UNIVERSIDADE FEDERAL FLUMINENSE

METHODS II :
Um sistema interativo de apoio ao aprendizado de métodos numéricos para problemas de Equações Diferenciais Parciais

DENIS ANDRÉ RIBEIRO LEAL
MARIANA LISBOA DA COSTA

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal Fluminense como parte dos requisitos para obtenção do Grau de Bacharel em Ciência da Computação

Banca Examinadora

Profª. Regina Célia Paula Leal Toledo, DSc
Prof. Marco Antonio M. Silva Ramos, DSc
Prof. Leonardo Cruz da Costa, MSc
Maurício José Machado Guedes, MSc

Departamento de Ciência da Computação

Niterói, 10 de agosto de 2004

NITERÓI

2004

Título: **METHODS II: Um sistema interativo de apoio ao aprendizado de métodos numéricos para problemas de Equações Diferenciais Parciais**

Autores: Denis André Ribeiro Leal e Mariana Lisboa da Costa

Orientadora: Regina Célia Paula Leal Toledo

BANCA EXAMINADORA

_____

Profª. Regina Célia Paula Leal Toledo, DSc
(Orientadora)

_____

Prof. Marco Antonio M. Silva Ramos, DSc
(Co-Orientador)

_____

Prof. Leonardo Cruz da Costa, MSc

_____

Maurício José Machado Guedes, MSc

Data da apresentação: _____/_____/_____

De modo especial, a todos os nossos amigos e professores, que com amizade e companheirismo caminharam junto conosco em toda essa etapa importante de nossas vidas.

AGRADECIMENTOS

Em primeiro lugar agradecemos a Deus, fonte da vida e da graça. Agradecemos por nos iluminar durante toda essa trajetória.
À nossa orientadora, *Profª. Regina Célia Paula Leal Toledo* que jamais deixou de nos incentivar. Sem a sua orientação, dedicação e auxílio, o estudo aqui apresentado seria praticamente impossível.
Ao nosso co-orientador, *Prof. Marco Antônio Silva Ramos*, pela sua atenção e tempo destinados tanto a nós, quanto a este trabalho.
Aos nossos pais *Mercia e Francisco (in memoriam), Sílvia e Armando*, que apesar das dificuldades enfrentadas, sempre incentivaram nossos estudos.
E a todos que, de maneira direta ou indireta, contribuíram de alguma forma para a realização deste trabalho.

"*Na natureza os resultados práticos são não só o meio de melhorar o bem-estar, mas a garantia da verdade*"
Francis Bacon

RESUMO

Neste trabalho estamos interessados em resolver as equações por métodos numéricos aproximados. Inicialmente apresentamos como um domínio bidimensional contínuo discretizado para aplicação do Método das Diferenças Finitas. O sistema foi desenvolvido para auxiliar os alunos a resolverem problemas de equações parciais. É uma forma prática do aluno lidar com os problemas e seus métodos, podendo visualizar os resultados através de gráficos. Acreditamos que isto certamente lhe dará um maior entendimento e um maior interesse sobre a teoria que lhe é apresentada. Basta entrar com os dados para visualizar um gráfico com as soluções aproximada e exata do problema resolvido utilizando o método escolhido. Vale ainda ressaltar, que o sistema possui um módulo teórico, cujo conteúdo serve como base para o aluno na utilização do módulo prático.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. Introdução

Esta monografia apresenta o *Methods II*, sistema que foi desenvolvido como projeto final do curso de Ciência da Computação da Universidade Federal Fluminense.

O presente projeto consistiu em desenvolver um sistema instrucional de apoio ao estudo de Problema de Valor de Contorno (PVC) em Equações Diferenciais Parciais (EDP), com objetivo de auxiliar os alunos que estudam métodos numéricos a ter uma visualização gráfica do comportamento dos mesmos. O projeto foi baseado, em um primeiro momento, no *Methods I*, desenvolvido por Marcelo Zuim em 2000. A diferença entre os dois projetos é o fato de o *Methods I* resolver problemas de Equações Diferenciais Ordinárias (EDO), enquanto o *Methods II* resolve problemas de Equações Diferenciais Parciais (EDP).

Em busca de um melhor aprendizado e à procura de uma maneira de aumentar a motivação dos alunos nas disciplinas que envolvem métodos numéricos, vem se tentando criar novas formas para facilitar seu ensino e criar interesse por parte deles. Dentre elas surgiu a idéia de fazer um aplicativo que mostrasse de maneira simples e objetiva o resultado da aplicação dos métodos numéricos, onde o aluno pudesse estudar o comportamento do método em determinado problema. Ele ainda teria a opção de ver graficamente o que estava ocorrendo com a solução em determinado instante e compará-la com a sua respectiva solução exata, e também verificar facilmente se o método seria estável ou instável. O aplicativo apresentaria ainda uma parte teórica em formato de hipertexto onde poder-se-ia navegar pela teoria das equações diferenciais parciais e dos métodos.

O sistema é composto dos seguintes módulos:

a) Módulo Teórico

Neste módulo o usuário terá acesso à teoria de métodos numéricos aplicados a problema de valor de contorno (PVC) em equações diferenciais parciais (EDP's), desenvolvido em html.

b) Módulo Prático

Este aplicativo foi desenvolvido em Borland C++ Builder 5.0 e nele o usuário poderá testar vários métodos numéricos para resolução de EDP's, podendo:

- Escolher o problema;
- Escolher o método;
- Fornecer os valores iniciais;
- Fornecer o incremento;
- Fornecer o final do intervalo desejado na variável evolutiva;

- Visualizar o resultado em gráficos comparativos da solução aproximada com a solução exata.

Nos capítulos a seguir mostraremos como foi construído o *Methods II*, usando conceitos aprendidos nas disciplinas Engenharia de Software I e II. No capítulo 3 será descrita a funcionalidade do aplicativo, mostrando sua utilização com exemplos de dois problemas típicos. Apresentaremos ainda, um resumo dos problemas abordados no *Methods II* assim como os métodos numéricos usados para obter aproximações de suas soluções. Enfim, no capítulo 6, serão apresentadas algumas considerações sobre o *Methods II* e algumas propostas para sua expansão e aprimoramento.

2. A Construção do Sistema

No processo de desenvolvimento de software é fundamental o entendimento dos requisitos do sistema. Os modelos construídos durante essa fase proverão as bases para o projeto e a implementação. Neste capítulo são apresentados os modelos de representação completa do sistema de um ponto de vista particular (isto é, uma agregação de um conjunto de visões de perspectivas específicas), usando notações da *Unified Modeling Language* (UML), úteis para realizar a atividade de análise de sistemas de informação. Apresentamos a seguir os diagramas da UML que foram úteis para obtenção de requisitos: diagrama de casos de uso e diagrama de seqüência.

2.1 Diagrama de Caso de uso

O diagrama de caso de uso especifica a funcionalidade que o sistema oferece do ponto de vista dos usuários. Este modelo utiliza atores para representar papéis desempenhados pelos usuários e casos de uso para representar o que os usuários podem realizar.

Cada caso de uso corresponde a um conjunto de eventos do sistema vistos da perspectiva do usuário e para cada caso de uso existe de forma geral um ator que requer algo do sistema.

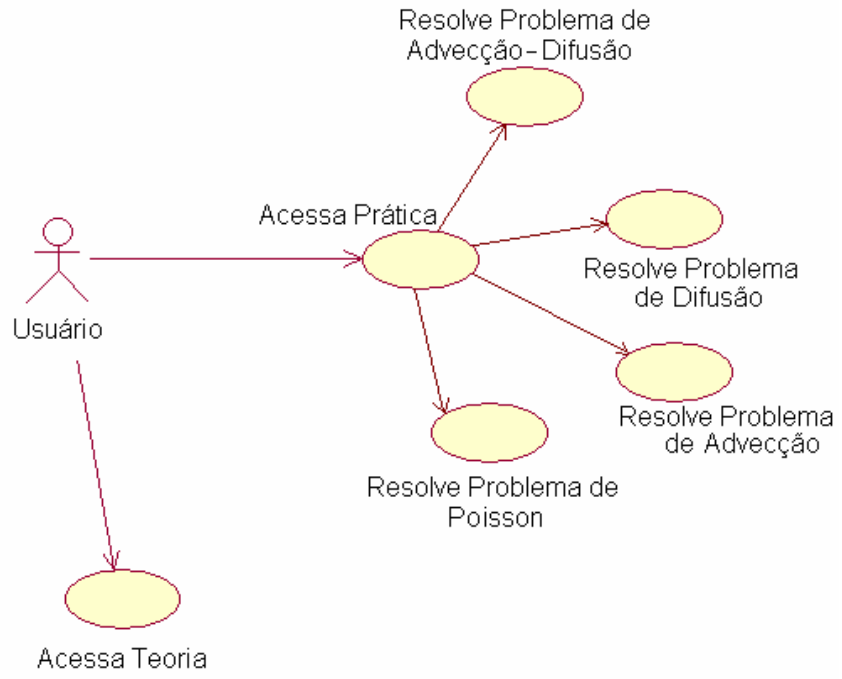


Figura 1: Diagrama de casos de uso

A seguir descrevemos cada um dos casos de uso que aparece na Figura 1.

Usuário acessa Teoria: O usuário pode acessar a teoria a partir do menu principal. A teoria contém um resumo de alguns métodos numéricos aplicados às equações diferenciais parciais.

Usuário acessa Prática: O usuário pode acessar a parte prática a partir do menu principal. Ao acessar a parte prática o usuário é levado ao Menu de Problemas. No módulo prático, o usuário pode escolher qual problema quer resolver, e também pode escolher o método que quer utilizar para resolver este problema.

Usuário resolve Problema de Advecção-Difusão: A partir do Menu de Problemas, escolhendo o problema de Advecção-Difusão, o usuário é levado à escolha do método. Os métodos disponíveis são: Explícito, Implícito, Explícito-upwind e Implícito-Upwind. Após escolher o método, o usuário é levado à tela na qual deve entrar com os valores desejados para a resolução do problema. Ao digitar os dados de entrada e clicar no botão Calcular, o gráfico contendo a solução exata e a solução aproximada aparece mostrando resultado do problema.

Usuário resolve Problema de Difusão: A partir do Menu de Problemas, escolhendo o problema de Difusão, o usuário é levado para a escolha do método desejado. Estão disponíveis os métodos: Explícito, Implícito e Crank-Nicolson. Escolhidos o problema e o método, o usuário é levado para uma tela onde deverá entrar com os dados. Assim, após todos os requisitos atendidos, o usuário terá como resultado um gráfico que mostra o comportamento da solução para o método escolhido por ele.

Usuário resolve Problema de Advecção: A partir do Menu de Problemas, escolhendo o problema de Advecção, o usuário é levado para a escolha do método. Os métodos disponíveis são os mesmos do Problema de Advecção-Difusão. Após escolher o método, o usuário é levado à tela na qual deve entrar com os valores desejados para a resolução do problema. Um gráfico contendo a solução exata e a solução aproximada aparece mostrando resultado do problema.

Usuário resolve Problema de Poisson: Após escolher o problema de Poisson, o usuário é levado à tela de entrada de dados para a resolução do problema. Nesta tela ele pode ver o método utilizado e a solução exata para a equação escolhida. Os resultados da solução aproximada e da solução exata aparecerão nos espaços destinados aos gráficos.

2.2 Diagrama de seqüência

O Diagrama de seqüência é um dos diagramas de interações da UML. Diagramas de interação têm como objetivo ilustrar como os objetos interagem através de mensagens para cumprir tarefas.

As figuras 2 a 7 apresentam os cenários dos casos de uso citados na seção 2.1. Elas representam as interações entre o usuário e o *Methods II*.

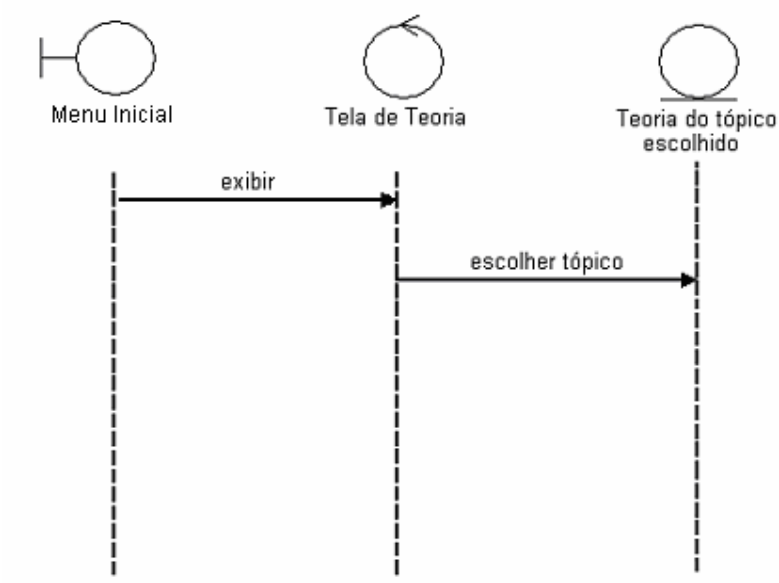


Figura 2: Usuário acessa Teoria

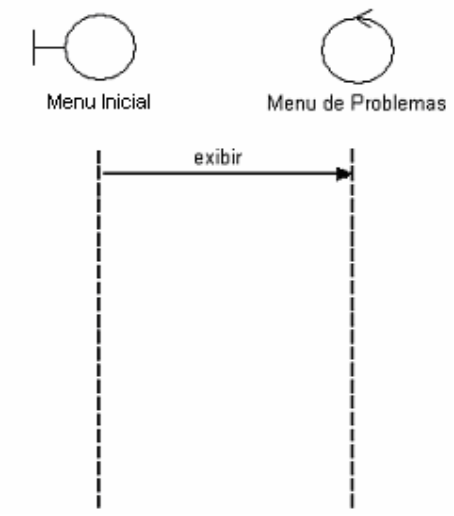


Figura 3: Usuário acessa Prática

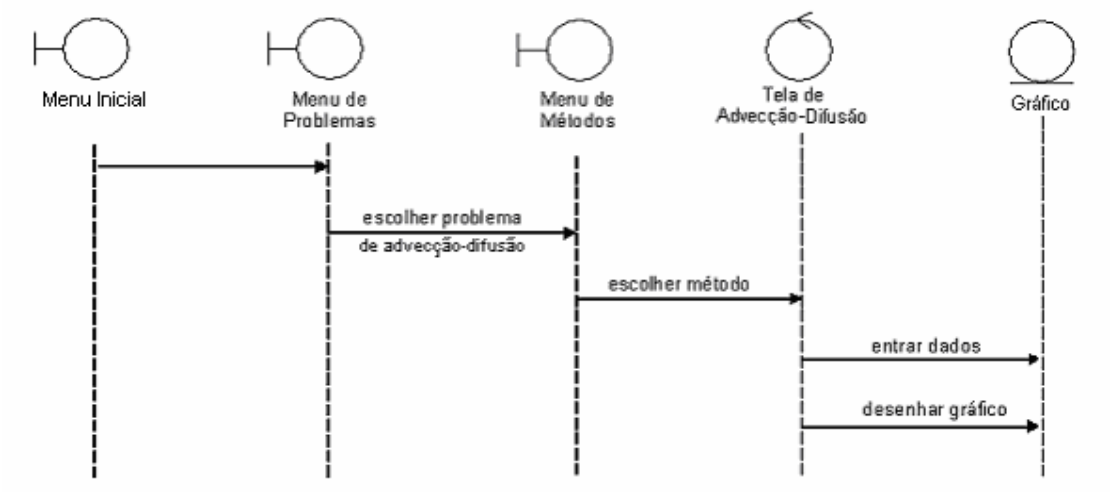


Figura 4: Usuário resolve Problema de Advecção-Difusão

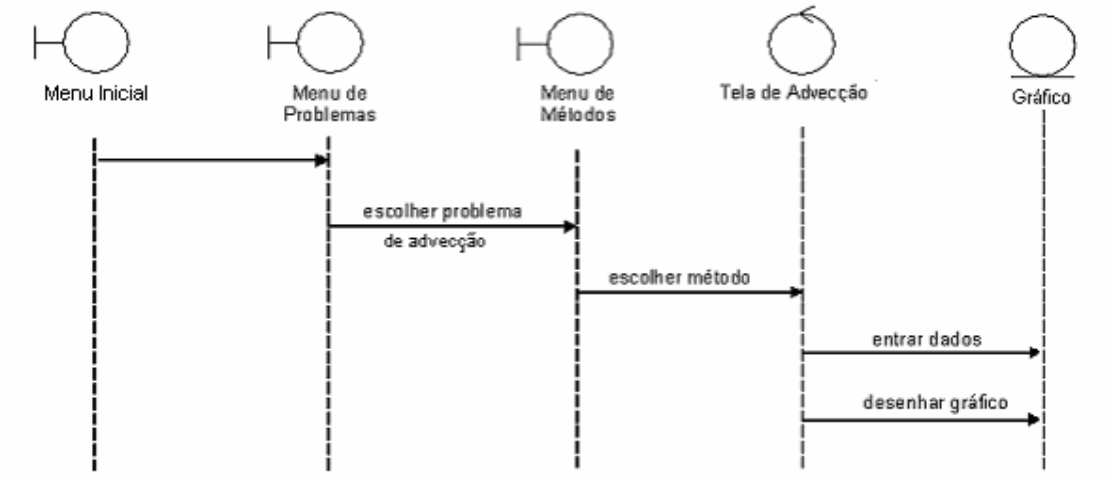


Figura 5: Usuário resolve equação de Advecção

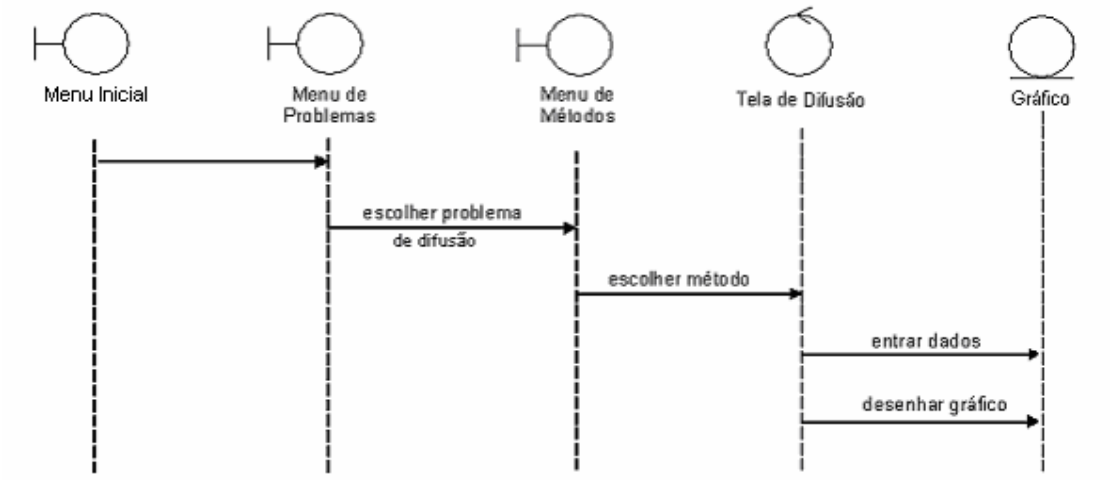


Figura 6: Usuário resolve Problema de Difusão

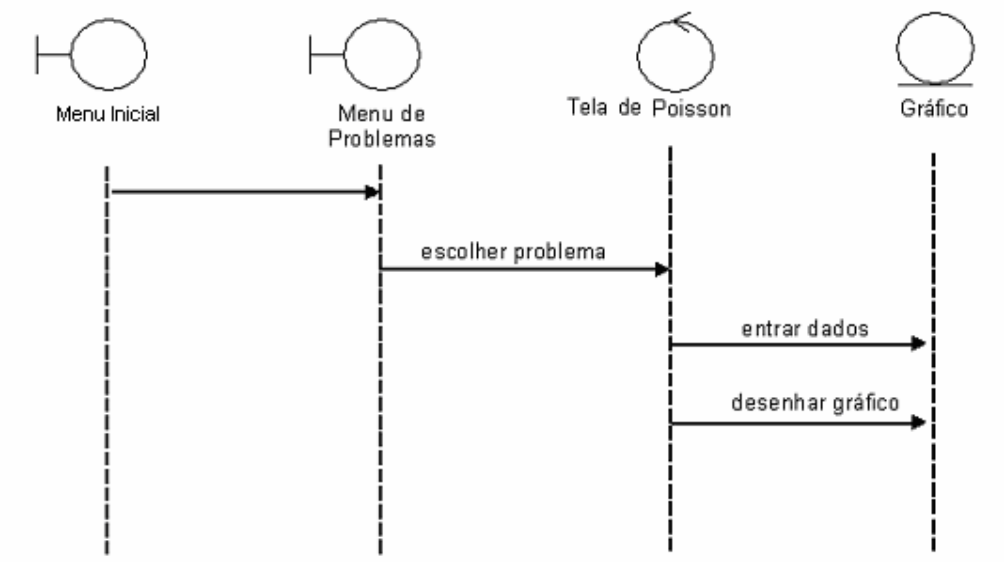


Figura 7: Usuário resolve equação de Poisson

2.3 A escolha das ferramentas

2.3.1 Borland C++ Builder

Para desenvolver o *Methods II*, tivemos que codificar os métodos, as equações e as operações, além de desenvolver uma interface gráfica que permitisse a visualização do gráfico, ou seja, além de programar a parte que envolvia matemática do sistema, tínhamos que construir uma interface agradável ao usuário.

Para escolher a linguagem, nos baseamos no fato de que já tínhamos dois métodos parcialmente codificados em C. Pensando no aproveitamento destes códigos, optamos então por utilizar a linguagem C++, pois nos dá o suporte necessário para uma correta implementação do sistema.

Em outras linguagens, você deve escrever todo o código do seu programa, inclusive para a criação de janelas, botões, etc.. Já em ambientes como o Borland C++ Builder, o procedimento será parte visual e parte com escrita de código. Além disso, com um clique em um objeto (por exemplo, um botão) será criada toda a interface com o usuário e o sistema irá elaborar parte do código, facilitando assim o programador e diminuindo consideravelmente o

tempo de desenvolvimento de um programa. Além disso, para fazer uma interface mais moderna e agradável, precisávamos de uma ferramenta visual e o Borland C++ Builder além de ser uma ferramenta fácil de usar, é bem familiar aos alunos do curso de Ciência da Computação. Uma outra vantagem é que ele permite utilizar o *TeeChart Pro 7*, uma ferramenta para criação de aplicativos gráficos, descrita na próxima seção.

2.3.2 TeeChart Pro 7 VCL/CLX

Para a parte gráfica, foi escolhida uma ferramenta chamada TeeChart Pro 7. Ela oferece diversos estilos de gráficos, funções matemáticas e estatísticas para escolher. Também inclui um poderoso e completo editor para cada componente e sub-componente, que permite que você desenvolva rapidamente complexas aplicações de representação por gráficos. O editor de gráficos é aprimorado pelo componente TeeCommander (uma barra de ferramentas com botões específicos para criação de gráficos), que proporciona acesso aos editores TeeChart e atributos comuns com apenas um clique. Tais editores estão disponíveis tanto na fase de desenvolvimento quanto na fase de execução.

Esta ferramenta possui versões licenciadas para Borland Delphi, Borland C++ Builder, Visual Basic e Visual C++. Existe também uma versão similar para as ferramentas.NET e Active X.

2.4 A codificação do Methods II

O *Methods II* foi implementado para ser uma ferramenta interativa e possui 11 módulos. Os quais os módulos uInicio, uEquacao, uMetodo, uSobre e uTeoria são responsáveis pela implementação de suas respectivas telas, tendo sido construído de forma interativa com as facilidades da ferramenta utilizada. Dessa forma o módulo uInicio é responsável pela tela Inicial. O módulo uEquação é responsável pela tela com o menu de problemas. O módulo uMétodo é responsável pela tela com o menu de métodos. O módulo uSobre é responsável pela tela com informações sobre o sistema. O módulo uTeoria é responsável pela tela com teoria sobre alguns métodos numéricos aplicados às equações diferenciais parciais.

Os módulos uGrafico e uLaplace contêm efetivamente a essência do programa, ou seja, neles estão as partes do código onde as equações e os seus métodos de resolução foram implementados.

O módulo uGrafico é o responsável pela implementação dos métodos para resolução das equações de Difusão, Advecção e Advecção-Difusão. Optamos por implementar a Equação de Advecção-Difusão que possui tanto o termo advectivo quanto o termo difusivo, tendo a Equação de Advecção e a Equação de Difusão como casos particulares. Para o caso da equação escolhida ser a Equação de Advecção, o termo difusivo é zerado internamente e para o caso da equação escolhida ser a Equação de Difusão, o termo advectivo é zerado.

Tomamos como exemplo um caso em que o usuário escolhe a Equação de Advecção. Neste caso o termo difusivo será zerado e a parte do código que calculava a Equação de Advecção-Difusão passará a calcular somente a Equação de Advecção. O usuário, por sua vez, na tela de entrada de dados não visualizará o campo para digitar o coeficiente de difusão.

Se o usuário resolve calcular a Equação de Difusão, neste caso, o termo advectivo será zerado. Desta forma, o código calculará a Equação de Difusão e o usuário não receberá o campo velocidade na tela de entrada de dados.

As descrições de cada método estão detalhadas nas seções: 4.2.2 (para o problema de Difusão), 4.4.1 (para equação de Advecção) e 4.5.1 (para equação de Advecção-Difusão).

No caso dos métodos implícitos, há a necessidade de se resolver para cada instante de tempo, um sistema de equações lineares. Nesse casos foi utilizada a fatoração LU[1], onde a decomposição da matriz do sistema no produto das matrizes triangulares L e U são feitas somente uma vez, e o vetor independente, vetor B, atualizado a cada passo.

No módulo uLaplace, somente um método foi implementado: o método das diferenças finitas. Optamos por implementar aproximações para dois problemas que o usuário escolhe na tela correspondente. Aqui também o sistema foi resolvido pela fatoração LU.

Maiores detalhes podem ser encontrados no código fonte que é apresentado no apêndice A ou no CD de instalação do *Methods II*.

[1] Seja o sistema linear A.x = b, o processo de fatoração LU para resolução deste sistema consiste em decompor a matriz A dos coeficientes em um produto de 2 ou mais fatores, e em seguida, resolver uma seqüência de sistemas lineares que nos conduzirá a solução do sistema linear original.

3. O *Methods II*

3.1 O Sistema

Como citado na introdução, o *Methods II* é uma ferramenta para auxiliar o aprendizado e a visualização da resolução de algumas Equações Diferenciais Parciais básicas utilizando alguns Métodos Numéricos usuais. Estes Métodos são ensinados na disciplina Métodos Numéricos II, obrigatória para o curso de graduação em Ciência da Computação e para o curso de Física da Universidade Federal Fluminense, e optativa para os cursos de Matemática e Engenharia.

O sistema foi desenvolvido em duas etapas. Na primeira, dois métodos já estavam parcialmente implementados. Estes métodos foram terminados, aprimorados e acrescentados ao sistema. Na segunda parte, outros métodos foram implementados e testados. Adicionalmente, foi criada uma interface gráfica que facilita a utilização do aplicativo e a visualização dos resultados.

A seguir será descrita a funcionalidade do *Methods II*, usando dois exemplos.

O sistema é composto de dois módulos: Módulo Teórico e Módulo Prático, como mostra a figura 5, que ilustra a tela inicial do *Methods II*.

Figura 8: Tela inicial

3.1.1 Módulo Teórico

O módulo teórico é constituído de páginas HTML compiladas que contêm um resumo dos principais métodos numéricos aplicados às equações diferenciais parciais.

Esta parte teórica pode ser acessada a qualquer momento pelo usuário no Menu Inicial do sistema. Neste menu o usuário encontra um botão rotulado Teoria. Ao clicar no botão Teoria o usuário recebe uma tela com o conteúdo da teoria, tela esta que é mostrada na figura 6.

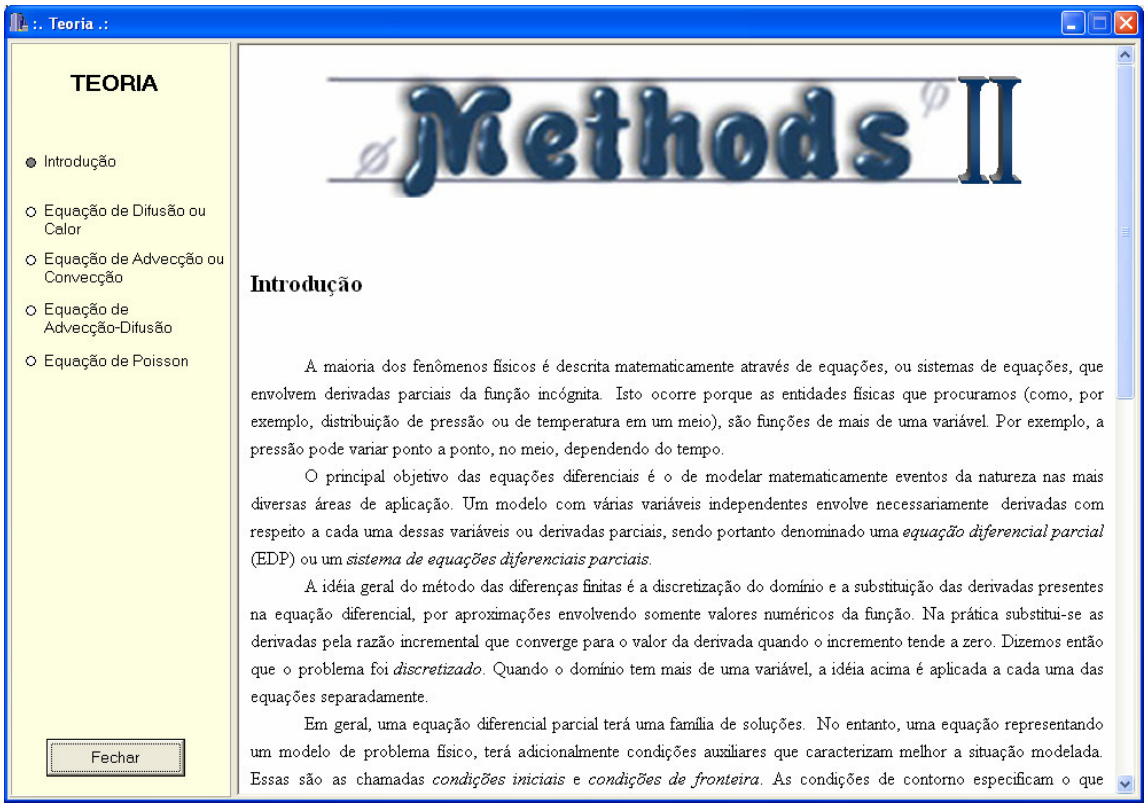


Figura 9: Tela de Teoria

Ao abrir esta tela de teoria, o usuário perceberá que ela é dividida em duas regiões. No lado esquerdo ele encontra um índice dos determinados assuntos detalhados nessa parte teórica. Os títulos dos assuntos são *links* onde o usuário pode clicar e receber o conteúdo sobre o assunto sobre o qual ele clicou. O conteúdo aparece na outra região, ao lado direito dos índices.

3.1.2. Módulo Prático

A parte prática é composta de módulos responsáveis pela resolução das equações diferenciais parciais usando os métodos numéricos. Os resultados são visualizados através de gráficos.

No módulo prático, o usuário pode escolher qual problema quer resolver, e também pode escolher o método que quer utilizar para resolver este problema. Escolhidos o problema e o método, o usuário é levado para uma tela onde deverá entrar com os dados. Assim, após todos os requisitos atendidos, o usuário terá como resultado um gráfico que mostra o comportamento da solução com o método escolhido por ele. A seqüência de telas a seguir (figuras 7 a 11) mostra como o usuário utiliza o módulo prático.

Primeiramente, na tela inicial do *Methods II*, o usuário deve clicar no botão rotulado Prática.

Figura 10: Entrando na parte prática

A seguir, o usuário é levado à tela onde deve escolher qual problema quer resolver. No *Methods II*, o usuário tem quatro opções de problemas para escolher: Equação de Difusão ou Calor, Equação de Advecção ou Convecção, Equação de Advecção-Difusão e Equação de Poisson. Clicando no botão referente à opção escolhida, o usuário é levado para a escolha do método desejado caso tenha escolhido uma das três primeiras equações, ou para a tela de

entrada de dados caso tenha escolhido a Equação de Poisson. A figura 8 mostra esta tela com as opções dos problemas.

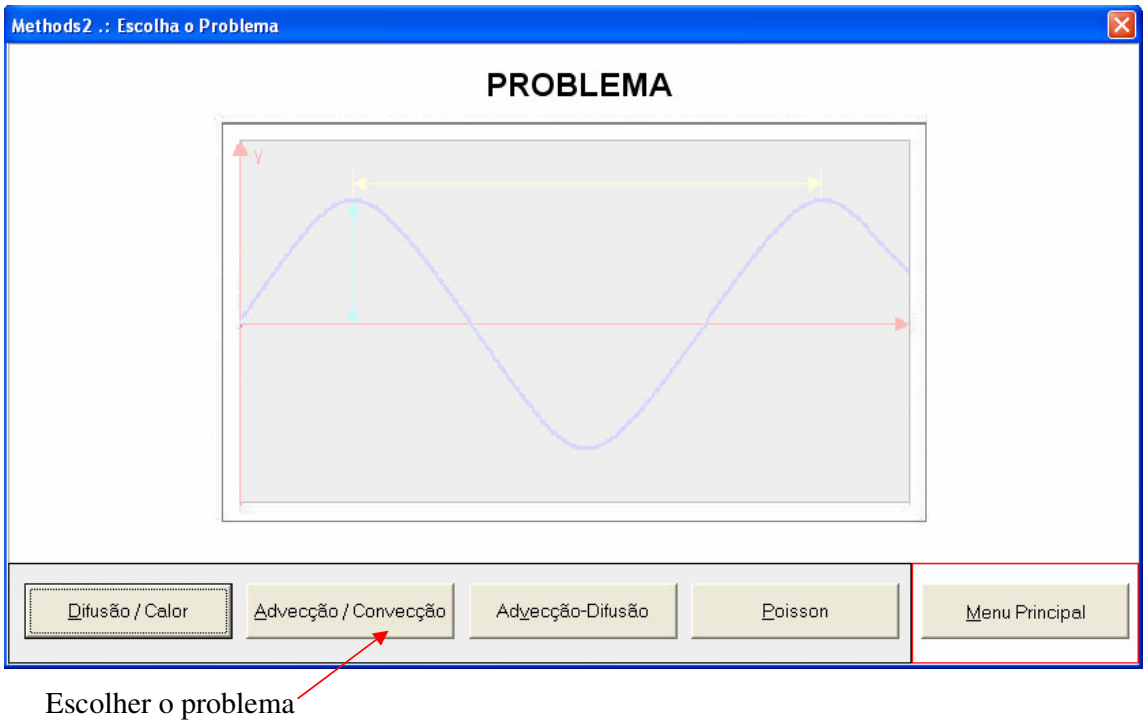


Figura 11: Escolhendo o Problema

Se a opção escolhida foi Equação de Difusão ou Calor unidimensional, Equação de Advecção ou Convecção, ou Equação de Advecção-Difusão, o usuário é levado para uma tela onde pode ver um resumo dos métodos que poderão ser utilizados na resolução do problema, bem como as condições iniciais e de contorno da equação escolhida.

Caso a opção escolhida seja a Equação de Poisson, o usuário é levado diretamente para a tela onde deve entrar com os dados do problema. Nesta página, o usuário também encontra um resumo que mostra a Equação de Poisson, o método utilizado, a função utilizada e a solução exata.

As figuras 9, 10 e 11 a seguir, mostram um exemplo do uso do sistema para resolver o problema de Advecção ou Convecção utilizando o método implícito-*upwind*. As figuras 12 e 13 mostram um exemplo do uso do sistema para resolver o problema da Equação de Poisson.

3.1.2.1 Exemplo utilizando problema de Advecção ou Convecção

Repare na figura 9, que além dos botões onde o usuário escolhe o método que quer utilizar, o usuário também pode visualizar a equação escolhida, as condições iniciais e de contorno do problema, e os métodos que pode escolher.

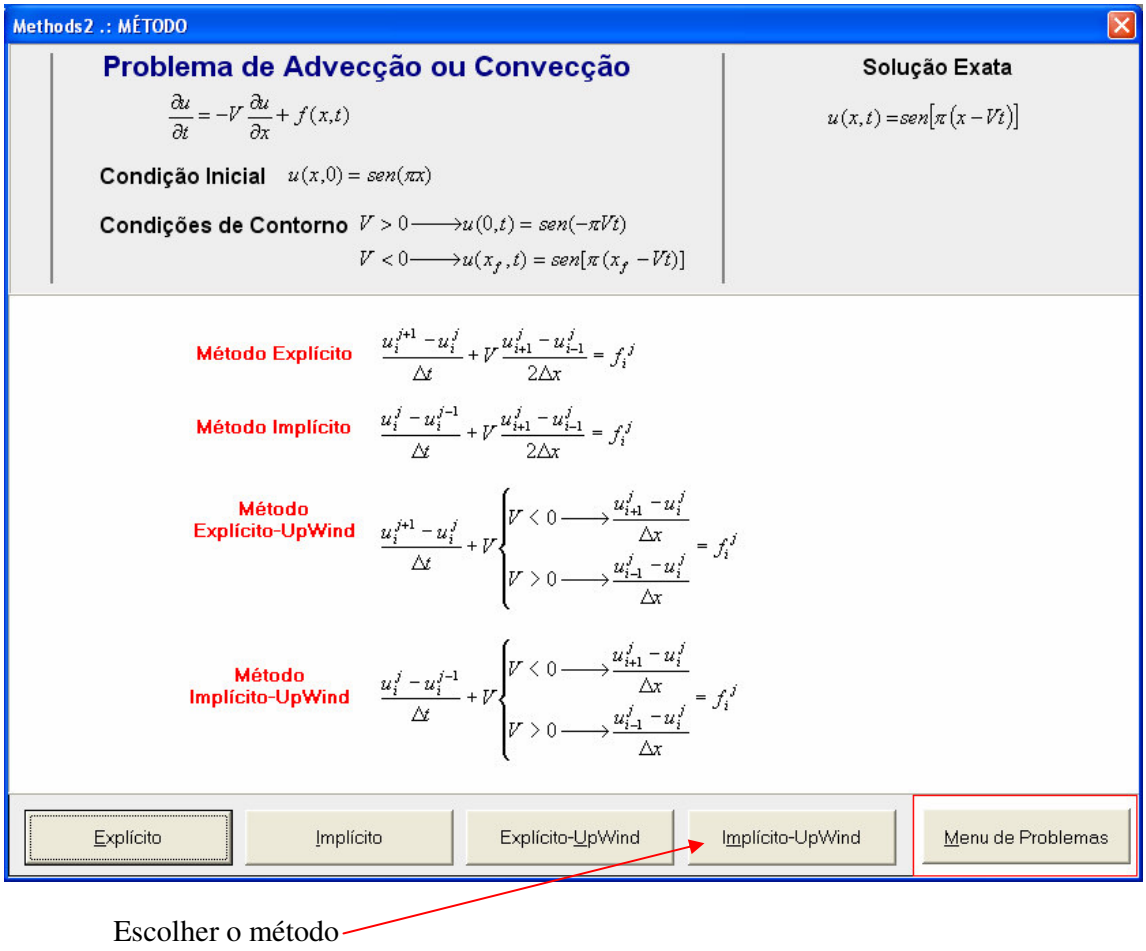


Figura 12: Escolhendo o método

Após escolher o método, o usuário é levado à tela mostrada na figura 10. Nela, é possível que o usuário entre com os valores desejados para a resolução do problema.

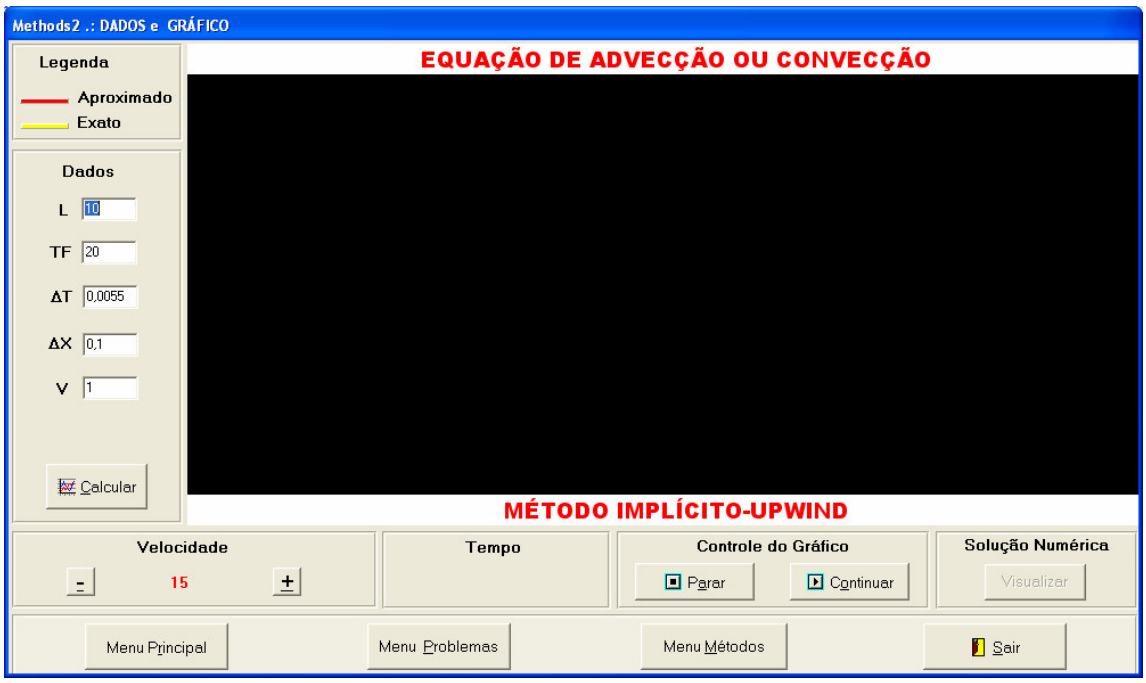


Figura 13: Tela do gráfico

Ao digitar os dados de entrada e clicar no botão Calcular, na região escura aparece o gráfico mostrando resultado do problema, como mostra a figura 11.

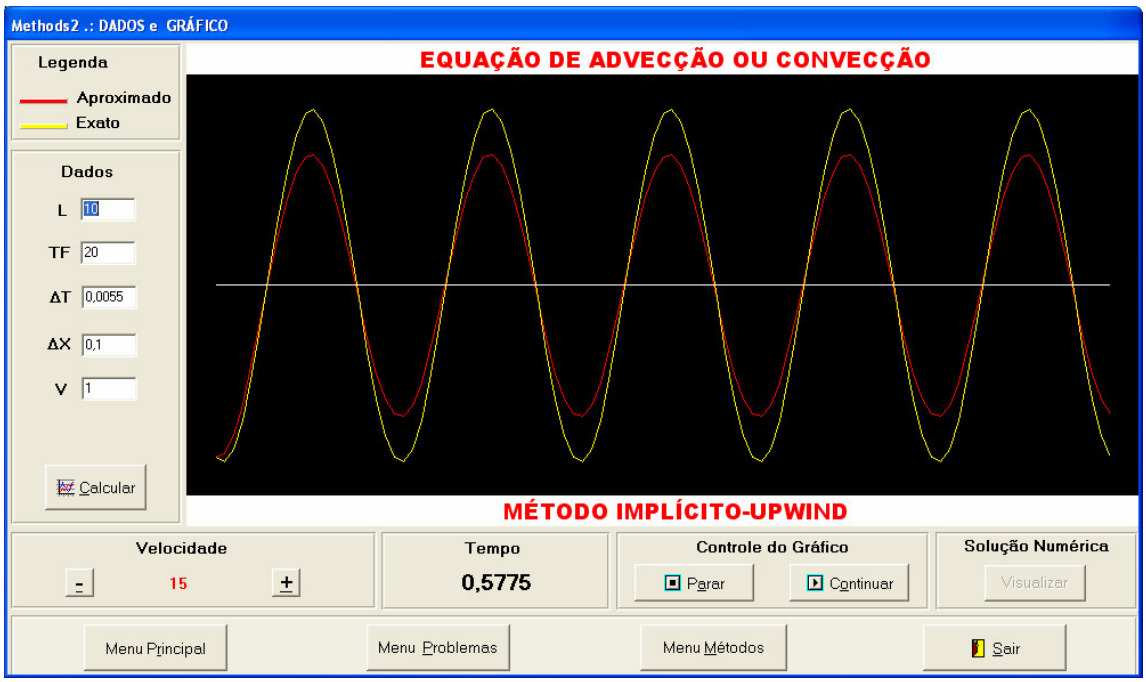


Figura 14: Gráfico em movimento

A legenda indica que a curva na cor amarela mostra a solução exata do problema, e a curva na cor vermelha mostra a solução do método escolhido.

Nesta tela, o usuário ainda pode verificar o andamento do problema, bem como controlar a evolução da solução parando-o para uma análise mais cuidadosa.

3.1.2.2 Exemplo utilizando o problema de Poisson

Quando o usuário, na tela de seleção de problemas, escolhe a equação de Poisson, ele é levado para tela onde deve escolher uma função e entrar com os dados para a resolução do problema. Nesta tela ele ainda pode ver o método utilizado e a solução exata para a equação escolhida. A figura 12 ilustra esta tela.

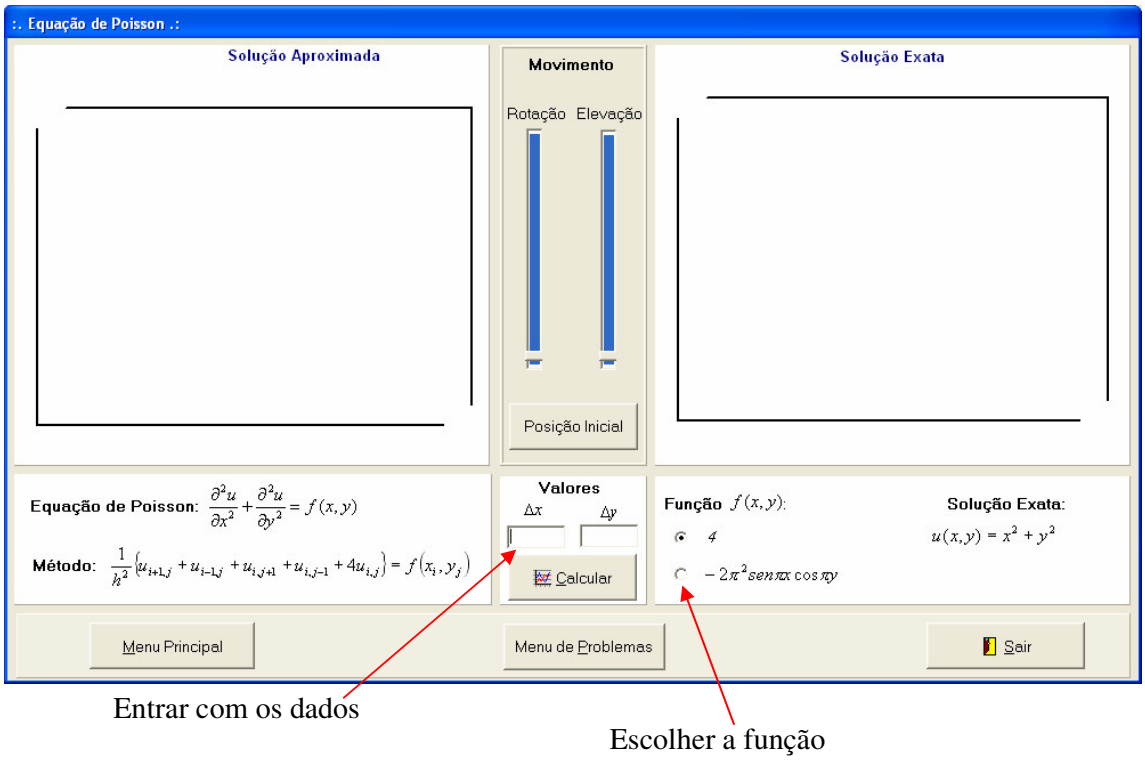


Figura 15: Inserindo dados e escolhendo a equação

Após escolher a função e digitar os dados de entrada, o usuário deve clicar no botão Calcular. Isto feito, os resultados da solução aproximada e da solução exata aparecerão nos espaços destinados aos gráficos, como mostra a figura 13.

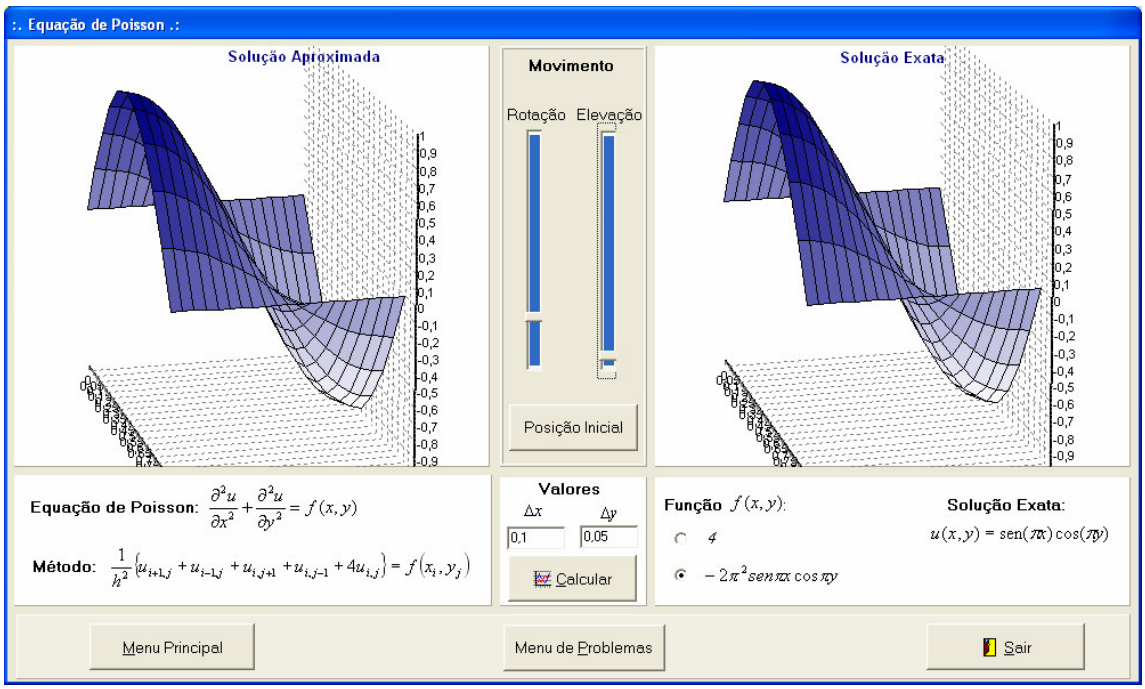


Figura 16: Gráfico da Equação de Poisson

Nesta tela, o usuário ainda pode movimentar o gráfico de modo a deixá-lo numa posição que ele possa visualizá-lo melhor. Existem duas barras que controlam este movimento. Uma controla a rotação horizontal e a outra controla a rotação vertical do gráfico. Clicando no botão rotulado Posição Inicial, o gráfico volta para a posição inicial.

3.2 Requisitos de instalação

A seguinte configuração é necessária para instalar o *Methods II*:

Mínimo:

Processador: Pentium II

Memória: 64 MB

Espaço em disco: 10 MB livres

Recomendado:

Processador: Pentium III ou superior

Memória: 128 MB ou superior

Espaço em disco: 10 MB livres

Softwares:

Não é necessário ter nenhum software específico instalado no computador.

Passos da instalação:

A seguir são mostrados os passos necessários para instalar o *Methods II*:

1. Insira o CD com a instalação do *Methods II* e abra a pasta Instalação.
2. Execute o arquivo SETUP.EXE.
3. Escolha o folder onde os arquivos serão instalados e pressione **Next**.
4. Confirme pressionando **Next** ou pressione **Back** para voltar e escolher outro folder.
5. Os arquivos serão copiados. Pressione **Finish** para terminar a instalação. Marque a opção "Yes, Launch the program file" se você se deseja executar o *Methods II* imediatamente.

4. O Módulo Teórico

4.1 Uma Introdução

A maioria dos fenômenos físicos é descrita matematicamente através de equações, ou sistemas de equações, que envolvem derivadas parciais da função incógnita. Isto ocorre porque as entidades físicas que procuramos (como, por exemplo, distribuição de pressão ou de temperatura em um meio), são funções de mais de uma variável. Por exemplo, a pressão pode variar ponto a ponto, no meio, dependendo do tempo.

O principal objetivo das equações diferenciais é o de modelar matematicamente eventos da natureza nas mais diversas áreas de aplicação. Um modelo com várias variáveis independentes envolve necessariamente derivadas com respeito a cada uma dessas variáveis ou derivadas parciais, sendo portanto denominado uma *equação diferencial parcial* (EDP) ou um *sistema de equações diferenciais parciais*.

A idéia geral do método das diferenças finitas é a discretização do domínio e a substituição das derivadas presentes na equação diferencial, por aproximações envolvendo somente valores numéricos da função. Na prática substituem-se as derivadas pela razão incremental que converge para o valor da derivada quando o incremento tende a zero. Dizemos então que o problema foi *discretizado*. Quando o domínio tem mais de uma variável, a idéia acima é aplicada a cada uma das equações separadamente.

Em geral, uma equação diferencial parcial terá uma família de soluções. No entanto, uma equação representando um modelo de problema físico, terá adicionalmente condições auxiliares que caracterizam melhor a situação modelada. Essas são as chamadas *condições iniciais* e *condições de fronteira*. As condições de contorno especificam o que acontece no limiar da região de definição, enquanto que as condições iniciais fornecem informações sobre o estado inicial do sistema, ou a partir de onde e com qual valor a solução vai se propagar. Quando a região é fechada e as condições de fronteira são fixadas em todo o entorno da região, usa-se a expressão *condição de contorno*.

4.1.1 Classificação

Para efeito de classificação das equações diferenciais parciais lineares de segunda ordem podemos usar a representação geral:

$$au_{xx}+2bu_{xy}+cu_{yy}+du_x+eu_y+fu=g$$

onde *a, b, c, d, e, f* e *g* são constantes ou funções conhecidas das variáveis independentes *x* e *y*.

Podemos classificar uma EDP da seguinte forma:

- Elíptica se $b^2-4ac<0$
- Parabólica se $b^2-4ac=0$
- Hiperbólica se $b^2-4ac>0$

Donde temos os exemplos clássicos representados por:

- Parabólica: equação do calor $u_t - u_{xx} = 0$
- Elíptica: equação de Laplace $u_{xx} + u_{yy} = 0$
- Hiperbólica: equação de advecção $u_t + au_x = 0$

4.1.2 Discretização

A aproximação numérica da solução de uma equação de derivadas parciais é obtida pela transformação do problema contínuo num problema discreto finito. No caso das equações diferenciais parciais, o problema tem a dificuldade adicional do número de variáveis independentes ser maior que um, isto é, o domínio deixa de ser um intervalo e passa a ser uma região do plano ou do espaço. A transformação acima referida é realizada tanto na função incógnita e na equação quanto no domínio de solução. No domínio ela é obtida através da subdivisão deste em um conjunto de pontos (discretização), em geram igualmente espaçados, ao qual damos o nome de *malha*. As transformações da função incógnita e da equação são obtidas, respectivamente pela avaliação desta nos pontos da malha e pela aproximação das derivadas por diferenças finitas.

4.2 Equações Parabólicas: equação do Calor

As equações parabólicas são equações de evolução, ou seja, elas modelam fenômenos que evoluem (de maneira geral, irreversivelmente) com o tempo, assim uma das variáveis tem sempre um caráter temporal que as distingue das demais. Esse aspecto serve para, no aspecto bidimensional, usarmos a variável *x* com o sentido espacial e *t* como variável evolutiva.

Como vimos, se $b^2-4ac=0$ na equação:

$$au_{xx} + 2bu_{xt} + cu_{tt} = r(x,t,u,u_x,u_t)$$

temos uma equação diferencial parabólica, cuja solução é *u(x, t)*.

O modelo fundamental da equação parabólica é a equação do calor:

$$u_t - a(x,t)u_{xx} = r(x,t), \qquad a(x,t) > 0, \qquad 0 \le x \le L, \qquad 0 < t < T \qquad \textbf{(1)}$$

$$u(x,0)=\psi, \qquad 0 \le x \le L \qquad \text{condição inicial}$$

$$\left.\begin{array}{l} u(0,t)=f(t), \quad 0 < t < T \\ u(L,t)=g(t), \quad 0 < t < T \end{array}\right\} \text{ condições de fronteira}$$

4.2.1 Discretização

Vamos usar o modelo da equação do calor, apresentado em **(1)**, com *a(x, t)* constante e *r(x,t) = 0*.

$$u_t = au_{xx}, \qquad a(x,t) > 0, \qquad 0 \le x \le L, \qquad 0 < t < T \qquad \textbf{(2)}$$

$$u(x,0)=\psi, \qquad 0 \le x \le L$$

$$u(0,t)=f(t), \qquad 0 < t < T$$

$$u(L,t)=g(t), \qquad 0 < t < T$$

Dividindo o intervalo [0,L], da variável espacial *x*, em *N* partes iguais de comprimento *h*, temos os *N+1* pontos $x_i=ih$, *i*=0, 1,...,N, onde $h = L/N$ e, dividindo o intervalo [0,T], da variável de tempo *t*, em *M* partes iguais de comprimento *k*, temos os *M+1* pontos $t_j=jk$, *j*=0, 1,...M, onde $k = T/M$. Assim vamos obter uma aproximação da solução nos pontos (x_i, t_j) da malha.

A solução exata será denotada por u_{ij} no ponto (x_i, t_j)e o valor aproximado de u_{ij} por U_{ij}.

4.2.2 Método das Diferenças Finitas

4.2.2.1 Método Explícito

Usando diferenças centradas de segunda ordem na variável espacial para aproximar a derivada de segunda ordem obtemos:

$$u_{xx}(x_i,t_j) \approx \frac{U_{i-1,j} - 2U_{ij} + U_{i+1,j}}{h^2}$$

e usando diferenças progressivas finitas para aproximar a derivada de primeira ordem produzimos a equação:

$$u_t(x_i,t_j) \approx \frac{U_{i,j+1} - U_{ij}}{k}$$

Substituindo essas aproximações em **(2)**, obtemos a equação aproximada:

$$\frac{U_{i,j+1} - U_{ij}}{k} = a\left(\frac{U_{i-1,j} - 2U_{ij} + U_{i+1,j}}{h^2}\right)$$

ou seja,

$$U_{i,j+1} = U_{i,j} + \sigma(U_{i-1} - 2U_{i,j} + U_{i+1,j}) \text{ , onde } \sigma = {ka}/{h^2} \qquad \textbf{(3)}$$

Assim, conhecidos os valores $U_{i-1,j}$, $U_{i,j}$ e $U_{i+1,j}$ calculamos $U_{i,j+1}$ explicitamente.

Sabemos que:

- *Um método numérico é <u>consistente</u> com relação a uma dada equação se o erro de truncamento local desse método para aquela equação for pelo menos de O(h);*
- *Um método numérico é <u>estável</u> se a equação de diferenças associada não amplifica erros dos passos anteriores;*
- *Um método numérico é <u>convergente</u> num ponto (x, t) do domínio se o erro global $E_{i,j}$ associado a esse ponto tende a zero quando os índices i e j tendem para infinito de maneira que o ponto x = i*h e t = j*k permaneça fixo.*

No método explícito $\sigma \leq \frac{1}{2}$ é a condição para estabilidade, sendo o método dito então condicionalmente estável.

4.2.2.2 Método Implícito

Uma fórmula implícita é aquela em que dois ou mais valores desconhecidos da linha *j* são especificados simultaneamente em termos de valores conhecidos das linhas *j-1, j-2*,... O cálculo explícito de U_{ij} não é possível e usualmente exige-se a solução de um sistema linear.

Aplicando diferenças regressivas do lado esquerdo da equação e diferenças centradas do lado direito, temos:

$$\frac{U_{i,j} - U_{i,j-1}}{k} = a\left(\frac{U_{i-1,j} - 2U_{ij} + U_{i+1,j}}{h^2}\right)$$

que depois de alguma manipulação algébrica pode ser reescrita da seguinte forma:

$$\sigma U_{i-1,j} + (1 + 2\sigma)U_{i,j} - \sigma U_{i+1,j} = U_{i,j-1}, \text{ para } i = 1, 2,..., \text{N-1 e } j = 1, 2, ..., \text{N} \qquad \textbf{(4)}$$

Essas equações reunidas formam um sistema de equações lineares.

Na forma matricial, impostas as condições de contorno, o sistema a ser resolvido, a cada passo de tempo é:

$$\text{A U}_j = U_{j-1} + c_j$$

Onde,

$$A=\begin{bmatrix} (1+2\sigma) & -\sigma & 0 & \cdots & \cdots & 0 \\ -\sigma & (1+2\sigma) & -\sigma & 0 & \cdots & 0 \\ 0 & -\sigma & (1+2\sigma) & -\sigma & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & -\sigma & (1+2\sigma) \end{bmatrix}$$

$$U_j = \begin{bmatrix} U_{1,j} \\ U_{2,j} \\ U_{3,j} \\ \vdots \\ U_{N-1,j} \end{bmatrix} \qquad U_{j-1} = \begin{bmatrix} U_{1,j-1} \\ U_{2,j-1} \\ U_{3,j-1} \\ \vdots \\ U_{N-1,j-1} \end{bmatrix} \qquad c = \begin{bmatrix} \sigma U_{0,j} \\ 0 \\ 0 \\ \vdots \\ \sigma U_{N,j} \end{bmatrix}$$

No *Methods II* os sistemas lineares são resolvidos por fatoração LU.

Para a estabilidade, a condição que precisa ser satisfeita é $\left|e^{\lambda}\right| \leq 1$. Mas, neste caso $\left|e^{\lambda}\right| < 1$ é verdade para todo σ, ou seja, o método é incondicionalmente estável. Algumas vezes, esse fato é usado para justificar a utilização de um método implícito, pois sendo esse método incondicionalmente estável, não devemos nos preocupar com a amplificação de erros e, portanto podemos utilizar uma malha menos fina para obter a solução.

4.2.2.3 Método de Crank-Nicolson

Este é um dos métodos mais utilizados na solução de equações parabólicas. Podemos dizer que esse método é a "média aritmética" dos métodos explícito e implícito. Tomando a média entre as expressões **(3)** e **(4)**, obtemos o método de Crank-Nicolson dado por:

$$U_{i,j+1} = U_{i,j} + \frac{\sigma(U_{i-1,j} + U_{i-1,j+1} - 2(U_{i,j} + U_{i,j+1}) + U_{i+1,j} + U_{i+1,j+1})}{2}$$

ou ainda,

$$\frac{U_{i,j+1} - U_{i,j}}{k} = \frac{a}{2h^2}\left(U_{i-1,j} - 2U_{i,j} + U_{i+1,j} + U_{i-1,j+1} - 2U_{i,j+1} + U_{i+1,j+1}\right)$$

Podemos observar que essas equações formam o seguinte sistema linear $AU_{j+1}=BU_j+c_j$, que é diagonalmente dominante.

$$\begin{pmatrix} 2+2\sigma & -\sigma & 0 & \cdots & 0 \\ -\sigma & 2+2\sigma & -\sigma & \cdots & 0 \\ \vdots & -\sigma & 2+2\sigma & -\sigma & \vdots \\ 0 & \cdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & -\sigma & 2+2\sigma \end{pmatrix} \begin{pmatrix} U_{1,j+1} \\ U_{2,j+1} \\ U_{3,j+1} \\ \vdots \\ U_{N,j+1} \end{pmatrix} =$$

$$\begin{pmatrix} 2+2\sigma & -\sigma & 0 & \cdots & 0 \\ -\sigma & 2+2\sigma & -\sigma & \cdots & 0 \\ \vdots & -\sigma & 2+2\sigma & -\sigma & \vdots \\ 0 & \cdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & -\sigma & 2+2\sigma \end{pmatrix} \begin{pmatrix} U_{1,j} \\ U_{2,j} \\ U_{3,j} \\ \vdots \\ U_{N,j} \end{pmatrix} + \begin{pmatrix} \sigma U_{0,j} \\ 0 \\ 0 \\ \vdots \\ \sigma U_{N,j} \end{pmatrix}$$

Para se obter a solução em cada estágio, é preciso resolver um sistema tridiagonal.

A análise da estabilidade segue o mesmo raciocínio desenvolvido para o método explícito.e concluímos que o método é incondicionalmente estável, pois $\left|e^{\lambda}\right|$ é sempre menor do que 1.

4.3 Equações Elípticas: equação de Poisson

Problemas de equilíbrio em duas ou três dimensões geralmente dão origem a equações elípticas. Exemplos típicos dessa classe são problemas de difusão e de pressão, problemas em elasticidade, problemas de camada limite, problemas de vibração de membranas, etc. Mais simplificadamente, os problemas elípticos caracterizam-se pela propagação de suas propriedades físicas em todas as direções coordenadas indistintamente, ao contrário das equações parabólicas e hiperbólicas onde essas propriedades propagam-se em direções preferenciais. Daí porque as condições de fronteira de um problema elíptico são normalmente especificadas ao longo de toda a fronteira. Sua forma geral é,

$$a(x,t)\frac{\partial^2 u}{\partial x^2} + 2b(x,t)\frac{\partial^2 u}{\partial x \partial y} + c(x,t)\frac{\partial^2 u}{\partial y^2} = d\left(x, y, u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right) \qquad \textbf{(5)}$$

De acordo com a definição apresentada na seção 4.1.1, a equação acima é elíptica em $\Re$ se $b^2 - 4ac < 0$ para todo ponto (x, y) de $\Re$.

Três tipos de problemas distintos envolvendo a equação **(5)** podem ser destacados dependendo das condições de fronteira:

- o problema de Dirichlet, requer que a solução u de **(5)** seja conhecida sobre a fronteira $\partial\Re$, isto é,

$$u(x, y) = f(x, y), \quad (x, y) \in \partial\Re .$$

- quando $\frac{\partial u}{\partial n}$ é conhecida sobre $\partial\Re$, ou seja,

$$\frac{\partial u}{\partial n} = g(x, y), \quad (x, y) \in \partial\Re ,$$

onde n é a normal externa à fronteira $\partial\Re$, o problema de valor de fronteira é conhecido como problema de Neumann.

- O problema de Robbins ou misto, surge quando conhecemos:

$$\alpha(x, y)u + \beta(x, y)\frac{\partial u}{\partial n} = \gamma(x, y) \text{ sobre } \partial\Re ,$$

onde α (x, y) > 0, β (x, y) > 0, (x, y) $\in \partial\Re$.

Quando em **(5)** tomamos a = c ≡ 1 e b = d ≡ 0 obtemos o protótipo de equação elíptica mais conhecido que é a famosa equação de Laplace:

$$-(u_{xx} + u_{yy}) = 0$$

4.3.1 Método das Diferenças Finitas

Os métodos de diferenças finitas, a exemplo do que fizemos na seção 4.2.2 para as equações parabólicas, consistem em substituir as derivadas parciais presentes na equação diferencial por aproximações por diferenças finitas. Para isto é necessário que os pontos onde essas diferenças serão calculadas sejam pré estabelecidos, ou seja, é necessário a definição de uma malha de pontos no domínio. Para ilustrar como esta discretização é realizada na prática consideremos a equação de Poisson:

$$-\Delta u = -(u_{xx} + u_{yy}) = f, \qquad \textbf{(6)}$$

definida no retângulo $R = \{(x, y), 0 \le x \le a, 0 \le y \le b\}$ com condição de Dirichlet:

$$u = g(x, y)$$

sobre a fronteira, $\partial\Re$ desse retângulo.

Primeiramente, para que possamos aproximar u_{xx} e u_{yy} por diferenças finitas cobrimos a região R com uma malha. Escolhemos a opção mais óbvia que consiste em traçar linhas paralelas aos eixos coordenados. Os pontos dessa malha serão denotados por (x_i, y_j), $x_i = ih$, $y_j = jk$, i =0, 1,..., M, j = 0, 1,..., N, onde h representa o espaçamento na direção x e k na direção y, M é o número de subdivisões na direção x e N na direção y. Denotamos por R_δ os pontos da malha interiores a R, isto é,

$$R_\delta = \{(x_i, y_j),\ 0 < i < M,\ 0 < j < N\}$$

e por $\partial\Re_\delta$ os pontos da malha que estão sobre a fronteira ou seja,

$$\partial\Re_\delta = \{(x_i, y_j),\ (i = 0,M,\ 0 \le j \le N) \text{ e } (0 \le i \le M,\ j = 0,N)\}$$

Podemos agora aproximar as derivadas da equação **(6)** da seguinte forma: A equação **(6)** é válida para todos os pontos de R então em particular para um ponto genérico de R_δ podemos escrever,

$$-(u_{xx}(x_i, y_j) + u_{yy}(x_i, y_j)) = f(x_i, y_j)$$

Assim, as derivadas podem ser aproximadas por:

$$u_{xx}(x_i, y_i) \approx \frac{u(x_i + h, y_j) - 2u(x_i, y_j) + u(x_i - h, y_j)}{h^2}$$

$$u_{yy}(x_i, y_j) \approx \frac{u(x_i, y_j + k) - 2u(x_i, y_j) + u(x_i, y_j - k)}{k^2}$$

Substituindo essas aproximações em **(6)** obtemos:

$$-\left(\frac{u(x_i + h, y_j) - 2u(x_i, y_j) + u(x_i - h, y_j)}{h^2} + \frac{u(x_i, y_j + k) - 2u(x_i, y_j) + u(x_i, y_j - k)}{k^2}\right) \approx f(x_i, y_j)$$

Denotaremos por $u_{i,j}$ o valor da solução no ponto (x_i, y_j) e por $U_{i,j}$ a solução da equação de diferenças:

$$-\left(\frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{h^2} + \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{k^2}\right) = f_{i,j}$$

A equação **(6)** deverá ser satisfeita para todos os pontos em R_δ. Para os pontos em $\partial\Re_\delta$ calculamos $U_{i,j}$ da condição de fronteira de Dirichlet

$$U_{i,j} = g(x_i, y_j).$$

Em notações matriciais, seguindo a convenção notacional adotada na seção 4.2, podemos escrever esse sistema como:

$$AU = c$$

onde o vetor U, a matriz A e o vetor c são dados respectivamente por:

$$U = \left(U_{1,1}, U_{2,1}, \ldots, U_{N-1,1}, U_{1,2}, U_{2,2}, \ldots, U_{N-1,2}, \ldots, U_{1,M-1}, U_{2,M-1}, \ldots, U_{N-1,M-1}\right)^T$$

$$A = \begin{pmatrix} a & b & & & & c & & & 0 \\ b & a & b & & & & \ddots & & \\ & \ddots & \ddots & \ddots & & 0 & & \ddots & \\ & & \ddots & \ddots & \ddots & & & & c \\ & & & \ddots & \ddots & \ddots & & & \\ c & & & & \ddots & \ddots & \ddots & & \\ & \ddots & & 0 & & \ddots & \ddots & \ddots & \\ & & \ddots & & & & b & a & b \\ 0 & & & c & & & & b & a \end{pmatrix}$$

$$c = \begin{pmatrix} f(x_1, y_1) + \dfrac{g(x_0, y_1)}{h^2} + \dfrac{g(x_1, y_0)}{k^2} \\ f(x_2, y_1) + \dfrac{g(x_2, y_0)}{k^2} \\ \vdots \\ f(x_{N-1}, y_1) + \dfrac{g(x_N, y_1)}{h^2} + \dfrac{g(x_{N-1}, y_0)}{k^2} \end{pmatrix}$$

já consideradas as condições de contorno.

Na matriz A os números *a, b* e *c* são os coeficientes da discretização de cinco pontos e são dados por:

$$a = \frac{2}{h^2} + \frac{2}{k^2}, \qquad c = -\frac{1}{k^2}$$

Já o valor de b não é constante na matriz toda. Em algumas posições esse valor é nulo. Essas posições correspondem àqueles pontos situados sobre a fronteira. Nas demais posições o valor de *b* é $b = -\dfrac{1}{h^2}$

O procedimento geral para encontrar uma solução aproximada da equação de Poisson, inclui os seguintes passos:

A – Estabelecer a malha, definindo os pontos de fronteira.

B – Escolher a discretização do laplaciano, para os pontos que não sejam da fronteira. Podemos usar uma das expressões da seção anterior.

C – Montar o sistema de equações lineares usando a discretização nos pontos interiores, $f(x_i, y_i)$, e as condições de contorno nos pontos da fronteira.

D – Resolver o sistema, usando algum método numérico para obter $u_{ij} \approx u(x_i, y_j)$, que são aproximações nos pontos interiores.

4.4 Equações Hiperbólicas: equação de Advecção

Ao contrário das equações parabólicas, onde, em geral, a solução é mais suave do que os dados do problema, nas equações hiperbólicas, as descontinuidades são transportadas sem suavização, podendo haver também a formação de singularidades (choques), mesmo no caso de dados iniciais bem comportados.

Os modelos mais simples para estudo de equações hiperbólicas são a equação de advecção $u_t + au_x=0$, a equação escalar conservativa $u_t+(f(u))_x=0$ e a equação da onda $u_{tt}+a^2u_{xx}=0$. Iremos nos aprofundar na equação de advecção.

4.4.1 Método das Diferenças Finitas

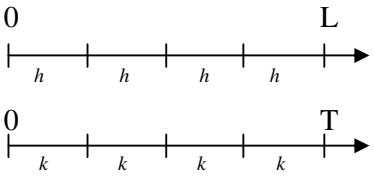
Seja, para $u(x, t)$ a equação de advecção de primeira ordem:

$$u_t + au_x = 0,\ a>0 \text{ (constante)},\ t>0,\ x\in \Re, \qquad \textbf{(7)}$$

juntamente com a condição inicial

$$u(x,0)\ ,\ x\in \Re$$

Considerando a região $t\geq 0$, $-\infty < x < +\infty$, coberta por uma malha uniforme no eixo x e na variável t, isto é, com espaçamento dado respectivamente h e k.



Um ponto genérico pode ser dado pelas coordenadas ih em x e jk em t. Por exemplo, P pode ser dado por $x=ih$ e $t=jk$, ou seja, P = (ih, jk).

4.4.1.1 Método Explícito

A maneira mais intuitiva de resolvermos a equação **(7)** seria usarmos diferenças progressivas no tempo e central no espaço, ou seja,

$$U_{i,j+1} - U_{i,j} = -\frac{ka}{h}\left(U_{i+1,j} - U_{i-1,j}\right)$$

Apesar de um método simples, este é um método inadequado por ser incondicionalmente instável.

4.4.1.2 Método Explícito com aproximação Upwind

Se *a(x, t)>0*, devemos usar diferença progressiva para aproximação da derivada espacial. Se *a(x, t)*<0, a derivada espacial é aproximada usando a fórmula atrasada. O método *upwind* incorpora as duas condições em um único método, de maneira a ser geral o suficiente para resolver problemas par qualquer $a \equiv a(x, t, u)$. O método *upwind* tem a seguinte expressão:

$$\frac{U_{i,j+1} - U_{i,j}}{k} - a_i^j \begin{cases} \dfrac{U_{i,j} - U_{i-1,j}}{\Delta x} & \text{,se } a(x,t) > 0 \\ \dfrac{U_{i,j} - U_{i+1,j}}{\Delta x} & \text{,se } a(x,t) < 0 \end{cases} = f_i^j$$

Comparando com o método explícito, suas expressões diferem apenas no segundo termo do lado esquerdo da igualdade que é uma aproximação para u_x, representando portanto, a resolução numérica do problema. Esse termo de primeira ordem estabiliza o método explícito, mas introduz uma suavização à solução numérica, que uma propriedade marcante dos métodos do tipo *upwind*.

4.4.1.3 Método Implícito

O Método Implícito utiliza a fórmula atrasada para discretizar a derivada com relação o tempo e é definido por:

$$\frac{U_{i,j+1} - U_{i,j}}{k} - a_i^j \frac{U_{i+i,j} - U_{i-1,j}}{2h} = f_i^j$$

Essas equações reunidas formam um sistema de equações lineares.

Representando na forma matricial, impostas as condições de contorno, o sistema a ser resolvido, a cada passo de tempo é:

$$\mathrm{A}\ \mathrm{U}^j = u^{j-1} + kf^j$$

Para a solução do sistema, são encontradas as aproximações no passo t_j, a partir de valores conhecidos no passo anterior. Assim, partindo da condição inicial $u^0(x)$, passo a passo, chegamos na solução em um tempo $t_j = jk$.

O Método Implícito é condicionalmente estável.

4.4.1.4 Método Implícito com aproximação Upwind

Neste esquema, a aproximação do termo da primeira derivada não é feita usando um esquema central, como mostrado na equação **(7)**. No esquema *upwind*, a aproximação do termo de primeira derivada considera a direção da velocidade de advecção *a(x, t)* na equação **(7)**.

Se *a(x, t)* é positiva, o termo de primeira derivada é aproximado por:

$$U_t = \frac{U_i - U_{i-1}}{\Delta x}$$

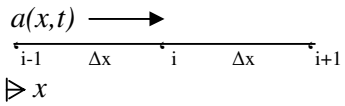


Figura 17 : Esquema *Upwind* para a(x,t)>0

Caso *a(x, t)* seja negativa, o termo da primeira derivada é aproximado por:

$$U_t = \frac{U_i - U_{i+1}}{\Delta x}$$

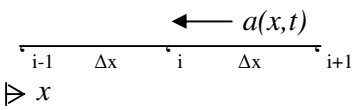


Figura 18: Esquema Upwind para a(x,t)<0

Aplicando o mesmo esquema *upwind* ao termo de primeira derivada do Método Implícito, temos:

$$\frac{U_{i,j} - U_{i,j-1}}{k} - a_i^j \begin{cases} \dfrac{U_{i,j} - U_{i-1,j}}{\Delta x} & ,\text{se } a(x,t) > 0 \\ \dfrac{U_{i,j} - U_{i+1,j}}{\Delta x} & ,\text{se } a(x,t) < 0 \end{cases} = f_{i,j}$$

4.5 Equação de Advecção-Difusão

Uma equação diferencial que considere os efeitos difusivo e advectivo se escreve:

$$\frac{\partial u}{\partial t} = K\frac{\partial^2 u}{\partial x^2} - V\frac{\partial u}{\partial x}$$

A equação acima é conhecida como advecção-difusão. Como os métodos numéricos convenientes para cada um destes casos são distintos, a solução numérica da equação de advecção-difusão é mais elaborada do que a solução numérica da equação de difusão.

Como exemplo de fenômeno físico pode-se citar a temperatura de um meio que é influenciada pela velocidade do ar (vento).

Além do fluxo provocado pela difusão, pode-se também considerar o transporte de substâncias devido à velocidade do meio. Neste caso, um termo advectivo, u_x, também aparece na equação diferencial.

A classificação desse tipo de equação pode ser tanto parabólica, como hiperbólica, dependendo da relação entre os valores dos coeficientes difusivo e advectivo.

4.5.1 Método das Diferenças Finitas

Os métodos que resolvem a equação de Advecção-Difusão são os mesmo usados para resolver problemas de Advecção, ou seja, método explícito, método implícito, método explícito com aproximação *upwind* e método implícito com aproximação *upwind,* representados resumidamente a seguir.

4.5.1.1 Método Explícito

$$\frac{U_{i,j} - U_{i,j-1}}{\Delta t} = K\frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{\Delta x^2} - V\frac{U_{i+1,j} - U_{i-1,j}}{2\Delta x}$$

4.5.1.2 Método Explícito com aproximação *Upwind*

$$\frac{U_{i,j+1} - U_{i,j}}{\Delta t} = K\frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{\Delta x^2} - V\begin{cases}\dfrac{U_{i,j} - U_{i-1,j}}{\Delta x} & \text{,se } V > 0 \\ \dfrac{U_{i,j} - U_{i+1,j}}{\Delta x} & \text{,se } V < 0\end{cases}$$

4.5.1.3 Método Implícito

$$\frac{U_{i,j+1}-U_{i,j}}{\Delta t}=K\frac{U_{i+1,j}-2U_{i,j}+U_{i-1,j}}{\Delta x^2}-V\frac{U_{i+1,j}-U_{i-1,j}}{2\Delta x}$$

4.5.1.4 Método Implícito com aproximação *Upwind*

$$\frac{U_{i,j+1}-U_{i,j}}{\Delta t}=K\frac{U_{i+1,j}-2U_{i,j}+U_{i-1,j}}{\Delta x^2}-V\begin{cases}\dfrac{U_{i,j}-U_{i-1,j}}{\Delta x} & \text{,se } V>0\\ \dfrac{U_{i,j}-U_{i+1,j}}{\Delta x} & \text{,se } V<0\end{cases}$$

5. Conclusão

O sistema *Methods II* é um sistema instrucional de Métodos Numéricos para Equações Diferenciais Parciais - problemas de valor de contorno. O objetivo desse sistema é auxiliar os alunos de cursos de graduação, resolvendo problemas e mostrando graficamente o comportamento dos métodos e suas aproximações para a solução.

Mais do que apresentar demonstrações precisas, pretendemos apresentar os principais conceitos sobre soluções numéricas e sobre a "qualidade dessas aproximações". Através de gráficos, podemos verificar se um determinado método está estável ou instável para determinados dados de entrada que utilizamos.

Daí, tiramos os maiores benefícios do uso do sistema *Methods II* :

- Podemos visualizar com facilidade as soluções aproximadas, bem como as soluções exatas, podendo assim compará-las;
- Estimular o aluno no aprendizado da disciplina Métodos Numéricos II, fornecendo um suporte prático à teoria ensinada em sala de aula.

Podemos sugerir para incrementos futuros, resolver os problemas de Advecção, Difusão e Advecção-Difusão bidimensionalmente. Adicionalmente, seria interessante o acréscimo de um interpretador de funções no módulo que resolve a equação de Poisson. Neste caso, ao invés de escolher uma função pré-determinada, o usuário poderia digitar sua própria função.

REFERÊNCIAS BIBLIOGRÁFICAS

ZUIM, Marcelo, *METHODS - Um sistema interativo de apoio ao aprendizado de métodos numéricos para problemas de valor inicial*, Trabalho de Conclusão de Curso (Graduação em Ciência da Computação), Universidade Federal Fluminense, Niterói, 2000.

ABREU, Estela dos Santos, TEIXEIRA, José Carlos Abreu et al. *Apresentação de trabalhos monográficos de conclusão de curso*. 2º ed., Niterói, EDUFF, 1994.

CUNHA, M. Cristina C., *Métodos Numéricos*, Campinas, Editora da UNICAMP, 2000.

GOLUB, Gene H. e ORTEGA, James M., *Scientific Computing and Differential Equations – Na Introduction to Numerical Methods*, Boston, Academic Press, Inc., 1981

CUMINATO, José Alberto e JUNIOR, Messias Meneguete, *Discretização de Equações Diferenciais Parciais – Técnicas de Diferenças Finitas*, 2002.

SCHILDT, Herbert, *Borland C++ Builder - Referência Completa,* Editora Campus, 2001.

AGUIAR, Teresa Cristina, *Apostila UML na fase de análise de sistemas de informação,* professora do Departamento de Ciência da Computação – UFF, 2002.

STEEMA SOFTWARE S.L., *TeeChart Pro 7*. Uma suite de componentes para geração de Gráficos. Disponível em: <http://www.steema.com>. Acessado em: 10/06/2004

Apêndice A - Código fonte do Methods II

Unit Methods.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
USERES("Methods.res");
USEFORM("uMetodo.cpp", formMetodo);
USEFORM("uGrafico.cpp", formGrafico);
USEFORM("uEquacao.cpp", formEquacao);
USEFORM("uInicio.cpp", formInicio);
USEFORM("uSobre.cpp", formSobre);
USEFORM("uInfo.cpp", formInfo);
USEFORM("uLaplace.cpp", formLaplace);
USEFORM("uTeoria.cpp", formTeoria);
USEFORM("uSolucao.cpp", formSolucao);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
        try
        {
                 Application->Initialize();
                 Application->CreateForm(__classid(TformInicio),
&formInicio);
                 Application->CreateForm(__classid(TformEquacao),
&formEquacao);
                 Application->CreateForm(__classid(TformMetodo),
&formMetodo);
                 Application->CreateForm(__classid(TformGrafico),
&formGrafico);
                 Application->CreateForm(__classid(TformSobre),  &formSobre);
                 Application->CreateForm(__classid(TformInfo), &formInfo);
                 Application->CreateForm(__classid(TformLaplace),
&formLaplace);
                 Application->CreateForm(__classid(TformTeoria),
&formTeoria);
                 Application->CreateForm(__classid(TformSolucao),
&formSolucao);
                 Application->Run();
        }
        catch (Exception &exception)
        {
                 Application->ShowException(&exception);
        }
        return 0;
}
//---------------------------------------------------------------------------
```

Unit uInicio.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "uInicio.h"
#include "uEquacao.h"
#include "uSobre.h"
#include "uTeoria.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TformInicio *formInicio;
//---------------------------------------------------------------------------
__fastcall TformInicio::TformInicio(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TformInicio::btnSairClick(TObject *Sender)
{
        exit(1);
}
//---------------------------------------------------------------------------
void __fastcall TformInicio::btnIniciarClick(TObject *Sender)
{
        formEquacao->Show();
}
//---------------------------------------------------------------------------
void __fastcall TformInicio::btnTeoriaClick(TObject *Sender)
{
         formTeoria->Show();
}
//---------------------------------------------------------------------------

void __fastcall TformInicio::btnCreditosClick(TObject *Sender)
{
        formSobre->Show();
}
//---------------------------------------------------------------------------
```

Unit uTeoria.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "uTeoria.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "SHDocVw_OCX"
#pragma resource "*.dfm"
TformTeoria *formTeoria;

void DesmarcarTitulo()
{
        formTeoria->shInt->Brush->Color = clWhite;
        formTeoria->shDif->Brush->Color = clWhite;
```

```
        formTeoria->shAdv->Brush->Color = clWhite;
        formTeoria->shAdvDif->Brush->Color = clWhite;
        formTeoria->shPoi->Brush->Color = clWhite;
}
//---------------------------------------------------------------------------
__fastcall TformTeoria::TformTeoria(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TformTeoria::FormActivate(TObject *Sender)
{
        String index_html = ExtractFilePath(ParamStr(0)) +
"teoria\\index.htm";
        TVariant Endereco = {index_html};
        pagina->Navigate(Endereco);
        DesmarcarTitulo();
        formTeoria->shInt->Brush->Color = clGray;
}
//---------------------------------------------------------------------------
void __fastcall TformTeoria::lblDifusaoClick(TObject *Sender)
{
        String difusao_html = ExtractFilePath(ParamStr(0)) +
"teoria\\difusao.htm";
        TVariant Endereco = {difusao_html};
        pagina->Navigate(Endereco);
        DesmarcarTitulo();
        formTeoria->shDif->Brush->Color = clGray;
}
//---------------------------------------------------------------------------
void __fastcall TformTeoria::lblAdveccaoClick(TObject *Sender)
{
        String adveccao_html = ExtractFilePath(ParamStr(0)) +
"teoria\\adveccao.htm";
        TVariant Endereco = {adveccao_html};
        pagina->Navigate(Endereco);
        DesmarcarTitulo();
        formTeoria->shAdv->Brush->Color = clGray;
}
//---------------------------------------------------------------------------
void __fastcall TformTeoria::lblIntroducaoClick(TObject *Sender)
{
        String index_html = ExtractFilePath(ParamStr(0)) +
"teoria\\index.htm";
        TVariant Endereco = {index_html};
        pagina->Navigate(Endereco);
        DesmarcarTitulo();
        formTeoria->shInt->Brush->Color = clGray;
}
//---------------------------------------------------------------------------
void __fastcall TformTeoria::btnFecharClick(TObject *Sender)
{
        formTeoria->Close();
}
//---------------------------------------------------------------------------
void __fastcall TformTeoria::lblPoissonClick(TObject *Sender)
{
        String poisson_html = ExtractFilePath(ParamStr(0)) +
"teoria\\poisson.htm";
        TVariant Endereco = {poisson_html};
        pagina->Navigate(Endereco);
```

```
        DesmarcarTitulo();
        formTeoria->shPoi->Brush->Color = clGray;
}
//---------------------------------------------------------------------------

void __fastcall TformTeoria::lblAdveccaoDifusaoClick(TObject *Sender)
{
        String adveccaodifusao_html = ExtractFilePath(ParamStr(0)) +
"teoria\\adveccaodifusao.htm";
        TVariant Endereco = {adveccaodifusao_html};
        pagina->Navigate(Endereco);
        DesmarcarTitulo();
        formTeoria->shAdvDif->Brush->Color = clGray;
}
//---------------------------------------------------------------------------
```

Unit uSobre.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "uSobre.h"
#include "uInfo.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TformSobre *formSobre;
//---------------------------------------------------------------------------
__fastcall TformSobre::TformSobre(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TformSobre::SpeedButton1Click(TObject *Sender)
{
        Close();
}
//---------------------------------------------------------------------------
void __fastcall TformSobre::btnInfoMariClick(TObject *Sender)
{
        String mariana_jpg = ExtractFilePath(ParamStr(0)) +
"figuras\\mariana.jpg";
        formInfo->imgPessoa->Picture->LoadFromFile(mariana_jpg);
        formInfo->lblNome->Caption = "Mariana Lisboa da Costa";
        formInfo->lblMatricula->Caption = "Matrícula: 198.31.047-7";
        formInfo->lblEmail->Caption = "m_lisboa@ig.com.br";
        formInfo->Caption = "Methods2 .: Autora: Mariana Lisboa da Costa";
        formInfo->Show();

}
//---------------------------------------------------------------------------

void __fastcall TformSobre::btnInfoDenisClick(TObject *Sender)
{
        String denis_jpg = ExtractFilePath(ParamStr(0)) +
"figuras\\denis.jpg";
```

```
        formInfo->imgPessoa->Picture->LoadFromFile(denis_jpg);
        formInfo->lblNome->Caption = "Denis André Ribeiro Leal";
        formInfo->lblEmail->Caption = "daleal@argentina.com";
        formInfo->lblMatricula->Caption = "Matrícula: 198.31.012-0";
        formInfo->Caption = "Methods2 .: Autor: Denis André Ribeiro Leal";
        formInfo->Show();
}
//---------------------------------------------------------------------------

void __fastcall TformSobre::btnInfoReginaClick(TObject *Sender)
{
        String regina_jpg = ExtractFilePath(ParamStr(0)) +
"figuras\\regina2.jpg";
        formInfo->imgPessoa->Picture->LoadFromFile(regina_jpg);
        formInfo->lblNome->Caption = "Regina Célia Paula Leal Toledo";
        formInfo->lblEmail->Caption = "Prof. de Métodos Numéricos II";
        formInfo->lblMatricula->Caption = "leal@dcc.ic.uff.br";
        formInfo->Caption = "Methods2 .: Orientadora: Regina Célia Paula
Leal Toledo";
        formInfo->Show();
}
//---------------------------------------------------------------------------

void __fastcall TformSobre::btnInfoMarcoClick(TObject *Sender)
{
        String marco_jpg = ExtractFilePath(ParamStr(0)) +
"figuras\\marco.jpg";
        formInfo->imgPessoa->Picture->LoadFromFile(marco_jpg);
        formInfo->lblNome->Caption = "Marco Antônio Silva Ramos";
        formInfo->lblEmail->Caption = "Prof. de Métodos Numéricos I";
        formInfo->lblMatricula->Caption = "marco@dcc.ic.uff.br";
        formInfo->Caption = "Methods2 .: Orientador: Marco Antônio Silva
Ramos";
        formInfo->Show();
}
//---------------------------------------------------------------------------
```

Unit uEquacao.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#include <winbase.h>
#pragma hdrstop

#include "uInicio.h"
#include "uEquacao.h"
#include "uMetodo.h"
#include "uGrafico.h"
#include "uLaplace.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TformEquacao *formEquacao;

int idEquacao = 0;

void IniciarCampos()
{
        String auxDt, auxDx, auxK;
        if(DecimalSeparator == ',')
```

```
        {
                auxDt = "0,0055";
                auxDx = "0,1";
                auxK = "0,2";
        }
        else
        {
                auxDt = "0.0055";
                auxDx = "0.1";
                auxK = "0.2";
        }
        formGrafico->edL->Text = "10";
        formGrafico->edTf->Text = "20";
        formGrafico->edDt->Text = auxDt;
        formGrafico->edDx->Text = auxDx;

        if(formGrafico->lblEquacao->Caption == "EQUAÇÃO DE DIFUSÃO OU
CALOR")
        {
                formGrafico->edV->Text = "0";
                formGrafico->edK->Text = auxK;
        }
        else if(formGrafico->lblEquacao->Caption == "EQUAÇÃO DE ADVECÇÃO OU
CONVECÇÃO")
        {
                formGrafico->edK->Text = "0";
                formGrafico->edV->Text = "1";
        }
        else if(formGrafico->lblEquacao->Caption == "EQUAÇÃO DE ADVECÇÃO-
DIFUSÃO")
        {
                formGrafico->edK->Text = auxK;
                formGrafico->edV->Text = "1";
        }
}


//---------------------------------------------------------------------------
__fastcall TformEquacao::TformEquacao(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TformEquacao::btnDifClick(TObject *Sender)
{
        idEquacao = 1;
        formMetodo->Show();
        formGrafico->lblEquacao->Caption = "EQUAÇÃO DE DIFUSÃO OU CALOR";
        formMetodo->btnMetodo1->Caption = "&Crank-Nicholson";
        formMetodo->btnMetodo2->Visible = false;
        formMetodo->Label5->Caption = "Método Crank-Nicolson";
        formMetodo->Label6->Caption = "";
        formMetodo->lblProblema->Caption = "Problema de Difusão ou Calor" ;
        String difusao_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\Difusao.bmp";
        formMetodo->ImageProblema->Picture->LoadFromFile(difusao_bmp);
        String explicito_dif_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\explicito-dif.bmp";
        formMetodo->Image1->Picture->LoadFromFile(explicito_dif_bmp);
        String implicito_dif_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\implicito-dif.bmp";
```

```
        formMetodo->Image2->Picture->LoadFromFile(implicito_dif_bmp);
        String crank_jpg = ExtractFilePath(ParamStr(0)) + "figuras\\crank-
nicolson.bmp";
        formMetodo->Image3->Picture->LoadFromFile(crank_jpg);
        formMetodo->Image4->Visible = false;
        String contornodifusao_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\contornodifusao.bmp";
        formMetodo->imagemContorno->Picture-
>LoadFromFile(contornodifusao_bmp);
        String exata_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\exatadifusao.bmp";
        formMetodo->imgExata->Picture->LoadFromFile(exata_bmp);
        formGrafico->edK->Visible = true;
        formGrafico->edK->Top = 199;
        formGrafico->edV->Visible = false;
        formGrafico->edV->Text = "0";
        formGrafico->Label5->Top = 202;
        formGrafico->Label5->Visible = true;
        formGrafico->Label6->Visible = false;
        IniciarCampos();

}
//---------------------------------------------------------------------------

void __fastcall TformEquacao::btnAdvClick(TObject *Sender)
{
        idEquacao = 2;
        formMetodo->Show();
        formGrafico->lblEquacao->Caption = "EQUAÇÃO DE ADVECÇÃO OU
CONVECÇÃO";
        formMetodo->btnMetodo1->Caption = "Explícito-&UpWind";
        formMetodo->btnMetodo2->Visible = true;
        formMetodo->btnMetodo2->Caption = "I&mplícito-UpWind";
        formMetodo->Label5->Caption = "Método Explícito-UpWind";
        formMetodo->Label6->Caption = "Método Implícito-UpWind";
        formMetodo->lblProblema->Caption = "Problema de Advecção ou
Convecção" ;
        String adveccao_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\Adveccao.bmp";
        formMetodo->ImageProblema->Picture->LoadFromFile(adveccao_bmp);
        String explicito_adv_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\explicito-adv.bmp";
        formMetodo->Image1->Picture->LoadFromFile(explicito_adv_bmp);
        String implicito_adv_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\implicito-adv.bmp";
        formMetodo->Image2->Picture->LoadFromFile(implicito_adv_bmp);
        String expupwind_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\explicito-up-adv.bmp";
        formMetodo->Image3->Picture->LoadFromFile(expupwind_bmp);
        String impupwind_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\implicito-up-adv.bmp";
        formMetodo->Image4->Picture->LoadFromFile(impupwind_bmp);
        formMetodo->Image4->Visible = true;
        String contornoadveccao_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\contornoadveccao.bmp";
        formMetodo->imagemContorno->Picture-
>LoadFromFile(contornoadveccao_bmp);
        String exata_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\exataadveccao.bmp";
        formMetodo->imgExata->Picture->LoadFromFile(exata_bmp);
        formGrafico->edV->Visible = true;
```

```
        formGrafico->edV->Top = 199;
        formGrafico->edK->Visible = false;
        formGrafico->edK->Text = "0";
        formGrafico->Label5->Visible = false;
        formGrafico->Label6->Top = 202;
        formGrafico->Label6->Visible = true;
        IniciarCampos();
}
//---------------------------------------------------------------------------

void __fastcall TformEquacao::btnAdvDifClick(TObject *Sender)
{
        idEquacao = 3;
        formMetodo->Show();
        formGrafico->lblEquacao->Caption = "EQUAÇÃO DE ADVECÇÃO-DIFUSÃO";
        formMetodo->btnMetodo1->Caption = "Explícito-&UpWind";
        formMetodo->btnMetodo2->Visible = true;
        formMetodo->btnMetodo2->Caption = "I&mplícito-UpWind";
        formMetodo->Label5->Caption = "Método Explícito-UpWind";
        formMetodo->Label6->Caption = "Método Implícito-UpWind";
        formMetodo->lblProblema->Caption = "Problema de Advecção-Difusão" ;
        String adveccao_difusao_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\Adveccao-difusao.bmp";
        formMetodo->ImageProblema->Picture-
>LoadFromFile(adveccao_difusao_bmp);
        String explicito_adv_dif_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\explicito-adv-dif.bmp";
        formMetodo->Image1->Picture->LoadFromFile(explicito_adv_dif_bmp);
        String implicito_adv_dif_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\implicito-adv-dif.bmp";
        formMetodo->Image2->Picture->LoadFromFile(implicito_adv_dif_bmp);
        String expupwind_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\explicito-up-adv-dif.bmp";
        formMetodo->Image3->Picture->LoadFromFile(expupwind_bmp);
        String impupwind_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\implicito-up-adv-dif.bmp";
        formMetodo->Image4->Picture->LoadFromFile(impupwind_bmp);
        formMetodo->Image4->Visible = true;
        String contornoadveccaodifusao_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\contornoadveccaodifusao.bmp";
        formMetodo->imagemContorno->Picture-
>LoadFromFile(contornoadveccaodifusao_bmp);
        String exata_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\exataadveccaodifusao.bmp";
        formMetodo->imgExata->Picture->LoadFromFile(exata_bmp);
        formGrafico->edV->Visible = true;
        formGrafico->edV->Top = 239;
        formGrafico->Label5->Top = 202;
        formGrafico->Label5->Visible = true;
        formGrafico->edK->Visible = true;
        formGrafico->edK->Top = 199;
        formGrafico->Label6->Top = 242;
        formGrafico->Label6->Visible = true;
        IniciarCampos();
}
//---------------------------------------------------------------------------

void __fastcall TformEquacao::btnTeoDifClick(TObject *Sender)
{
        String difusao_htm = ExtractFilePath(ParamStr(0)) +
"teoria\\difusao.htm";
```

```
        ShellExecute(Application-
>Handle,"open",difusao_htm.c_str(),0,0,SW_SHOWDEFAULT);
}
//---------------------------------------------------------------------------

void __fastcall TformEquacao::SpeedButton1Click(TObject *Sender)
{
        String adveccao_htm = ExtractFilePath(ParamStr(0)) +
"teoria\\adveccao.htm";
        ShellExecute(Application-
>Handle,"open",adveccao_htm.c_str(),0,0,SW_SHOWDEFAULT);
}
//---------------------------------------------------------------------------

void __fastcall TformEquacao::Label2Click(TObject *Sender)
{
        formInicio->Show();
        Close();
}
//---------------------------------------------------------------------------

void __fastcall TformEquacao::FormClose(TObject *Sender,
      TCloseAction &Action)
{
        formInicio->Show();
}
//---------------------------------------------------------------------------

void __fastcall TformEquacao::btnLapClick(TObject *Sender)
{
        formLaplace->Show();
}
//---------------------------------------------------------------------------
```

Unit uMetodo.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#include <io.h>
#include <stdio.h>
#pragma hdrstop

#include "uInicio.h"
#include "uMetodo.h"
#include "uEquacao.h"
#include "uGrafico.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"


TformMetodo *formMetodo;

//---------------------------------------------------------------------------
__fastcall TformMetodo::TformMetodo(TComponent* Owner)
        : TForm(Owner)
```

```
{
}
//---------------------------------------------------------------------------
void __fastcall TformMetodo::btnExplicitoClick(TObject *Sender)
{
       formGrafico->lblTitulo->Caption = "MÉTODO EXPLÍCITO - DIF. CENTRAL" ;
       formGrafico->Show();
       Close();
}
//---------------------------------------------------------------------------

void __fastcall TformMetodo::btnImplicitoClick(TObject *Sender)
{
       formGrafico->lblTitulo->Caption = "MÉTODO IMPLÍCITO - DIF. CENTRAL";
       formGrafico->Show();
       Close();
}
//---------------------------------------------------------------------------

void __fastcall TformMetodo::Label2Click(TObject *Sender)
{
        Close();
        formEquacao->Visible = true;
}
//---------------------------------------------------------------------------

void __fastcall TformMetodo::btnMetodo1Click(TObject *Sender)
{
        if(formGrafico->lblEquacao->Caption == "EQUAÇÃO DE DIFUSÃO OU
CALOR")
                formGrafico->lblTitulo->Caption = "MÉTODO CRANK-NICOLSON";
        else if(formGrafico->lblEquacao->Caption == "EQUAÇÃO DE ADVECÇÃO OU
CONVECÇÃO")
                formGrafico->lblTitulo->Caption = "MÉTODO EXPLÍCITO-UPWIND";
        else if(formGrafico->lblEquacao->Caption == "EQUAÇÃO DE ADVECÇÃO-
DIFUSÃO")
                formGrafico->lblTitulo->Caption = "MÉTODO EXPLÍCITO-UPWIND";
        formGrafico->Show();
        Close();
}
//---------------------------------------------------------------------------

void __fastcall TformMetodo::FormClose(TObject *Sender,
      TCloseAction &Action)
{
        formMetodo->Show();
}
//---------------------------------------------------------------------------

void __fastcall TformMetodo::btnMetodo2Click(TObject *Sender)
{
        formGrafico->lblTitulo->Caption = "MÉTODO IMPLÍCITO-UPWIND";
        formGrafico->Show();
        Close();
}
//---------------------------------------------------------------------------
```

Unit uLaplace.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#include <math.h>

#pragma hdrstop

#include "uLaplace.h"
#include "uEquacao.h"
#include "uInicio.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)

#pragma resource "*.dfm"

TformLaplace *formLaplace;

double **matriz, *vetor, *coordx, *coordy, *exato, *contorno, *contornox,
*contornoy;
int *indices, nnos;

bool VerificarCampos()
{
        bool todosPreenchidos = true;
        if((formLaplace->edDx->Text == "") || (formLaplace->edDy->Text ==
""))
        {
                todosPreenchidos = false;
        }
        return(todosPreenchidos);
}

double **Alocar_matriz(int m, int n)
{
  double **v;
  int   i;

  //Aloca as linhas da matriz
  v = (double **) calloc (m, sizeof(double *));// Um vetor de m ponteiros
para Tnum
  //Aloca as colunas da matriz
  for ( i = 0; i < m; i++ )
      v[i] = (double *) calloc (n, sizeof(double));  //m vetores de n
doubles
  return (v); //Retorna o ponteiro para a matriz
}

double **Liberar_matriz(int n, double **v)
{
  int  i;
  if (v == NULL) return (NULL);
  for (i=0; i<n; i++)
        free (v[i]); //Libera as linhas da matriz
  free (v);      //Libera a matriz
  return (NULL);
}

double funcaof(double val_x, double val_y)
{
```

```
        double aux;
        if (formLaplace->rdEquacao1->Checked)
                aux = 4;
        else if (formLaplace->rdEquacao2->Checked)
                aux = (-2*(M_PI*M_PI)*sin(M_PI*val_x)*cos(M_PI*val_y));
        return aux;
}

double funcaog(double val_x, double val_y)
{
        double aux;
        if (formLaplace->rdEquacao1->Checked)
                aux = ((val_x*val_x)+(val_y*val_y));
        else
                aux = (sin(M_PI*val_x)*cos(M_PI*val_y));
        return aux;
}

//Divide matriz em L e U
void LU_fatoracao(double **A, int *indx, int n)
{

    int M = n;
    int N = n;


    int i=0,j=0,k=0,jp=0;



    int minMN =  (M < N ? M : N) ;         // min(M,N);

    for (j=1; j<= minMN; j++)
    {
        jp = j;
        double t = fabs(A[j-1][j-1]);
        for (i=j+1; i<=M; i++)
            if ( fabs(A[i-1][j-1]) > t)
            {
                jp = i;
                t = fabs(A[i-1][j-1]);
            }
        indx[j-1] = jp;
        if (jp != j)
            for (k=1; k<=N; k++)
            {
                t = A[j-1][k-1];
                A[j-1][k-1] = A[jp-1][k-1];
                A[jp-1][k-1] =t;
            }
        if (j<M)
        {
            double recp =  1.0 / A[j-1][j-1];
            for (k=j+1; k<=M; k++)
                A[k-1][j-1] *= recp;
        }
        if (j < minMN)
        {
            int ii,jj;
            for (ii=j+1; ii<=M; ii++)
                for (jj=j+1; jj<=N; jj++)
```

```
                    A[ii-1][jj-1] -= A[ii-1][j-1]*A[j-1][jj-1];
        }
    }
}

//Resolve os sistemas
double *LU_resolucao(double **A, int *indx, double *b, int n)
{
    int i,ii=0,ip,j;
    double sum = 0.0;
    for (i=1;i<=n;i++)
    {
        ip=indx[i-1];
        sum=b[ip-1];
        b[ip-1]=b[i-1];
        if (ii)
            for (j=ii;j<=i-1;j++)
                sum -= A[i-1][j-1]*b[j-1];
        else if (sum) ii=i;
            b[i-1]=sum;
    }
    for (i=n;i>=1;i--)
    {
        sum=b[i-1];
        for (j=i+1;j<=n;j++)
            sum -= A[i-1][j-1]*b[j-1];
        b[i-1]=sum/A[i-1][i-1];
    }

    return(b);
}

//Arredonda double para inteiro mais próximo
int arredonda(double x)
{
      if ((x - abs(x)) >= 0.5)
                return ceil(x);
      else
               return floor(x);
}

//---------------------------------------------------------------------------
__fastcall TformLaplace::TformLaplace(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TformLaplace::btnGerarClick(TObject *Sender)
{
        bool TodosCamposPreenchidos = VerificarCampos();
        if (TodosCamposPreenchidos == false)
        {
                ShowMessage ("Atenção, todos os campos devem ser
preenchidos.");
                return;
        }

        edDx->Font->Color = clBlack;
        edDy->Font->Color = clBlack;

        double Lx = 1;      //Domínios em x e y.
```

```
        double Ly = 1;

        double Hx = StrToFloat((edDx->Text));     //Recebe tamanho do
elemento no eixo x
        double Hy = StrToFloat((edDy->Text));     //Recebe tamanho do
elemento no eixo y

        int NElem = (arredonda(Lx/Hx));   //Nº elementos em x
        int MElem = (arredonda(Ly/Hy));   //Nº elementos em y
        double NovoHx = Lx/NElem;
        double NovoHy = Ly/MElem;

        if(Hx != NovoHx || Hy != NovoHy)
        {
                ShowMessage("Divisão não-inteira. Valores arredondados para:
\nDx = " + FloatToStrF(NovoHx,ffGeneral,4,2) + "\nDy = " +
FloatToStrF(NovoHy,ffGeneral,4,2));
                if(Hx != NovoHx)
                {
                        edDx->Text = FloatToStrF(NovoHx,ffGeneral,4,2);
                        edDx->Font->Color = clRed;
                        Hx = NovoHx;
                }
                if(Hy != NovoHy)
                {
                        edDy->Text = FloatToStrF(NovoHy,ffGeneral,4,2);
                        edDy->Font->Color = clRed;
                        Hy = NovoHy;
                }
        }

        int N = NElem - 1;   //Nº de nós interiores no eixo x
        int M = MElem - 1;   //Nº de nós interiores no eixo y

        nnos = M*N;    //Nº total de nós interiores

        if(nnos > 365)
        {
                if (MessageDlg("Atenção! Os valores escolhidos podem fazer
com que o programa trave.\nClique OK para continuar ou Cancel para escolher
outros valores.", mtWarning, TMsgDlgButtons() << mbOK << mbCancel, 0) ==
mbOK)
                {
                         return;
                }
        }

        int nnoscont = (NElem+1)*(MElem+1);  //Nº de nós no contorno

        //Alocação dos vetores necessários
        indices = (int *)calloc(nnos,sizeof(double));
        exato = (double *)calloc(nnos,sizeof(double));
        coordx = (double *)calloc(nnos,sizeof(double));
        coordy= (double *)calloc(nnos,sizeof(double));
        matriz = Alocar_matriz(nnos,nnos);
        vetor = (double *)calloc(nnos,sizeof(double));
        contorno = (double *)calloc(nnoscont,sizeof(double));
        contornox = (double *)calloc(nnoscont,sizeof(double));
        contornoy = (double *)calloc(nnoscont,sizeof(double));

        //Inicializa a matriz e os vetores
```

```
int j, i;
for(i=0;i<nnos;i++)
{
        for(j=0;j<nnos;j++)
        {
                matriz[i][j] = 0;
        }
        exato[i];
        coordx[i];
        coordy[i];
        vetor[i];
}

int k;
//Preenchimento da matriz
for(k = 1; k <= nnos; k++)
       matriz[k-1][k-1] = -2*((Hx*Hx)+(Hy*Hy));
for(k = 1; k <= nnos-1; k++)
        matriz[k-1][k] = Hy*Hy;
for(k = 2; k <= nnos; k++)
        matriz[k-1][k-2] = Hy*Hy;
for(k = 1; k <= ((M-1)*N); k++)
        matriz[k-1][k+N-1] = Hx*Hx;
for(k = N+1; k <= nnos; k++)
        matriz[k-1][k-N-1] = Hx*Hx;

//Preenchimento do vetor
k = 0;
double ValorMinimo = 100;
double ValorMaximo = -100;
for(j = 1; j<=M; j++)
{
        for(i = 1; i<=N; i++)
        {
                k += 1;
                coordx[k-1] = i*Hx;
                coordy[k-1] = j*Hy;
                exato[k-1] = funcaog(i*Hx, j*Hy);
                vetor[k-1] = (Hx*Hx)*(Hy*Hy)*funcaof(i*Hx, j*Hy);

                if(i==1)
                {
                        vetor[k-1] -= (Hy*Hy)*funcaog(0, j*Hy);
                        if(j>1 && N!=1)
                                matriz[k-1][k-2] = 0;
                }
                if(i==N)
                {
                        vetor[k-1] -= (Hy*Hy)*funcaog(Lx, j*Hy);
                        if(j<M && N!=1)
                                matriz[k-1][k] = 0;
                }
                if(j==1)
                        vetor[k-1] -= (Hx*Hx)*funcaog(i*Hx, 0);
                if(j==M)
                        vetor[k-1] -= (Hx*Hx)*funcaog(i*Hx, Ly);
        }
}

//Verificação da condição de contorno
k =0;
```

```
        for(j = 0; j<=MElem; j++)
        {
                for(i = 0; i<=NElem; i++)
                {
                        if((j==0) || (j==MElem) || (i==0) || (i==NElem))
                        {
                                contorno[k] =  funcaog(i*Hx, j*Hy);
                                if(contorno[k] < ValorMinimo)
                                        ValorMinimo = contorno[k];
                                if(contorno[k] > ValorMaximo)
                                        ValorMaximo = contorno[k];
                                contornox[k] =  i*Hx;
                                contornoy[k] =  j*Hy;
                        }
                        k+=1;
                }
        }

        //Fatoração LU resolve o sistema
       LU_fatoracao(matriz,indices,nnos);
       vetor = LU_resolucao(matriz,indices,vetor,nnos);


        //Prepara o gráfico para ser desenhado
        formLaplace->Series1->Clear();
        formLaplace->Series2->Clear();
        ValorMinimo = arredonda(ValorMinimo);
        ValorMaximo = arredonda(ValorMaximo);
        Series1->GetVertAxis->SetMinMax(ValorMinimo,ValorMaximo);
        Series2->GetVertAxis->SetMinMax(ValorMinimo,ValorMaximo);
        Series1->NumXValues = NElem;
        Series1->NumZValues = MElem;
        Series2->NumXValues = NElem;
        Series2->NumZValues = MElem;

        //Coloca os valores dos pontos do contorno no gráfico
        for (i=0;i<nnoscont;i++)
        {
                formLaplace->Series1-
>AddXYZ(contornox[i],contorno[i],contornoy[i], ' ', clTeeColor);
                formLaplace->Series2-
>AddXYZ(contornox[i],contorno[i],contornoy[i], ' ', clTeeColor);
        }
        //Coloca os valores dos pontos interiores no gráfico
        for (i=0;i<nnos;i++)
        {
                formLaplace->Series1->AddXYZ(coordx[i],vetor[i],coordy[i],
' ', clTeeColor);
                formLaplace->Series2->AddXYZ(coordx[i],exato[i],coordy[i],
' ', clTeeColor);
        }
        formLaplace->Series1->Active = true;
        formLaplace->Series2->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TformLaplace::FormCreate(TObject *Sender)
{
       Series1->IrregularGrid = true;
       Series2->IrregularGrid = true;
       formLaplace->Series1->Active = false;
       formLaplace->Series2->Active = false;
```

```
}
//---------------------------------------------------------------------------

void __fastcall TformLaplace::btnSairClick(TObject *Sender)
{
        matriz = Liberar_matriz(nnos, matriz);
        free(vetor);
        free(exato);
        free(coordx);
        free(coordy);
        free(contorno);
        free(contornox);
        free(contornoy);
        exit(1);
}
//---------------------------------------------------------------------------

void __fastcall TformLaplace::FormClose(TObject *Sender,
      TCloseAction &Action)
{
        matriz = Liberar_matriz(nnos, matriz);
        free(vetor);
        free(exato);
        free(coordx);
        free(coordy);
        free(contorno);
        free(contornox);
        free(contornoy);
        edDx->Text = "";
        edDy->Text = "";
}
//---------------------------------------------------------------------------

void __fastcall TformLaplace::btnMenuEquacaoClick(TObject *Sender)
{
        formEquacao->Show();
        Close();
}
//---------------------------------------------------------------------------

void __fastcall TformLaplace::btnMenuInicioClick(TObject *Sender)
{
        formInicio->Show();
        Close();
}
//---------------------------------------------------------------------------

void __fastcall TformLaplace::barRotationChange(TObject *Sender)
{
        formLaplace->grafLaplace->View3DOptions->Orthogonal = false;
        formLaplace->grafLaplace->View3DOptions->Rotation = barRotation-
>Position;
        formLaplace->grafExato->View3DOptions->Orthogonal = false;
        formLaplace->grafExato->View3DOptions->Rotation = barRotation-
>Position;
}
//---------------------------------------------------------------------------

void __fastcall TformLaplace::barElevationChange(TObject *Sender)
{
        formLaplace->grafLaplace->View3DOptions->Orthogonal = false;
```

```
        formLaplace->grafLaplace->View3DOptions->Elevation = barElevation-
>Position;
        formLaplace->grafExato->View3DOptions->Orthogonal = false;
        formLaplace->grafExato->View3DOptions->Elevation = barElevation-
>Position;
}
//---------------------------------------------------------------------------

void __fastcall TformLaplace::btnDefaultClick(TObject *Sender)
{
        formLaplace->barRotation->Position = 345;
        formLaplace->barElevation->Position = 345;
        grafLaplace->View3DOptions->Orthogonal = true;
        grafExato->View3DOptions->Orthogonal = true;
}
//---------------------------------------------------------------------------

void __fastcall TformLaplace::edDxKeyPress(TObject *Sender, char &Key)
{
        if ((Key == ',') || (Key == '.'))
                Key = DecimalSeparator;
}
//---------------------------------------------------------------------------

void __fastcall TformLaplace::edDyKeyPress(TObject *Sender, char &Key)
{
        if ((Key == ',') || (Key == '.'))
                Key = DecimalSeparator;
}
//---------------------------------------------------------------------------

void __fastcall TformLaplace::rdEquacao1Click(TObject *Sender)
{
                String solucao1_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\x2y2.bmp";
                formLaplace->ImageSolExata->Picture-
>LoadFromFile(solucao1_bmp);
}
//---------------------------------------------------------------------------

void __fastcall TformLaplace::rdEquacao2Click(TObject *Sender)
{
     String solucao2_bmp = ExtractFilePath(ParamStr(0)) +
"figuras\\sen(pix)cos(piy).bmp";
     formLaplace->ImageSolExata->Picture->LoadFromFile(solucao2_bmp);

}
//---------------------------------------------------------------------------
```

Unit uGrafico.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#include <math.h>
#include <stdio.h>
#pragma hdrstop


#include "uGrafico.h"
#include "uMetodo.h"
#include "uInicio.h"
#include "uEquacao.h"
#include "uSolucao.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TformGrafico *formGrafico;

typedef double Tnum;


Tnum **A, *C[2], xi, xf, V, dx, dt, ti, tf, K, t, x;
int *ind, nelems, k0, k1, idMetodo;
FILE *arq;



//Verifica se usuário não deixou algum campo em branco
bool VerificarCamposPreenchidos()
{
        bool todosPreenchidos = true;
        if((formGrafico->edTf->Text == "") || (formGrafico->edDt->Text ==
"") ||
          (formGrafico->edK->Text == "") || (formGrafico->edV->Text == "")
||
          (formGrafico->edDx->Text == "") || (formGrafico->edL->Text == ""))
        {
                todosPreenchidos = false;
        }
        return(todosPreenchidos);
}


//Calcula valor exato da equação
Tnum C_Exato(float x,float t)
{
      return(exp(-M_PI*M_PI*K*t)*sin(M_PI*(x-V*t)));
}


//Monta o gráfico que será mostrado
void grafico(int vetor, double t)
{
         double y;

         formGrafico->graf->BottomAxis->Maximum = nelems-1;
         formGrafico->Series1->Clear();
         formGrafico->Series2->Clear();
         formGrafico->Series3->Clear();
```

```
        x = xi;
        formGrafico->Series3->AddY(0, ' ', clTeeColor);
        for (int m=0; m<nelems;m++)
        {
               y = C_Exato(x, t);
               if(t>0.0)
                       fprintf(arq, "%.20f \n",C[vetor][m]);
               formGrafico->Series2->AddY(y, ' ', clTeeColor);
               formGrafico->Series1->AddY(C[vetor][m], ' ', clTeeColor);
               formGrafico->Series3->AddY(0, ' ', clTeeColor);
               x+=dx;
        }
}


//Aloca espaço em mémoria para um vetor do tipo double
Tnum *Alocar_vetor_real (int n)
{
  Tnum *v;        /* ponteiro para o vetor */

  /* aloca o vetor */
  v = (Tnum *)calloc (n, sizeof(Tnum));
  if (v == NULL)
  {
     ShowMessage ("** Memoria Insuficiente **");
     return (NULL);
  }
  return (v);    /* retorna o ponteiro para o vetor */
}


//Libera o espaço alocado para um vetor do tipo double
Tnum *Liberar_vetor_real (Tnum *v)
{
  if (v == NULL) return (NULL);
  free(v);        /* libera o vetor */
  return (NULL);  /* retorna o ponteiro */
}


//Aloca espaço para um vetor do tipo inteiro
int *Alocar_vetor_inteiro (int n)
{
  int *v;        /* ponteiro para o vetor */

  /* aloca o vetor */
  v = (int *)calloc (n, sizeof(int));
  return (v);    /* retorna o ponteiro para o vetor */
}


//Libera o espaço alocado a um vetor do tipo inteiro
int *Liberar_vetor_inteiro (int *v)
{
  if (v == NULL) return (NULL);
  free(v);        /* libera o vetor */
  return (NULL);  /* retorna o ponteiro */
}


//Aloca espaço para uma matriz do tipo double
```

```
Tnum **Alocar_matriz_real (int m, int n)
{
  Tnum **v;  /* ponteiro para a matriz */
  int   i;    /* variavel auxiliar      */

  /* aloca as linhas da matriz */
  v = (Tnum **) calloc (m, sizeof(Tnum *));    // Um vetor de m ponteiros
para Tnum

  /* aloca as colunas da matriz */
  for ( i = 0; i < m; i++ )
      v[i] = (Tnum *) calloc (n, sizeof(Tnum));/* n vetores de n floats */
  return (v); /* retorna o ponteiro para a matriz */
}


//Libera espaço alocado a uma matriz do tipo double
Tnum **Liberar_matriz_real (int n, Tnum **v)
{
  int  i;  /* variavel auxiliar */
  if (v == NULL) return (NULL);
  for (i=0; i<n; i++)
        free (v[i]); /* libera as linhas da matriz */
  free (v);      /* libera a matriz (vetor de ponteiros) */
  return (NULL); /* retorna um ponteiro nulo */
}


//Preenche a matriz de coeficiente
Tnum **PreencherMatriz(int metodo, Tnum **v)
{
        Tnum X1, X2, X3;

        if (metodo == 2)
        {
              X1 = ((-K*dt)/(dx*dx)) - ((V*dt)/(2*dx));
              X2 = ((2*K*dt)/(dx*dx)) + 1;
              X3 = ((-K*dt)/(dx*dx)) + ((V*dt)/(2*dx));
        }
        else if (metodo == 4)
        {
                X1 = -(K*dt)/(dx*dx);
              X1 -= ((fabs(V)+V)*dt)/(2*dx);
              X2 = ((2*K*dt)/(dx*dx))+1;
              X2 += (dt*fabs(V))/(dx);
              X3 = -(K*dt)/(dx*dx);
              X3 -= (dt*(fabs(V)-V))/(2*dx);
        }
        else if (metodo == 5)
        {
                X1 = ((-K*dt)/(2*dx*dx));
              X2 = ((K*dt)/(dx*dx)) + 1;
              X3 = ((-K*dt)/(2*dx*dx));
        }

      //Preenche todos as posições da matriz com zeros
      for (int m = 0;m < nelems;m++)
      {
            for (int n = 0;n < nelems;n++)
            {
                  v[m][n]= 0;
```

```
            }
      }

      //Zera a primeira linha, última linha, ou ambas, dependendo de qual é
a equação
      if (K==0)
      {
            //Se difusão for igual a zero e V>0 então zera a primeira linha
            if (V>0)
            {
                  v[0][0] = 1;
                        v[nelems-1][nelems-1] = X2 + X3;
                        v[nelems-1][nelems-2] = X1;

            }
            //Se difusão for igual a zero e V<0 então zera a última linha
            else if (V<0)
            {
                  v[nelems-1][nelems-1] = 1;
                        v[0][0] = X1 + X2;
                        v[0][1] = X3;
            }
      }
      //Se difusão for diferente de zero então zera a primeira e última
linhas
      else
      {
            v[0][0] = 1;
            v[nelems-1][nelems-1] = 1;
      }

      //Preenche o resto da matriz
      for (int m = 1;m < nelems-1;m++)
      {
            for (int n = 0;n < nelems;n++)
            {
                  if (m==n)
                        v[m][n] = X2;
                  else if (n==m+1)
                        v[m][n] = X3;
                  else if (n==m-1)
                        v[m][n] = X1;
                         else
                               v[m][n] = 0;
            }

      }
        return(v);
}


//Inicia vetor de resultados
void Inic_C(void)
{
      int i;
      Tnum x;

      x = xi;
      for (i = 0;i < nelems;i++)
      {
              C[0][i] = sin(M_PI*x);
```

```
                x += dx;
                    grafico(k0, t);
                    fprintf(arq, "%.20f \n",C[k0][i]);
        }
          fprintf(arq, "----------------\n\n");
}

//Divide matriz em L e U
void LU_fatoracao(double **A, int *indx, int n)
{

    int M = n;
    int N = n;


    int i=0,j=0,k=0,jp=0;



    int minMN =  (M < N ? M : N) ;         // min(M,N);

    for (j=1; j<= minMN; j++)
    {
        jp = j;
        double t = fabs(A[j-1][j-1]);
        for (i=j+1; i<=M; i++)
            if ( fabs(A[i-1][j-1]) > t)
            {
                jp = i;
                t = fabs(A[i-1][j-1]);
            }
        indx[j-1] = jp;
        if (jp != j)
            for (k=1; k<=N; k++)
            {
                t = A[j-1][k-1];
                A[j-1][k-1] = A[jp-1][k-1];
                A[jp-1][k-1] =t;
            }
        if (j<M)
        {
            double recp =  1.0 / A[j-1][j-1];
            for (k=j+1; k<=M; k++)
                A[k-1][j-1] *= recp;
        }
        if (j < minMN)
        {
            int ii,jj;
            for (ii=j+1; ii<=M; ii++)
                for (jj=j+1; jj<=N; jj++)
                    A[ii-1][jj-1] -= A[ii-1][j-1]*A[j-1][jj-1];
        }
    }
}

//Resolve os sistemas
double *LU_resolucao(double **A, int *indx, double *b, int n)
{
    int i,ii=0,ip,j;
    double sum = 0.0;
    for (i=1;i<=n;i++)
```

```
    {
        ip=indx[i-1];
        sum=b[ip-1];
        b[ip-1]=b[i-1];
        if (ii)
            for (j=ii;j<=i-1;j++)
                sum -= A[i-1][j-1]*b[j-1];
        else if (sum) ii=i;
            b[i-1]=sum;
    }
    for (i=n;i>=1;i--)
    {
        sum=b[i-1];
        for (j=i+1;j<=n;j++)
            sum -= A[i-1][j-1]*b[j-1];
        b[i-1]=sum/A[i-1][i-1];
    }

    return(b);
}


//Calculo do Método de Crank-Nicolson
void CalcNovoTempoCrankNicolson()
{
        Tnum aux;

        for (int i=1; i<nelems-1; i++)
        {
                aux = ((K*dt)/(2*dx*dx))*C[k0][i-1];
                aux += (1 - ((K*dt)/(dx*dx)))*C[k0][i];
                aux += ((K*dt)/(2*dx*dx))*C[k0][i+1];
                C[k1][i] = aux;
        }
        C[k1] = LU_resolucao(A, ind, C[k1], nelems);
}


//Calculo do Método Implícito-Dif.Central
void CalcNovoTempoImplicito()
{
        C[0] = LU_resolucao(A, ind, C[0], nelems);
}


//Calculo do Método Implícito-UpWind
void CalcNovoTempoImplicitoUpWind()
{
        C[0] = LU_resolucao(A, ind, C[0], nelems);
}


//Calculo do Método Explícito-Dif.Central
void CalcNovoTempoExplicito()
{
      int i;
      Tnum aux;

        if (K==0)
        {
                if (V>0)
```

```
                        C[k0][nelems-1] = C[k0][nelems-2];
                else if(V<0)
                        C[k0][0] = C[k0][1];
          }
        for (i = 1; i < nelems-1; i++)
        {
                aux = K*(C[k0][i-1] - 2*C[k0][i] + C[k0][i+1])/(dx*dx);
                aux = aux - V*(C[k0][i+1] - C[k0][i-1])/(2*dx);
                aux = aux*dt + C[k0][i];
                C[k1][i] = aux;
        }
}


//Calculo do Método Explícito-UpWind
void CalcNovoTempoExplicitoUpWind()
{
        int i;
        Tnum aux;

          if (K==0)
          {
                if (V>0)
                        C[k0][nelems-1] = C[k0][nelems-2];
                else if(V<0)
                        C[k0][0] = C[k0][1];
          }
        for (i = 1;i < nelems-1;i++)
        {
                aux = K*(C[k0][i-1] - 2*C[k0][i] + C[k0][i+1])/(dx*dx);
                aux +=  ( (fabs(V)-V)*C[k0][i+1] - 2*fabs(V)*C[k0][i]
                     +   (fabs(V)+V)*C[k0][i-1] ) / (2*dx);
                aux =  aux*dt + C[k0][i];
                C[k1][i] = aux;
        }
}


//Calculo das condições de contorno
void cond_contorno(int kn)
{
          if (K==0)
          {
                if (V>0)
                        C[kn][0] = C_Exato(0,t);
                else if (V<0)
                        C[kn][nelems-1] =  C_Exato(xf,t);
          }
          else
          {
                C[kn][0] = C_Exato(0,t);
                C[kn][nelems-1] =  C_Exato(xf,t);
          }
}


//Passos de resolução dos métodos
void passos()
{
          formGrafico->lblTempo->Caption = FloatToStrF(t,ffGeneral,5,2);
          fprintf(arq, "tempo = %.3f \n", t);
```

```
        fprintf(arq, "----------------\n");
        if (idMetodo == 1)
      {
              cond_contorno(k1);
              CalcNovoTempoExplicito();
              grafico(k1, t);
              k1 = k0;
            k0 = 1-k0;
        }
        else if (idMetodo == 3)
        {
              cond_contorno(k1);
              CalcNovoTempoExplicitoUpWind();
              grafico(k1, t);
              k1 = k0;
            k0 = 1-k0;
        }
        else if (idMetodo == 4)
        {
              cond_contorno(k0);
              CalcNovoTempoImplicitoUpWind();
              grafico(k0, t);
        }
        else if (idMetodo == 2)
        {
              cond_contorno(k0);
              CalcNovoTempoImplicito();
              grafico(k0, t);
        }
        else if (idMetodo == 5)
        {
              cond_contorno(k1);
              CalcNovoTempoCrankNicolson();
              grafico(k0, t);
              k1 = k0;
              k0 = 1 - k0;
        }
        fprintf(arq, "----------------\n\n");
      t += dt;
}


//Recebe dados digitados pelo usuário
void BuscarDados()
{
        xf = StrToFloat(formGrafico->edL->Text);
        tf = StrToFloat(formGrafico->edTf->Text);
        dt = StrToFloat(formGrafico->edDt->Text);
        K = StrToFloat(formGrafico->edK->Text);
        V = StrToFloat(formGrafico->edV->Text);
        dx = StrToFloat(formGrafico->edDx->Text);
}


//Libera a memória alocada a vetores e matrizes
void LiberarMemoria(int metodo)
{
        if (metodo == 1)
        {
              C[0] = Liberar_vetor_real (C[0]);
              C[1] = Liberar_vetor_real (C[1]);
```

```
        }
        else if (metodo == 2)
        {
                C[0] = Liberar_vetor_real (C[0]);
                A = Liberar_matriz_real (nelems, A);
                ind = Liberar_vetor_inteiro (ind);
        }
        else if (metodo == 3)
        {
                C[0] = Liberar_vetor_real (C[0]);
                C[1] = Liberar_vetor_real (C[1]);
        }
        else if (metodo == 4)
        {
                C[0] = Liberar_vetor_real (C[0]);
                A = Liberar_matriz_real (nelems, A);
                ind = Liberar_vetor_inteiro (ind);
        }
        else if (metodo == 5)
        {
                C[0] = Liberar_vetor_real (C[0]);
                C[1] = Liberar_vetor_real (C[1]);
                A = Liberar_matriz_real (nelems, A);
                ind = Liberar_vetor_inteiro (ind);
        }
}

/* Arredonda float para inteiro mais proximo */
int arredonda(float x)
{
      if ((x - abs(x)) >= 0.5)
                return ceil(x);
      else
              return floor(x);
}

//---------------------------------------------------------------------------
__fastcall TformGrafico::TformGrafico(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::Button1Click(TObject *Sender)
{
        if(t>tf)
        {
                Relogio->Enabled = false;
                fclose(arq);
                btnSolNum->Enabled = true;
                ShowMessage("TEMPO MÁXIMO (T=" + FloatToStr(tf) + ")
ALCANÇADO.");
                formGrafico->Relogio->Destroying();
                LiberarMemoria(idMetodo);
        }
        else
                passos();
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::btnPararClick(TObject *Sender)
```

```
{
        Relogio->Enabled = false;
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::btnMenuMetodosClick(TObject *Sender)
{
        Relogio->Enabled = false;
        formMetodo->Show();
        Close();
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::btnSairClick(TObject *Sender)
{
        LiberarMemoria(idMetodo);
        Relogio->Enabled= false;
        exit(1);
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::btnContinuarClick(TObject *Sender)
{
        Relogio->Enabled = true;
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::btnGerarClick(TObject *Sender)
{
        k0 = 0; k1 = 1; t = xf = tf = dt = K = V = dx = 0;
        btnSolNum->Enabled = false;
        Relogio->Enabled = false;
        arq = fopen("solucao.txt","w");
        fprintf(arq, "tempo = %.3f \n", t);
        fprintf(arq, "----------------\n");

        bool TodosCamposPreenchidos = VerificarCamposPreenchidos();
        if (TodosCamposPreenchidos)
        {
             LiberarMemoria(idMetodo);

             BuscarDados(); //Recebe os dados do usuário

             //Verifica se a divisão é inteira
             Tnum num = (xf - xi)/dx;
             if ((num - arredonda(num)) != 0)
               if (MessageDlg("Atenção! Os valores escolhidos para L e Dx
resulta numa divisão não-inteira. Assim as condições de contorno não serão
obtidas corretamente.\nClique OK para continuar ou Cancel para escolher
outros valores.", mtWarning, TMsgDlgButtons() << mbOK << mbCancel, 0) ==
mbOK)
                  return;

             //Calcula o número de nós
             nelems = (arredonda((xf - xi)/dx))+1;
             if(nelems > 105)
             {
                       if (MessageDlg("Atenção! Os valores escolhidos para
L e Dx podem fazer com que o programa trave.\nClique OK para continuar ou
Cancel para escolher outros valores.", mtWarning, TMsgDlgButtons() << mbOK
<< mbCancel, 0) == mbOK)
```

```
                        return;
            }

            //  Dependendo do Método, aloca matrizes e vetores necessários
e inicializa-os.
            if (formGrafico->lblTitulo->Caption == "MÉTODO EXPLÍCITO - DIF.
CENTRAL")
            {
                  idMetodo = 1;       // Identificação do método

                  C[0] = Alocar_vetor_real(nelems);    // Vetores de
resultado
                  C[1] = Alocar_vetor_real(nelems);
                  Inic_C();     // Inicializa vetor de resultados
            }
            else if (formGrafico->lblTitulo->Caption == "MÉTODO IMPLÍCITO
- DIF. CENTRAL")
            {
                  idMetodo = 2;  // Identificação do método

                  C[0] = Alocar_vetor_real(nelems);    //Vetor de
resultados

                  A = Alocar_matriz_real(nelems, nelems);      //Matriz
de coeficientes
                  A = PreencherMatriz(idMetodo, A);    //Preenche a
Matriz

                  Inic_C();     //Inicializa vetor de resultados

                  ind = Alocar_vetor_inteiro(nelems); //Vetor utilizado
na Fatoração LU
                  LU_fatoracao(A, ind, nelems);
            }
             else if (formGrafico->lblTitulo->Caption == "MÉTODO
EXPLÍCITO-UPWIND")
            {
                   idMetodo = 3;       // Identificação do método

                   C[0] = Alocar_vetor_real(nelems);     //Vetores de
Resultados
                   C[1] = Alocar_vetor_real(nelems);

                   Inic_C();    //Inicializa vetor de Resultados
            }

            else if (formGrafico->lblTitulo->Caption == "MÉTODO IMPLÍCITO-
UPWIND")
            {
                   idMetodo = 4;  //Identificação do Método

                   C[0] = Alocar_vetor_real(nelems); //Vetor de
Resultados

                   A = Alocar_matriz_real(nelems, nelems);  //Matriz de
Coeficientes
                   A = PreencherMatriz(idMetodo, A);  //Preenche a Matriz
                   Inic_C();   //Inicializa o vetor de resultados
                   ind = Alocar_vetor_inteiro(nelems);   //Vetor usado na
Fatoração LU
                   LU_fatoracao(A, ind, nelems);
```

```
            }
            else if (formGrafico->lblTitulo->Caption == "MÉTODO CRANK-
NICOLSON")
            {
                    idMetodo = 5;  //Identificação do Método

                    C[0] = Alocar_vetor_real(nelems);    //Vetores de
Resultados
                    C[1] = Alocar_vetor_real(nelems);
                    A = Alocar_matriz_real(nelems, nelems);      //Matriz
de Coeficientes
                    A = PreencherMatriz(idMetodo, A);    //Preenche a
Matriz
                    Inic_C();  //Inicializa o vetor de resultados

                    ind = Alocar_vetor_inteiro(nelems);    //Vetor usado
na Fatoração LU
                    LU_fatoracao(A, ind, nelems);
            }

            t = ti + dt;
            Relogio->Enabled = true;
        }

        else
                ShowMessage ("Atenção, todos os campos devem ser
preenchidos.");
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::FormActivate(TObject *Sender)
{
        LiberarMemoria(idMetodo);
        xi = 0; ti = 0;
        x = xi; t = ti;
        nelems = 0;
        Relogio->Interval = 26;
        lblVelGrafico->Caption = 15;
        formGrafico->btnMais->Enabled = true;
        formGrafico->btnMenos->Enabled = true;
        formGrafico->Series1->Clear();
        formGrafico->Series2->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::btnMenuPrincipalClick(TObject *Sender)
{
        Relogio->Enabled= false;
        formInicio->Show();
        Close();
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::btnMenuEquacoesClick(TObject *Sender)
{
        Relogio->Enabled= false;
        formEquacao->Show();
        Close();
}
//---------------------------------------------------------------------------
```

```
void __fastcall TformGrafico::btnMaisClick(TObject *Sender)
{
        int vel = StrToInt(lblVelGrafico->Caption);
        Relogio->Interval -= 5;
        vel+=1;
        lblVelGrafico->Caption = IntToStr(vel);
        if(vel == 20)
                btnMais->Enabled = false;
        btnMenos->Enabled = true;
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::btnMenosClick(TObject *Sender)
{
        int vel = StrToInt(lblVelGrafico->Caption);
        Relogio->Interval += 5;
        vel-=1;
        lblVelGrafico->Caption = IntToStr(vel);
        if(vel == 1)
                btnMenos->Enabled = false;
        btnMais->Enabled = true;
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::edLKeyPress(TObject *Sender, char &Key)
{
        if ((Key == ',') || (Key == '.'))
                Key = DecimalSeparator;
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::edTfKeyPress(TObject *Sender, char &Key)
{
        if ((Key == ',') || (Key == '.'))
                Key = DecimalSeparator;
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::edDtKeyPress(TObject *Sender, char &Key)
{
        if ((Key == ',') || (Key == '.'))
                Key = DecimalSeparator;
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::edDxKeyPress(TObject *Sender, char &Key)
{
        if ((Key == ',') || (Key == '.'))
                Key = DecimalSeparator;
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::edVKeyPress(TObject *Sender, char &Key)
{
        if ((Key == ',') || (Key == '.'))
                Key = DecimalSeparator;
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::edKKeyPress(TObject *Sender, char &Key)
{
```

```
         if ((Key == ',') || (Key == '.'))
                Key = DecimalSeparator;
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::FormClose(TObject *Sender,
      TCloseAction &Action)
{
        LiberarMemoria(idMetodo);
}
//---------------------------------------------------------------------------

void __fastcall TformGrafico::btnSolNumClick(TObject *Sender)
{
        Relogio->Enabled = false;
        String arq_txt = ExtractFilePath(ParamStr(0)) + "\solucao.txt";
        TVariant Endereco = {arq_txt};
        formSolucao->pagSolucao->Navigate(Endereco);
        formSolucao->Show();
}
//---------------------------------------------------------------------------
```

Unit uSolucao.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "uSolucao.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "SHDocVw_OCX"
#pragma resource "*.dfm"
TformSolucao *formSolucao;
//---------------------------------------------------------------------------
__fastcall TformSolucao::TformSolucao(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TformSolucao::btnFecharClick(TObject *Sender)
{
        Close();
}
//---------------------------------------------------------------------------
```

Unit uInfo.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "uInfo.h"
//---------------------------------------------------------------------------
```

```
#pragma package(smart_init)
#pragma resource "*.dfm"
TformInfo *formInfo;
//---------------------------------------------------------------------------
__fastcall TformInfo::TformInfo(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TformInfo::SpeedButton1Click(TObject *Sender)
{
        Close();
}
//---------------------------------------------------------------------------
```