02
       6       0.0000000   0.2000000E-02   0.2000000E-02
       7   0.2000000E-02   0.2000000E-02   0.2000000E-02
       8   0.2000000E-02       0.0000000   0.2000000E-02
       9       0.0000000   0.1000000E-02       0.0000000
      10   0.1000000E-02   0.2000000E-02       0.0000000
      11       0.0000000       0.0000000   0.1000000E-02
      12   0.2000000E-02   0.1000000E-02       0.0000000
      13   0.1000000E-02       0.0000000       0.0000000
      14       0.0000000   0.2000000E-02   0.1000000E-02
      15       0.0000000   0.1000000E-02   0.2000000E-02
      16   0.2000000E-02   0.2000000E-02   0.1000000E-02
      17   0.1000000E-02   0.2000000E-02   0.2000000E-02
      18   0.2000000E-02       0.0000000   0.1000000E-02
      19   0.2000000E-02   0.1000000E-02   0.2000000E-02
      20   0.1000000E-02       0.0000000   0.2000000E-02
      21   0.1000000E-02   0.1000000E-02       0.0000000
      22       0.0000000   0.1000000E-02   0.1000000E-02
      23   0.1000000E-02   0.2000000E-02   0.1000000E-02
      24   0.2000000E-02   0.1000000E-02   0.1000000E-02
      25   0.1000000E-02       0.0000000   0.1000000E-02
      26   0.1000000E-02   0.1000000E-02   0.2000000E-02
      27   0.1000000E-02   0.1000000E-02   0.1000000E-02
==== DECOUPAGE  TERMINE
```

APÊNDICE C - Transcrição do arquivo edgelist.dat correspondente ao caso teste

```
54
1       1       9
2       1       11
3       1       13
4       2       12
5       2       13
6       3       10
7       2       18
8       3       12
9       4       9
10      4       10
11      3       16
12      5       11
13      4       14
14      5       15
15      6       14
16      6       15
17      5       20
18      6       17
19      7       16
20      7       17
21      7       19
22      8       18
23      8       19
24      8       20
25      9       21
26      9       22
27      10      21
28      10      23
29      11      22
30      12      21
31      13      21
32      11      25
33      12      24
34      14      22
35      14      23
36      13      25
37      15      22
38      16      23
39      16      24
40      15      26
41      17      23
42      18      24
43      17      26
44      18      25
45      19      24
46      19      26
47      20      25
48      20      26
49      21      27
50      22      27
51      23      27
52      24      27
53      25      27
54      26      27
```

APÊNDICE D - Transcrição do arquivo edgenodelist.dat correspondente ao caso teste

```
8
1 16  7 19 24 23 17 26 27 19 21 45 39 38 20 46 52 41 43 54 51
2 23 17 26 27 14  6 15 22 41 43 54 51 35 18 40 50 15 16 37 34
3 24 19  8 18 27 26 20 25 45 23 22 42 52 46 24 44 54 48 47 53
4  3 16 24 12 10 23 27 21 11 39 33  8  6 38 52 30 28 51 49 27
5 27 26 20 25 22 15  5 11 54 48 47 53 50 40 17 32 37 14 12 29
6 10 23 27 21  4 14 22  9 28 51 49 27 10 35 50 25 13 34 26  9
7 12 24 18  2 21 27 25 13 33 42  7  4 30 52 44  5 49 53 36 31
8 21 27 25 13  9 22 11  1 49 53 36 31 25 50 32  3 26 29  2  1
```

APÊNDICE E - Transcrição do arquivo coordinates.dat correspondente ao caso teste

27

1	0.000000	0.000000	0.000000
2	0.002000	0.000000	0.000000
3	0.002000	0.002000	0.000000
4	0.000000	0.002000	0.000000
5	0.000000	0.000000	0.002000
6	0.000000	0.002000	0.002000
7	0.002000	0.002000	0.002000
8	0.002000	0.000000	0.002000
9	0.000000	0.001000	0.000000
10	0.001000	0.002000	0.000000
11	0.000000	0.000000	0.001000
12	0.002000	0.001000	0.000000
13	0.001000	0.000000	0.000000
14	0.000000	0.002000	0.001000
15	0.000000	0.001000	0.002000
16	0.002000	0.002000	0.001000
17	0.001000	0.002000	0.002000
18	0.002000	0.000000	0.001000
19	0.002000	0.001000	0.002000
20	0.001000	0.000000	0.002000
21	0.001000	0.001000	0.000000
22	0.000000	0.001000	0.001000
23	0.001000	0.002000	0.001000
24	0.002000	0.001000	0.001000
25	0.001000	0.000000	0.001000
26	0.001000	0.001000	0.002000
27	0.001000	0.001000	0.001000

APÊNDICE F - Exemplo do formato do arquivo de saída do programa HFG3D

(corrente injetada de 1000 A na freqüência de 500kHz em geometria com 44652 nós)

Freq.(kHz) = 500
Z = (21.505 ; 23.3518) =
31.7454 / 47.3576 graus
I = (1000 ; 0.00387171) =
1000 / 0.000221833 graus
S (kVA) = (10752.5 ; 11675.9) =
15872.7 / 47.3576 graus
VI* (kVA) = (21505 ; 23351.8) =
31745.5 / 47.3576 graus
Vdef (kV)= (21.5049 ; 23.3519) =
31.7454 / 47.3578 graus
R (ohm) = 21.505
L (microH) = 9.14442
C (nF) = 59.207
XL (ohm) = 28.7281
XC (ohm) = 5.37622

44652
1.96954780703922e+04 1.23729796592832e+04
1.98836372639682e+04 1.29676288789242e+04
1.98836495620582e+04 1.29676137347496e+04
2.01485353748669e+04 1.39047044754362e+04
2.04391887729684e+04 1.49031443016466e+04
2.00842083997203e+04 1.34966682181167e+04
2.05596571313096e+04 1.56548447062714e+04
2.01473161582157e+04 1.38186874446093e+04
1.95548479653142e+04 1.18084153633049e+04
1.96881408029379e+04 1.21858721528382e+04
1.94754661282341e+04 1.13341992012498e+04
1.95759710322153e+04 1.15882725378918e+04
1.93898826614986e+04 1.10284567726722e+04
1.94714360598882e+04 1.12256914261047e+04
2.00842304064403e+04 1.34966403956521e+04
2.04391986602137e+04 1.49031312986938e+04
2.01473426808398e+04 1.38186539197400e+04
2.05596721687175e+04 1.56548259968368e+04
1.98278843033361e+04 1.24649639798202e+04
1.98500993183270e+04 1.26020671020482e+04
1.96811040767875e+04 1.17437507905937e+04
1.96714171812056e+04 1.18025929556042e+04
1.95485248543235e+04 1.13333483713734e+04
1.94160967219273e+04 1.13762746789334e+04
2.08420734615269e+04 1.67304755560049e+04
2.10301293061748e+04 1.85829382101996e+04
2.10301241623075e+04 1.85829448809030e+04
2.15833605677657e+04 2.42409167366560e+04
1.95548364624581e+04 1.18084297895601e+04

Obs: O restante do arquivo de saída foi omitido em função do seu tamanho excessivo.

APÊNDICE G - Código fonte do programa de interpolação

```
#include <stdio.h>
#include <math.h>
#include <complex.h>

#define MAXLINHA 100
#define NOMBREDELEMENTS 18
#define NOMBREDELEMENTSVOLUMIQUES 29
#define NOMBREDEMACROELEMENTS 24
#define DTOPOLOGIEDESELEMENTS 37
#define COORDONNEESDESNOEUDS 22


int main()
{
  /*************************************************************************
   *            Declaração das variáveis                                   *
   *************************************************************************/

   int i, j, k, l, m, n; // contadores

   // variáveis envolvidas na importação de dados e na escrita de arquivos de saída

   int *tophexa; // ponteiro para matriz que armazena a topologia dos elementos hexaedricos
   int *topquadr; //ponteiro para matriz que armazena a topologia dos elementos hexaedricos
   float *coordenadas; //ponteiro paramatriz que armazena as coordenadas dos pontos da
malha
   char s[MAXLINHA]; // string de caracteres usado nas buscas na leitura do PF3
   int teste; // variável de teste auxiliar na leitura do PF3
   double a , b; //variáveis auxiliares na leitura do arquivo solucao.txt
   double complex *potenciais; //ponteiro para matriz armazenando os potenciais contidos em
solucao.txt
   int nhexaedros; //numero de elementos hexaedricos extraídos do PF3
   int nquadrilateros; //numero de elementos quadrilaterais no plano do solo extraídos do PF3
   int nnos; //numero de nos da malha extraido do arquivo PF3
   int nnossolucao; //numero de nos da malha extraido do arquivo solucao.txt

   // variáveis usadas na interpolação proprimente dita.


   int *N1;
   int *N2;
   int *N3;
   int *N4;
   double *X; // ponteiro para vetor que armazena as coordenadas X dos nós da malha
   double *Y; // ponteiro para vetor que armazena as coordenadas Y dos nós da malha
   double PX[41][41]; //matriz com as coordenadas X dos pontos do nível do solo
   double PY[41][41]; //matriz com as coordenadas Y dos pontos do nível do solo
   double V[41][41]; // matriz que armazenará o módulo apenas (sem fase) dos fasores de
potencial elétrico no nível do solo
```

```
   double ME[41][41]; //matriz que armazenará o módulo do campo elétrico calculado no
plano do solo
   double EX[41][41]; //matriz que armazenará a componente X do campo elétrico no nível do
solo
   double EY[41][41];
   double px, py; // variáveis auxiliares na construção de PX e PY
   double xmax, xmin, ymax, ymin;
   double complex Lx, Ly;
   double complex v1, v2, v3, v4;
   double complex H1, H2, H3, H4;
   double complex dH1dx, dH2dx, dH3dx, dH4dx, dH1dy, dH2dy, dH3dy, dH4dy;
   double C, xc,yc;

  for(i=0;i<=40;i++)
  {
              for(j=0; j<=40;j++)
              {
                    ME[i][j]=0;
                    EX[i][j]=0;
                    EY[i][j]=0;
                    V[i][j]=0;
              }
  }

  /*************************************************************************
   *    Abertura dos arquivos de leitura e para gravação                   *
   *************************************************************************/

  FILE *fp1;
  fp1 = fopen("solucao.txt", "r");

  FILE *fp2;
  fp2 = fopen("gs60solo.pf3", "r");

  FILE *fp3;
  fp3 = fopen("potenciais.txt", "w");

  FILE *fp4;
  fp4 = fopen("tophexa.txt", "w");

  FILE *fp5;
  fp5 = fopen("topquadr.txt", "w");

  FILE *fp6;
  fp6 = fopen("coordenadas.txt", "w");

  FILE *fp7;
  fp7 = fopen("PX.m", "w");

 FILE *fp8;
```

```
fp8 = fopen("PY.m", "w");

FILE *fp9;
fp9 = fopen("XY.txt", "w");

FILE *fp10;
fp10 = fopen("N1N2N3N4.txt", "w");

FILE *fp11;
fp11 = fopen("pertencequadrado.txt", "w");

FILE *fp12;
fp12 = fopen("V.m", "w");

FILE *fp13;
fp13 = fopen("EX.m", "w");

FILE *fp14;
fp14 = fopen("ME.m", "w");



/*************************************************************************
 *    Leitura do arquivo PF3 e armazenamento do seu conteúdo             *
 *************************************************************************/

/* Busca pelo string "NOMBRE  D'ELEMENTS" */
teste=1;
for(i=0; teste!=0; i++)
{

      /* o tamanho máximo de "s" é o do string "NOMBRE  D'ELEMENTS": apos */
      if(i==NOMBREDELEMENTS)
      {
            for(j = 0; j<=NOMBREDELEMENTS-2; j++)
            {
                s[j]=s[j+1];
            }

            i--;
      }


      s[i] = getc(fp2);
      s[i+1] = '\0';
      teste = strcmp("NOMBRE  D'ELEMENTS",s);
}
```

```
  /* Cópia do próximo número após "NOMBRE D'ELEMENTS". Trata-se do número de
elementos volumétricos da malha.*/
  fscanf(fp2, "%d", &nhexaedros);


  /* Busca pelo string "NOMBRE  D'ELEMENTS VOLUMIQUES" */
  teste=1;
  for(i=0; teste!=0; i++)
  {
       if(i==NOMBREDELEMENTSVOLUMIQUES)
       {
                for(j = 0; j<=NOMBREDELEMENTSVOLUMIQUES-2; j++)
                {
                    s[j]=s[j+1];
                }

                i--;
       }


       s[i] = getc(fp2);
       s[i+1] = '\0';
       teste = strcmp("NOMBRE  D'ELEMENTS VOLUMIQUES",s);
  }


  /* Cópia do próximo número após "NOMBRE  D'ELEMENTS VOLUMIQUES". Trata-se
do número de elementos superficiais no plano do solo*/
  fscanf(fp2, "%d", &nquadrilateros);

   /* Busca pelo string "NOMBRE DE MACRO-ELEMENTS" */
  teste=1;
  for(i=0; teste!=0; i++)
  {

       if(i==NOMBREDEMACROELEMENTS)
       {
                for(j = 0; j<=NOMBREDEMACROELEMENTS-2; j++)
                {
                    s[j]=s[j+1];
                }

                i--;
       }


       s[i] = getc(fp2);
       s[i+1] = '\0';
```

```
        teste = strcmp("NOMBRE DE MACRO-ELEMENTS",s);
    }

    /* Cópia do próximo número após "NOMBRE DE MACRO-ELEMENTS". Trata-se do
número de nós da malha.*/
    fscanf(fp2, "%d", &nnos);



    /* Busca pelo string "DESCRIPTEUR DE TOPOLOGIE DES ELEMENTS" */
    teste=1;
    for(i=0; teste!=0; i++)
    {

        if(i==DTOPOLOGIEDESELEMENTS)
        {
                for(j = 0; j<=DTOPOLOGIEDESELEMENTS-2; j++)
                {
                    s[j]=s[j+1];
                }

                i--;
        }


        s[i] = getc(fp2);
        s[i+1] = '\0';
        teste = strcmp("DESCRIPTEUR DE TOPOLOGIE DES ELEMENTS",s);
    }



    /* Cópia do trecho do arquivo pf3 abaixo de "DESCRIPTEUR DE TOPOLOGIE DES
ELEMENTS" */

    // "tophexa" será utilizado como uma matriz com 20 colunas e 'nhexaedros' linhas
    // as 12 primeiras colunas armazenam a 1a. linha do elemento hexaédrico do PF3
    // as 8 últimas colunas armazenam a 2a. linha (nós numerados glabalmente) do elemento

    tophexa = (int*)malloc(sizeof(int)*nhexaedros*20);

    for(i=0; i<=((12+8)*nhexaedros)-1; i++)
    {
        fscanf(fp2, "%d", &tophexa[i]);
    }

    // "topquadr" será utilizado como uma matriz com 16 colunas e 'nquadrilateros' linhas
    // as 12 primeiras colunas armazenam a 1a. linha do elemento hexaédrico do PF3
```

```
// as 4 últimas colunas armazenam a 2a. linha (nós numerados glabalmente) do elemento
topquadr = (int*)malloc(sizeof(int)*nquadrilateros*16);

for(i=0; i<=((12+4)*nquadrilateros)-1; i++)
{
      fscanf(fp2, "%d", &topquadr[i]);
}


/* Busca pelo string "COORDONNEES DES NOEUDS" */
teste=1;
for(i=0; teste!=0; i++)
{

      if(i==COORDONNEESDESNOEUDS)
      {
            for(j = 0; j<=COORDONNEESDESNOEUDS-2; j++)
            {
                s[j]=s[j+1];
            }

            i--;
      }


      s[i] = getc(fp2);
      s[i+1] = '\0';
      teste = strcmp("COORDONNEES DES NOEUDS",s);
}

/* Cópia do trecho do arquivo pf3 abaixo de "COORDONNEES DES NOEUDS" */

// "coordenadas" será utilizado como uma matriz com 4 colunas e 'nnos' linhas
// a 1a. coluna armazenam a numeracão global do nó contida no arquivo PF3
// as 3 últimas colunas armaznam as coordenadas do nó correspondente

coordenadas = (float*)malloc(sizeof(float)*nnos*4);

for(i=0; i<=(4*nnos)-1; i++)
{
      fscanf(fp2, "%f", &coordenadas[i]);
}


/*************************************************************************
 * Leitura e armazenamento do arquivo 'solucao.txt' no vetor complexo 'potenciais'      *
*************************************************************************/

  fscanf(fp1, "%d", &nnossolucao);
```

```
    potenciais = (double complex*)malloc(sizeof(double complex)*nnossolucao);

    for(i = 0; i<=nnossolucao-1; i++)
    {
        fscanf(fp1, "%lf", &a);
        fscanf(fp1, "%lf", &b);
        potenciais[i] = a + b*_Complex_I;
        fprintf(fp3,"%10.15lf + j%10.15lf\n", creal(potenciais[i]), cimag(potenciais[i]));
    }


/*************************************************************************
*   Transposição dos dados armazenados para formato adequado à interpolação e      *
*   geração de matrizes necessárias ao cálculo                                     *
*************************************************************************/

    // criação dos vetores X e Y de coordenadas
    X = (double*)malloc(sizeof(double)*nnos);
    Y = (double*)malloc(sizeof(double)*nnos);

    for(i=0; i<=nnos-1; i++)
    {
          X[i] = coordenadas[4*i + 1];
          Y[i] = coordenadas[4*i + 2];
          fprintf(fp9, "%d\t%lf\t%lf\n", i,X[i],Y[i]);
    }

    // criação dos vetores N1, N2, N3 e N4 armazenando respectivamente os nós 1 a 4 dos
quadriláteros
    N1 = (int*)malloc(sizeof(int)*nquadrilateros);
    N2 = (int*)malloc(sizeof(int)*nquadrilateros);
    N3 = (int*)malloc(sizeof(int)*nquadrilateros);
    N4 = (int*)malloc(sizeof(int)*nquadrilateros);

    for(i=0; i<=nquadrilateros-1; i++)
    {
          N1[i] = topquadr[16*i + 12];
          N2[i] = topquadr[16*i + 13];
          N3[i] = topquadr[16*i + 14];
          N4[i] = topquadr[16*i + 15];
          fprintf(fp10, "%d\t%d\t%d\t%d\t%d\n", i, N1[i], N2[i], N3[i], N4[i]);
    }

    //renumeração dos quadriláteros de 1 até 'nquadrilateros'
    for(i=0; i<=nquadrilateros-1; i++)
    {
          topquadr[16*i + 0] = i+1;
    }
```

```
for( m = -20 ; m <= 20;  m++)
{
   for( n = -20; n<=20; n++)
   {
      j = m+20;
      k = n+20;
      px = m*2;
      py = n*2;
      PX[k][j] = px; //geração das matrizes de coordenadas do domínio
      PY[k][j] = py; // no plano do solo


      for(i = 0; i<=nquadrilateros-1; i++)
      {
         //Descobre as coordenadas xmax e ymax limites do elemento i
         xmax = X[topquadr[16*i + 12]-1];
         ymax = Y[topquadr[16*i + 12]-1];
         for(l=12; l <=16-1; l++)
         {
               if(X[topquadr[16*i + l]-1]>xmax)
               {
                        xmax = X[topquadr[16*i + l]-1];
               }

               if(Y[topquadr[16*i + l]-1]>ymax)
               {
                        ymax = Y[topquadr[16*i + l]-1];
               }
         }

         //Descobre as coordenadas xmin e ymin limites do elemento i
         xmin = X[topquadr[16*i + 12]-1];
         ymin = Y[topquadr[16*i + 12]-1];
         for(l=12; l <=16-1; l++)
         {
               if(X[topquadr[16*i + l]-1]<xmin)
               {
                        xmin = X[topquadr[16*i + l]-1];
               }

               if(Y[topquadr[16*i + l]-1]<ymin)
               {
                        ymin = Y[topquadr[16*i + l]-1];
               }
         }

         // se o ponto (px,py) pertence ao quadrilatero 'i', faz a interpolação para ele
         if( (xmin<=px)&&(px<=xmax)&&(ymin<=py)&&(py<=ymax) )
         {
```

```
                fprintf(fp11, "%lf,%lf pertence a %d\n", px, py, i+1);

                Lx = fabs(xmax-xmin);
                Ly = fabs(ymax-ymin);

                xc = (xmax+xmin)/2;
                yc = (ymax+ymin)/2;

                C = 1/(Lx*Ly);

                v1 = potenciais[N1[i]-1];
                v2 = potenciais[N2[i]-1];
                v3 = potenciais[N3[i]-1];
                v4 = potenciais[N4[i]-1];



                H1 = C *( xc+(Lx/2)-px)*( yc+(Ly/2)-py);
                H2 =  C*(-xc+(Lx/2)+px)*( yc+(Ly/2)-py);
                H3 =  C*( -xc+(Lx/2)+px)*(-yc+(Ly/2)+py);
                H4 =  C*( xc+(Lx/2)-px)*(-yc+(Ly/2)+py);

                dH1dx = C*(-yc-(Ly/2)+py);
                dH2dx = C*( yc+(Ly/2)-py);
                dH3dx = C*(-yc+(Ly/2)+py);
                dH4dx = C*( yc-(Ly/2)-py);

                dH1dy = C*(-xc-(Lx/2)+px);
                dH2dy = C*( xc-(Lx/2)-px);
                dH3dy = C*(-xc+(Lx/2)+px);
                dH4dy = C*( xc+(Lx/2)-px);


                V[k][j] = cabs( (H1*v1)+(H2*v2)+(H3*v3)+(H4*v4) );
                printf("%lf\t%lf\t%lf\n", V[k][j], creal(v1), cimag(v1));
                EX[k][j] = cabs ((v1*dH1dx)+(v2*dH2dx)+(v3*dH3dx)+(v4*dH4dx) );
                EY[k][j] = cabs( (v1*dH1dy)+(v2*dH2dy)+(v3*dH3dy)+(v4*dH4dy) );
                ME[k][j] = sqrt(  pow(EX[k][j], 2) + pow(EY[k][j], 2) );


                break;
            }

            else
            {
                continue;
            }
        }
    }
}
```

```
    fprintf(fp7, "PX = [\n");
    fprintf(fp8, "PY = [\n");
    fprintf(fp12, "V = [\n");
    fprintf(fp13, "EX = [\n;");
    fprintf(fp14, "ME = [\n");
    for(i=0;i<=41-1;i++)
    {
                for(j=0;j<=41-1;j++)
                {
                            fprintf(fp7,"%.15lf\t", PX[i][j]);
                            fprintf(fp8,"%.15lf\t", PY[i][j]);
                            fprintf(fp12,"%.15lf\t", V[i][j]);
                            fprintf(fp13,"%.15lf\t", EX[i][j]);
                            fprintf(fp14,"%.15lf\t", ME[i][j]);
                }
                fprintf(fp7, ";\n");
                fprintf(fp8, ";\n");
                fprintf(fp12, ";\n");
                fprintf(fp13, ";\n");
                fprintf(fp14, ";\n");
    }
    fprintf(fp7, "];");
    fprintf(fp8, "];");
    fprintf(fp12, "];");
    fprintf(fp13, "];");
    fprintf(fp14, "];");


    fclose(fp7);
    fclose(fp8);
    fclose(fp12);
    fclose(fp13);
    fclose(fp14);

/*************************************************************************
*   Escreve arquivos de saída                                            *
*************************************************************************/

   for(i = 0; i<=nhexaedros-1;i++)
   {
       for(j = 0; j<= 20-1; j++)
       {
           fprintf(fp4, "%d\t", tophexa[20*i + j]);
       }
       fprintf(fp4, "\n");
   }
```

```
 for(i = 0; i<=nquadrilateros-1;i++)
 {
     for(j = 0; j<= 16-1; j++)
     {
         fprintf(fp5, "%d\t", topquadr[16*i + j]);
     }
     fprintf(fp5, "\n");
 }


 for(i = 0; i<=nnos-1;i++)
 {
     fprintf(fp6, "%.0f\t", coordenadas[4*i + 0]);
     for(j = 1; j<= 4-1; j++)
     {
         fprintf(fp6, "%f\t", coordenadas[4*i + j]);
     }
     fprintf(fp6, "\n");
 }


 if(nnos==nnossolucao)
 {
       printf("'nnos' e 'nnossolucao' sao iguais!\n");
 }

 system("PAUSE");
 return 0;
}
```