Um Método *Streamline Diffusion* Descontínuo Aplicado a um Modelo de Espalhamento de Manchas de Petróleo

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida por Márcio Rodolfo Fernandes e aprovada pela comissão julgadora.

Campinas, 5 de dezembro de 2003.

Prof. Dr. José Luiz Boldrini
Orientador

Prof. Dr. Petronio Pulino
Co-orientador

Banca Examinadora
1 João Frederico da Costa Azevedo Meyer - IMECC/UNICAMP
2 Murilo Francisco Tomé - ICMSC/USP
3 José Alberto Cuminato - ICMSC/USP
4 Luis Carlos de Castro Santos - IME/USP

Tese apresentada ao Instituto de Matemática, Estatística e Computação Científica, UNICAMP, como requisito parcial para a obtenção do Título de Doutor em Matemática Aplicada.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Fernandes, Márcio Rodolfo

F391m Um método Streamline Diffusion descontínuo aplicado a um modelo de espalhamento de manchas de petróleo/Márcio Rodolfo Fernandes -- Campinas, [S.P. :s.n.], 2003.

Orientador : José Luiz Boldrini

Co-orientador: Petronio Pulino.

Tese (Doutorado) - Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1.Equações diferenciais – Soluções numéricas. 2.Método de elementos finitos. 3.Poluição marinha por óleo. 4.Água – Poluição por petróleo. I. Boldrini, José Luiz. II. Pulino, Petronio. III. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Computação Científica. IV. Título.

.

Aos meus pais Antonio *(in Memoriam)* e Conceição,
à minha esposa Cristiane,
à minha irmã Morgana,
com todo o carinho.

Agradecimentos

- Aos meus orientadores Boldrini e Petronio, sempre prontos a ajudar.
- À minha esposa Cristiane, pela paciência.
- Ao Departamento de Matemática da UFSC, por ter propiciado o meu afastamento.
- À CAPES, pelo suporte financeiro.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Notações

$a(\cdot,\cdot)$	:	forma bilinear
$\alpha(x,t)$	:	coeficiente de difusão
$b(\cdot)$	:	forma linear
$\mathcal{B}(K)$	:	espaço das funções bolha
$\vec{\beta}(x,t)$	:	coeficiente de advecção
D	:	tensor de deformação
δ	:	parâmetro de estalização do método *Streamline Diffusion*
$\partial\Omega$	:	bordo do domínio
$\partial\Omega_+ = \{\ x \in \partial\Omega\ /\ \vec{\eta}(x)\cdot\vec{\beta}(x) \geq 0\ \}$	:	bordo de saída do fluxo
$\partial\Omega_- = \{\ x \in \partial\Omega\ /\ \vec{\eta}(x)\cdot\vec{\beta}(x) < 0\ \}$	:	bordo de entrada do fluxo
div	:	operador divergente
∇	:	operador gradiente
$h = \max\{\ h_K\ /\ K \in T_h\ \}$	:	diâmetro do elemento K
$\mathcal{H}^1(\Omega)$	:	espaço das funções $f \in \mathcal{L}_2(\Omega)$ com $f^{'} \in \mathcal{L}_2(\Omega)$
H_0^1	:	espaço de funções
H_g^1	:	espaço de funções
$(H, \langle\cdot,\cdot\rangle_H)$	:	espaço de Hilbert
$I = (0,T]$	:	intervalo de tempo
$I = [0,T]$	:	intervalo de tempo
$I_n = (t_{n-1}, t_n)$	:	subintervalos da partição Π
K	:	elemento da triangularização T_h
$k_n = t_n - t_{n-1}$	:	passo local no tempo
L	:	gradiente do vetor $\vec{u}$
$\mathcal{L}_2(\Omega)$	:	espaço das funções quadrado-integráveis
$L_r(T_h)$	:	espaço dos polinômios de Lagrange de grau r
Δ	:	operador Laplaciano
μ	:	viscosidade de um fluido
M_h	:	espaço de aproximação
$\max\{u,v\}$	:	máximo entre u e v

$\vec{\eta}(x)$	:	vetor normal unitário
$\|\cdot\|$	:	norma do espaço $\mathcal{L}_2(\Omega)$
$\|\cdot\|_H$	:	norma do espaço de Hilbert H
Ω	:	domínio
P	:	operador diferencial
$\mathcal{P}_r(K)$	:	espaço dos polinômios de grau menor ou igual a r, sobre K
$P^{'}(u)[v] = \frac{d}{d\lambda}\{P(u+\lambda v)\}\mid_{\lambda=0}$	:	derivada de P na direção de v
PGDT	:	Petrov-Galerkin Descontínuo no Tempo
PGDT-CM	:	PGDT com controle de massa
PGDT-CM-SUB	:	PGDT com controle de massa e localização de subdomínios
Π	:	partição de I
$\langle\cdot,\cdot\rangle$	:	produto interno do espaço $\mathcal{L}_2(\Omega)$
$\langle\cdot,\cdot\rangle_H$	:	produto interno do espaço de Hilbert H
$\cdot$	:	produto escalar de $I\!R^n$
ρ	:	densidade de um fluido
$I\!R^n$	:	espaço vetorial real n-dimensional
Re	:	Número de Reynolds
σ_1	:	tensor de *stress*
$\sigma(u)$	:	coeficiente de absorção
SD	:	método *Streamline Diffusion*
S_n	:	faixa de tempo de I
SOR	:	método de relaxações sucessivas
$T_h = \{K\}$	:	triangularização de Ω
$u_{\vec{\beta}} = \vec{\beta}\cdot\nabla u$	:	derivada na direção de $\vec{\beta}$
$u_{\vec{\beta}\vec{\beta}} = \vec{\beta}\cdot\nabla(\vec{\beta}\cdot\nabla u)$	:	segunda derivada na direção de $\vec{\beta}$
V_h	:	espaço de aproximação
V_h^b	:	espaço de aproximação
V_h^g	:	espaço de aproximação
V_h^{0n}	:	espaço de aproximação

Índice

Introdução

O transporte e o armazenamento de petróleo têm um risco significativo de vazamentos acidentais com grandes conseqüências. Tal risco torna-se maior em zonas costeiras e especialmente indesejável em regiões onde indústrias petroquímicas coexistam com atividades pesqueiras e/ou turísticas.

Acidentes com impactos ambientais deste tipo têm ocorrido ao longo do tempo com enormes conseqüências econômicas. Assim, o planejamento de políticas de prevenção e minimização dos efeitos de tais vazamentos constitui um dos desafios da comunidade científica, sendo o conhecimento do movimento e da área atingida por essas manchas de petróleo após um vazamento, parte das informações necessárias para a decisão das ações a serem tomadas.

A realização deste tipo de previsão passa por alguns estágios: modelagem matemática do processo em questão; obtenção de soluções numéricas para as equações que descrevem o modelo; comparação com dados reais de derramamentos visando a averiguação da qualidade das aproximações numéricas, bem como do modelo.

Segundo Fay [13], a evolução do processo se divide em três fases, de acordo com o tipo de forças que governam o comportamento do petróleo sobre a água. Inicialmente, o regime dominante é o gravitacional-inercial que dura por volta de uma hora ou mais, dependendo do óleo, e se caracteriza pelo fato do petróleo ainda não ter se misturado à água. Em seguida, temos o regime gravitacional-viscoso no qual o deslocamento da mancha se deve principalmente aos efeitos combinados dos ventos e correntes marítimas, durando por volta de duas semanas, dependendo do volume do derrame. Finalmente, ocorre o regime dominado pela viscosidade e pela tensão superficial, em que a mancha já não pode ser vista a olho nu, estando o óleo totalmente misturado com a água e podendo durar meses.

A fase representada pelo regime gravitacional-viscoso pode ser descrita por uma equação não-linear de advecção-difusão, bidimensional nas variáveis espaciais, juntamente com condições iniciais e de contorno adequadas. Este modelo se deve a Benqué [4] tendo sido explorado por Cuesta [10] através de simulações numéricas, sendo esta a principal motivação para a realização do presente trabalho.

A equação que descreve o deslocamento da mancha é dada por

$$\frac{\partial u}{\partial t} - c\Delta\left(u^3\right) + div(u\vec{\beta}) = f.$$

Nesta, u representa a altura da mancha medida em relação ao nível da água, c é um coeficiente utilizado para modificar a intensidade do coeficiente de difusão e para mudar o caráter da equação, f é uma fonte de poluentes e $\vec{\beta}$ é um vetor que combina os efeitos de arrasto dos ventos, correntes e marés.

Com o grande desenvolvimento experimentado pelos métodos numéricos, discussões sobre a eficiência dos métodos de diferenças finitas (MDF) e elementos finitos (MEF) tornaram-se comuns. Um breve histórico sobre tais métodos é apresentado por Maliska em [43], visando elucidar alguns pontos importantes. Segundo este autor, o MDF sempre foi utilizado por analistas da área de escoamento de fluidos, enquanto o MEF foi empregado para problemas de elasticidade. Problemas, do ponto de vista físico, totalmente diferentes. Os de escoamento são altamente não-lineares (Navier-Stokes), enquanto os de elasticidade não apresentam os termos convectivos, não-lineares, e assemelham-se a problemas puramente difusivos de transferência de calor. Foi natural, portanto, o fato de os pesquisadores do MDF terem se concentrado na tentativa de dominar as não-linearidades dos termos convectivos e no problema de acoplamento entre as equações, dificuldades não encontradas em problemas de elasticidade. Por muito tempo, foi deixado para segundo plano o problema do tratamento de geometrias complexas, e o MDF teve todo o seu desenvolvimento baseado nos sistemas de coordenadas ortogonais. Por esta razão, muitas pessoas ainda vinculam o MDF a malhas cartesianas, equivocadamente, uma vez que ele pode ser aplicado a qualquer tipo de malha, mesmo não-estruturada.

Por outro lado, o MEF sempre teve a vantagem de usar malhas não-estruturadas, o que lhe permite resolver problemas em geometrias complexas. O MEF não teve grande penetração na área de fluidos por muito tempo, porque se acreditava que a equação diferencial a ser resolvida necessitava de um princípio variacional para que o método pudesse ser aplicado. Com isso, a aplicação de tal método em fluidos foi retardada.

Até o início da década de 70, tinha-se, portanto, o MDF com grande experiência na área de fluidos, mas sem habilidades para tratar geometrias complexas, e o MEF, hábil no tratamento da geometria, mas sem ferramentas para tratar os termos convectivos presentes nas equações do movimento. Mesmo suplantando a questão do princípio variacional, através do uso do método de Galerkin e outras variantes, o MEF não teve sucesso imediato em problemas de fluidos, uma vez que o método de Galerkin (que é equivalente ao uso de diferenças centrais no MDF) é adequado apenas para problemas puramente difusivos, produzindo instabilidade em problemas com convecção dominante. Este e outros problemas similares motivaram pesquisas para o aprimoramento do método dos volumes finitos (MVF), no qual as equações aproximadas são obtidas através de balanços de conservação da propriedade envolvida (massa, quantidade de movimento, etc.) no volume elementar. A observação do caráter físico de cada termo da equação diferencial permitiu que métodos mais robustos fossem desenvolvidos. Grande parte dos analistas envolvidos com o MDF passaram a usar o MVF. Importantes desenvolvimentos foram realizados no MVF, mas ainda em coordenadas ortogonais. Em meados da década de 70, em razão do aparecimento de computadores mais

velozes, os sistemas de coordenadas ortogonais começaram a ceder espaço para os sistemas de coordenadas generalizadas coincidentes com a fronteira do domínio, e o MVF passou a resolver problemas de fluidos em geometrias irregulares. Paralelamente, o MEF passou a empregar outras funções de interpolação para permitir o tratamento adequado dos termos convectivos.

No panorama atual, observa-se que ambos os métodos (MEF e MVF) estão resolvendo problemas altamente convectivos, inclusive com ondas de choque, em geometrias arbitrárias, mostarndo que existe entre eles uma forte semelhança em termos de generalidade. Do ponto de vista matemático não poderia ser diferente, uma vez que o MDF, o MEF e o MVF são derivados do método dos resíduos ponderados, empregando-se diferentes funções peso. Logo, não existe sentido em argumentar que um determinado método é sempre superior a outro, visto que eles são derivados do mesmo princípio e diferem apenas na forma de minimização escolhida. O que se tem, na prática, são diferentes graus de experiência dos diversos métodos, para diferentes problemas.

A nossa preferência pessoal pela utilização do MEF nesse trabalho, é justificada pelas características das regiões onde os derramamentos de óleo ocorrem, regiões costeiras com geometria irregular que, devido à nossa formação, tornam-se mais facilmente tratáveis com as malhas não-estruturadas desse tipo de método.

Como já enfatizamos, problemas de advecção-difusão com caráter predominantemente hiperbólico, isto é, com coeficiente difusivo pequeno em relação ao coeficiente advectivo, não podem ser aproximados por métodos numéricos usuais pela possível presença de ondas de choque - regiões de descontinuidade da solução - ou de regiões de transição muito rápida (gradientes com grande intensidade).

De fato, a equação de nosso interesse pode ser mais complicada ainda. Partindo de uma condição inicial com suporte compacto, o termo difusivo, por depender de u, será muito pequeno em algumas regiões do domínio e se anulará fora do suporte da solução, tornando o problema puramente hiperbólico nestas regiões, não conhecidas *a priori*.

O uso de diferenças centrais no MDF funciona muito bem para a discretização de operadores elíticos, exibindo estabilidade e boa ordem de aproximação, sendo bastante eficientes no tratamento do termo difusivo da equação. Porém o mesmo não ocorre com o termo advectivo, havendo perda de estabilidade e precisão, inclusive nos casos em que se recorre ao refinamento da malha.

Uma possibilidade de correção da falta de estabilidade numérica é a utilização de combinações ponderadas de esquemas de diferenças não-centradas para o tratamento do termo advectivo. Todavia, esta técnica conhecida como *upwind* elimina as oscilações espúrias introduzindo difusão artificial excessiva, causando demasiada atenuação dos choques. Somente para problemas simples, em geral unidimensionais e com coeficientes constantes, é possível o cálculo exato da ponderação responsável pela introdução da quantidade mínima de difusão artificial, suficiente para eliminar oscilações indesejáveis.

A aplicação do MEF para a resolução de problemas de advecção-difusão é uma outra

possibilidade. Como já ressaltamos, tal método tem a vantagem de se adaptar de forma mais fácil às complexidades das geometrias costeiras, embora tais geometrias também possam ser tratadas pelo MDF e pelo MVF, pela utilização de sistemas de coordenadas generalizadas coincidentes com a fronteira do domínio. Assim como no caso do MDF, a utilização do método de Galerkin usual não conduz a bons resultados. O predomínio do termo advectivo se traduz na perda da coercividade do operador discretizado, gerando oscilações.

Para a correção deste defeito, vários autores tentaram produzir efeitos *upwind* com o MEF, encontrando dificuldades semelhantes ao caso da utilização das diferenças finitas.

Em 1976, Christie [8] propôs a alteração do espaço das funções-teste, construídas somando-se funções na direção do fluxo, às funções de interpolação. Este procedimento, conhecido como Petrov-Galerkin, elimina as oscilações, mas apresenta excessiva difusão na direção transversal ao fluxo. Além disso, uma das dificuldades para a sua implementação é o fato de que a construção das funções-teste é bastante trabalhosa no caso de escoamentos complexos.

Como uma alternativa, o método *Streamline Diffusion* foi proposto por Hughes e Brooks [32] em 1979, de forma a não apresentar este tipo dificuldade. Tal método, que combina precisão e estabilidade, utiliza uma formulação variacional do tipo Petrov-Galerkin, com as funções-teste construídas pelas funções de interpolação usuais mais uma perturbação automática na direção do fluxo. Inicialmente, esse método foi introduzido em problemas estacionários e lineares de advecção-difusão, tendo sua análise matemática sido iniciada por Johnson e Nävert [33] e continuada para problemas de evolução por Johnson, Nävert, Pitkaranta e Saranen nos anos seguintes ([44], [34] e [35]).

Para problemas não-lineares e dependentes do tempo, transporte advectico não-linear, Johnson e Szepessy [38] analisaram tal método aplicado à equação de Burger enquanto Johnson e Saranen [35], além de Franca e Frey [17], consideraram as equações de Euler e Navier-Stokes. Além disso, em [41], podemos encontrar uma coletânea de resultados compilada por Johnson, sobre métodos h-adaptativos para equações lineares elíticas, parabólicas e hiperbólicas, baseados em estimadores de erro *a posteriori*.

Ainda baseado na alteração do espaço das funções-teste, surgiu a idéia de introduzir funções conhecidas como bolhas, em cada elemento, resultando num método que, sob certas condições (ver [18]), é equivalente ao método *Streamline Diffusion*. A introdução de tais bolhas ajuda na estabilização do operador de advecção e foi proposta na década de 70, inicialmente para a discretização das equações de Navier-Stokes [58], tendo sido retomada no início dos anos 90 para problemas de advecção-difusão [21].

Partindo da observação de como a introdução de bolhas altera a formulação variacional do problema, seguiram-se vários artigos sobre métodos estabilizados de elementos finitos, isto é, métodos que exploram modificações no problema variacional. Dentre eles, destacamos os trabalhos de Franca ([19], [21], [22] e [23]).

Outra possibilidade de método para problemas evolutivos de transporte é conhecido como Semi-Lagrangiano e combina elementos finitos (ou diferenças finitas) com integração sobre as curvas características da equação, tendo sido proposto em 1972 por Robert, Henderson

e Turnbull [50] para problemas metereológicos. Em [51], Robert sugeriu alterações no algoritmo de forma a conferir estabilidade também para passos maiores no tempo enquanto em [48] o método foi aplicado e analisado para problemas de advecção-difusão.

Uma coletânea sobre algoritmos do mesmo tipo do método Semi-Lagrangiano pode ser encontrada em [12].

Neste trabalho, que se divide na forma descrita a seguir, nosso objetivo é isolar o estágio referente à obtenção de soluções numéricas confiáveis para as equações que descrevem o modelo de espalhamento de manchas de petróleo. Em trabalhos futuros, pretendemos passar ao estágio de simulação com dados reais, como no artigo de Cuesta [10] que apresenta um código computacional para este problema, que usa o método de diferenças finitas com estratégia *upwind*, comparando seus resultados com dados de um acidente com o petroleiro Amoco Cadiz, ocorrido na Europa em 1978.

No Capítulo 1, recordamos aspectos teóricos elementares sobre as equações de advecção-difusão, tanto num caso de difusão linear quanto num caso não-linear no termo difusivo, explorando algumas de suas propriedades. Dentre elas, destacamos que, na equação que define o modelo de espalhamento, a integral de u sobre o domínio é totalmente determinada pela correspondente integral de f, o que confere ao problema uma característica de balanço (conservação) da massa total da equação. Tais propriedades deveriam estar presentes nas soluções numéricas.

No Capítulo 2, exploramos alguns dos métodos estabilizados de elementos finitos mais conhecidos, aplicando-os a tal modelo pela associação com métodos de discretização da variável temporal (Euler) e de linearização (SOR e Newton), assim como métodos que discretizam tempo e espaço de uma só vez (Petrov Galerkin Descontínuo no Tempo) [35] e ainda o método Semi-Lagrangiano, bastante utilizado em pesquisas meteorológicas [48].

A maioria dos autores sugere a utilização do método Semi-Lagrangiano em associação com métodos de diferenças finitas, entretanto, pela própria característica de nosso trabalho, utilizamos elementos finitos. Assim, para a sua implementação, surge uma dificuldade adicional: a localização de um dado ponto entre os triângulos da malha. Para isso, também apresentamos uma estratégia de localização.

Neste capítulo ainda, analisamos a quantidade de difusão artificial acrescentada por cada um dos métodos e a qualidade de suas aproximações numéricas, pela comparação entre seus resultados através de um problema com solução conhecida. O objetivo do capítulo é o de identificar os principais defeitos de cada um destes métodos quando aplicados ao problema em questão.

No Capítulo 3, encontramos duas contribuições originais deste trabalho.

A primeira delas está diretamente ligada ao balanço da massa total da equação. Esta é uma propriedade obtida *a priori* que geralmente é perdida pelas aproximações, devido ao uso

de integrações numéricas aliadas às características geométricas do problema. Sendo assim, desenvolvemos uma maneira de forçar a solução aproximada pelo método Petrov-Galerkin Descontínuo no Tempo, que obteve o melhor desempenho em nossos testes, a obedecer tal propriedade. Com isso, obtivemos um método eficiente para o problema não-linear de espalhamento, mesmo em malhas não muito finas.

A segunda contribuição consiste no aumento significativo da eficiência do método Petrov-Galerkin Descontínuo no Tempo, pela reestruturação de seu algoritmo. Levando em consideração o suporte compacto da solução da equação que define o modelo de espalhamento, desenvolvemos um mecanismo de localização que nos permite acompanhar a solução ao longo do tempo e resolver o problema apenas numa parte do domínio computacional, isto é, num subdomínio que contenha o suporte da solução. Com isto, em cada nível de tempo, o sistema linear resultante apresenta sensível diminuição de sua ordem e, conseqüentemente, do tempo de execução do algoritmo.

Além disso, ainda neste capítulo, modificamos o modelo de espalhamento pela inclusão de um termo não-linear de evaporação, diretamente proporcional à área definida pela mancha de petróleo. Desta forma, destacamos a importância do conhecimento preciso do suporte da mancha, propriedade obtida pelo método Petrov-Galerkin Descontínuo no Tempo, visando não interferir no processo de evaporação do petróleo. Tal modificação no modelo, tem por objetivo tornar o problema mais próximo da realidade.

O objetivo do Capítulo 4 é o de apontar para simulações mais realistas, considerando problemas de espalhamanto de manchas de petróleo em situações mais gerais. Para isto, incorporamos aos programas computacionais o cálculo do escoamento incompressível utilizando as equações de Navier-Stokes.

Desta forma, iniciamos este capítulo apresentando um simulador para a resolução de tais equações e em seguida nos dedicamos à validação de seu código pela aplicação em problemas clássicos da literatura - escoamento entre placas paralelas e cavidade quadrada.

Na parte final do capítulo, apresentamos simulações numéricas de derramamentos de petróleo tendo como cenário a Baía de Guanabara, no Estado do Rio de Janeiro.

No Apêndice A, damos ênfase aos aspectos computacionais relativos aos métodos anteriores, expondo e explicando cada uma das subrotinas que compõem a implementação do método Petrov-Galerkin Descontínuo no Tempo com Controle de Massa, com o objetivo de tornar este capítulo um manual de utilização do software que desenvolvemos.

Capítulo 1

Aspectos Teóricos Elementares das Equações de Advecção-Difusão

1.1 O Problema Linear

Vamos considerar o seguinte problema linear de advecção-difusão:

$$\frac{\partial u(x,t)}{\partial t} - c(x,t)\Delta(u(x,t)) + div(\vec{\beta}(x,t)u(x,t)) + \sigma(x,t)u(x,t) = f(x,t) \ \ , \ \ \Omega \times I, \quad (1.1)$$

sendo $u : \Omega \times I \rightarrow I\!R$ a função incógnita do problema representando uma determinada quantidade, por exemplo, onde Ω é um subconjunto limitado de $I\!R^n$ e $I = (0, T]$ é um intervalo de tempo. $\vec{\beta}(x,t) = (\beta_1(x,t), \beta_2(x,t), \cdots, \beta_n(x,t))$ é um dado campo de velocidades, responsável pelo transporte advectivo e $c(x,t)$ é uma função não-negativa que representa o coeficiente de difusão. A função σ mede a taxa de crescimento (ou decaimento) da grandeza física representada por u e a função f é uma fonte (ou sorvedouro) desta grandeza.

A equação (1.1) é, do ponto de vista teórico, de natureza hiperbólica-parabólica, dependendo dos valores assumidos por $c(x,t)$. Em regiões onde $c(x,t) > 0$, o problema é parabólico, enquanto que em regiões onde $c(x,t) \equiv 0$, o problema é hiperbólico. Assim, se $c(x,t)$ puder assumir valores positivos ou nulos, o problema, mesmo sendo linear, se torna complexo devido à interação entre as regiões parabólicas e hiperbólicas (as condições de contorno adequadas podem se tornar complexas).

Além disso, do ponto de vista qualitativo e também numérico, mesmo no caso em que $c(x,t) > 0$ o comportamento da solução da equação pode ter aspecto predominantemente hiperbólico (pelo menos em grandes regiões do domínio) quando $c(x,t)$ for relativamente pequeno em comparação com $\mid \vec{\beta}(x,t) \mid$, isto é, em problemas físicos em que haja predominância

do caráter advectivo em relação ao difusivo.

A seguir nesta seção, para facilitar a exposição dos comportamentos característicos em cada caso, assumiremos que $c(x,t)$ seja uma função constante, não-negativa, mas relativamente pequena.

Em particular, iniciaremos com $c \equiv 0$, isto é, com a seguinte equação puramente hiperbólica:

$$\frac{\partial u(x,t)}{\partial t} + div(\vec{\beta}(x,t)u(x,t)) + \sigma(x,t)u(x,t) = f(x,t) \ \ , \ \ \Omega \times I,$$

ou, lembrando que

$$div(\vec{\beta}(x,t)u(x,t)) = \vec{\beta}(x,t) \cdot \nabla u(x,t) + u(x,t) div(\vec{\beta}(x,t)),$$

com

$$\frac{\partial u(x,t)}{\partial t} + \vec{\beta}(x,t) \cdot \nabla u(x,t) + \gamma(x,t)u(x,t) = f(x,t) \ \ , \ \ \Omega \times I,$$

onde $\gamma(x,t) = \sigma(x,t) + div(\vec{\beta}(x,t))$.

Assumiremos também estarmos no caso estacionário, ou seja, u e todos os termos da equação independentes do tempo:

$$\vec{\beta}(x) \cdot \nabla u(x) + \gamma(x)u(x) = f(x) \ \ , \ \ x \in \Omega. \tag{1.2}$$

As linhas de fluxo correspondentes ao campo de velocidades $\vec{\beta}$, também chamadas de curvas características, são dadas pelas curvas parametrizadas por $s \in I\!R$

$$x(s) = (x_1(s), x_2(s), \cdots, x_n(s)),$$

soluções do seguinte sistema de equações diferenciais ordinárias:

$$\frac{dx_i}{ds} = \beta_i(x) \ \ , \ \ i = 1, 2, \cdots, n.$$

Assumindo que $\vec{\beta}$ seja Lipschitz-contínua, isto é,

$$\left|\vec{\beta}(x) - \vec{\beta}(y)\right| \leq C\,|x-y| \quad \forall x, y \in \Omega,$$

para alguma constante C, podemos garantir que existe uma única característica passando por $\tilde{x} \in \Omega$, ou seja, uma única função $x(s)$ satisfazendo

$$\frac{dx_i}{ds} = \beta_i(x) \quad , \quad i = 1, 2, \cdots, n$$

e a condição inicial

$$x(0) = \tilde{x}.$$

Se $x(s)$ é uma característica, então pela regra da cadeia

$$\frac{d}{ds}(u(x(s))) = \sum_{i=1}^{n} \frac{\partial u}{\partial x_i}\frac{dx_i}{ds} = \sum_{i=1}^{n} \frac{\partial u}{\partial x_i}\beta_i = \vec{\beta} \cdot \nabla u.$$

Desta forma, (1.2) pode ser reescrita como

$$\frac{d}{ds}(u(x(s))) + \gamma u(x(s)) = f(x(s)). \tag{1.3}$$

Assim, ao longo de cada característica a equação diferencial parcial (1.2) é reduzida a uma equação diferencial ordinária. Se u for conhecida em algum ponto de uma dada característica $x(s)$, podemos determiná-la sobre toda esta característica integrando (1.3).

Vamos dividir a fronteira de Ω da seguinte maneira

$$\partial\Omega_+ = \{\ x \in \partial\Omega \ / \ \vec{\eta}(x) \cdot \vec{\beta}(x) \geq 0\ \},$$

também conhecido como bordo de saída do fluxo e

$$\partial\Omega_- = \{\ x \in \partial\Omega \ / \ \vec{\eta}(x) \cdot \vec{\beta}(x) < 0\ \},$$

que recebe o nome de bordo de entrada do fluxo, sendo $\vec{\eta}(x)$ o vetor unitário normal no ponto $x \in \partial\Omega$ e "$\cdot$" o produto escalar do $I\!R^n$.

Portanto, para que não tenhamos problemas na determinação de u, as condições de fronteira sobre ela devem ser impostas apenas em $\partial\Omega_-$. Isto é, determinamos u num ponto qualquer do domínio integrando (1.3) ao longo da curva característica que passe por este ponto e se inicie em $\partial\Omega_-$. Em particular, isto significa que no problema (1.2), os efeitos se propagam ao longo das características. Portanto, se a condição de contorno for descontínua em algum ponto de $\partial\Omega_-$, esta descontinuidade será transportada para dentro do domínio Ω.

Para descrever o comportamento da solução no caso em que $c > 0$ é relativamente pequeno, vamos considerar o seguinte problema estacionário $(1.4) - (1.5)$:

$$-c\Delta(u(x)) + \vec{\beta}(x) \cdot \nabla u(x) + u(x) = f(x) \ \ , \ \ x \in \Omega \tag{1.4}$$

$$u(x) = g(x) \ , \ x \in \ \partial\Omega, \tag{1.5}$$

com $div(\vec{\beta}) = 0$ e $\Omega \subset I\!R^2$, e compará-lo com o problema obtido fazendo-se $c = 0$:

$$\vec{\beta}(x) \cdot \nabla u(x) + u(x) = f(x) \ \ , \ \ x \in \Omega \tag{1.6}$$

$$u(x) = g(x) \ , \ x \in \ \partial\Omega_-. \tag{1.7}$$

Como já sabemos, a solução de $(1.6) - (1.7)$ pode ser descontínua se g for descontínua. Isto não ocorre com a solução de $(1.4) - (1.5)$. O coeficiente de difusão $c > 0$ transforma o salto de descontinuidade numa região de transição rápida da solução, com largura $O(\sqrt{c})$ em torno da curva característica que contém o ponto de descontinuidade. Portanto, quanto menor c, mais rapidamente a solução irá variar. Além disso, se os valores de u calculados em $(1.6)-(1.7)$ não coincidirem com os valores prescritos de g na fronteira $\partial\Omega_-$, para o problema $(1.4) - (1.5)$, a solução deste problema apresentará uma "camada limite" com largura $O(c)$, porque u foi forçada a assumir tais valores.

Por fim, retornaremos ao problema dependente do tempo

$$\frac{\partial u(x,t)}{\partial t} + \vec{\beta}(x,t) \cdot \nabla u(x,t) + \gamma(x,t)u(x,t) = f(x,t) \ \ , \ \ \Omega \times I.$$

Se fizermos $x_0 = t$ e $\beta_0 = 1$, então esta equação pode ser reescrita como

$$\sum_{i=0}^{n} \frac{\partial u}{\partial x_i}\beta_i + \gamma u = f, \tag{1.8}$$

que é do mesmo tipo de (1.2) e cujo comportamento já foi analisado. Em particular, as características de (1.8) são curvas $(x(t), t)$ no espaço-tempo, onde $x(t)$ satisfaz

$$\frac{dx_i}{dt} = \beta_i(x,t) \ \ , \ \ i = 0, 1, 2, \cdots, n.$$

Assim, considerações análogas às anteriores podem ser feitas no caso dependente do tempo.

Na próxima seção, discutiremos algumas propriedades de um problema não-linear de advecção-difusão.

1.2 O Problema Não-Linear

Nesta seção e na seqüência deste trabalho, estaremos particularmente interessados em obter um método numérico eficiente para o cálculo da solução de um problema não-linear de advecção-difusão que serve de modelo para o espalhamento de manchas de petróleo em superfícies aquosas sob a influência de ventos, correntes marítimas e marés. Tal modelo foi proposto por Benqué e pode ser encontrado em [10]:

$$\frac{\partial u(x,t)}{\partial t} - c(x,t)\Delta\left(|u(x,t)|^2 u(x,t)\right) + div(\vec{\beta}(x,t)u(x,t)) = f(x,t) \ \ , \ \ \Omega \times I$$

$$u(x,t) = g(x) \ , \ x \in \ \partial\Omega_- \ , \ \ t \in I$$

$$\frac{\partial u(x,t)}{\partial \vec{\eta}} = 0 \ , \ \ x \in \ \partial\Omega_+ \ , \ \ t \in I$$

$$u(x,0) = u_0(x) \ , \ x \in \Omega.$$

Aqui, Ω é um domínio limitado do $I\!R^2$ com fronteira dada por $\partial\Omega = \partial\Omega_+ \cup \partial\Omega_-$, onde $\partial\Omega_+$ e $\partial\Omega_-$ já foram definidas no início deste capítulo. Como de costume, $\vec{\eta}(x)$ é o vetor unitário normal no ponto $x \in \partial\Omega$. No modelo, a variável u representa a altura da mancha de petróleo, medida em relação ao nível da superfície aquosa.

Vamos assumir que o campo de velocidades corresponda a um fluxo incompressível, isto é,

$$div(\vec{\beta}(x,t)) = 0.$$

Lembrando que

$$div(\vec{\beta}(x,t)u(x,t)) = \vec{\beta}(x,t) \cdot \nabla u(x,t) + u(x,t) div(\vec{\beta}(x,t)),$$

concluímos que, neste caso,

$$div(\vec{\beta}(x,t)u(x,t)) = \vec{\beta}(x,t) \cdot \nabla u(x,t). \tag{1.9}$$

Iniciaremos descrevendo o problema num caso mais geral da não-linearidade, mas admitindo que a advecção seja nula, que $c(x,t) \equiv 1$ e que não haja fontes nem sorvedouros:

$$\frac{\partial v(x,t)}{\partial t} - \Delta\left(|v(x,t)|^{m-1} v(x,t)\right) = 0 \ , \ \ \Omega \times I \tag{1.10}$$

$$v(x,t) = 0 \ , \ x \in \ \partial\Omega \ , \ \ t \in I \tag{1.11}$$

$$v(x,0) = v_0(x) \ , \ x \in \Omega, \tag{1.12}$$

onde $m \in I\!N, m > 1$, é uma constante, e Ω é um domínio limitado do $I\!R^2$.

Uma vez que o termo $|v(x,t)|^{m-1} v(x,t)$ se anula para $v = 0$, a equação (1.10) não é uniformemente parabólica, não sendo abrangida pela teoria clássica. Entretanto, equações deste tipo (parabólicas degeneradas) têm sido estudadas por vários autores, veja por exemplo [5].

A seguinte família de soluções não-negativas para o problema $(1.10) - (1.12)$, descoberta independentemente por Barenblatt [3] e Pattle [45], nos será bastante útil nos testes futuros de vários métodos numéricos, porque a partir desta solução do problema de difusão, poderemos construir a solução exata do mesmo problema com advecção constante:

$$v(x,y,t,a,\tau) = (t+\tau)^{-\frac{1}{m}} \left[\{a^2 - \frac{(m-1)}{4m^2}(x^2+y^2)(t+\tau)^{-\frac{1}{m}}\}^+ \right]^{\frac{1}{m-1}}, \tag{1.13}$$

onde $a > 0$, $\tau > 0$ e $f^+(x) = \max\{f(x), 0\}$.

Para isso, vamos admitir que $v(x,t)$ resolva um caso particular de (1.10) com $m = 3$, isto é, v seja dada por (1.13), e usá-la na construção de uma solução para o problema não-linear de advecção-difusão reescrito usando (1.9) e com f nula:

$$\frac{\partial u(x,t)}{\partial t} - \Delta\left(|u(x,t)|^2 u(x,t)\right) + \vec{\beta}(x,t) \cdot \nabla u(x,t) = 0 \ , \ \ \Omega \times I. \tag{1.14}$$

Seja $\xi(x,t)$ satisfazendo

$$\frac{d\xi_i}{dt} = -\beta_i(x,t) \ , \ \ i = 1,2 \tag{1.15}$$

e a condição inicial

$$\xi(x,0) = x. \tag{1.16}$$

Definimos $u(x,t) = v(\xi(x,t),t)$. Então, usando a regra da cadeia e (1.15) temos que

$$\frac{\partial u}{\partial t} = \frac{\partial v}{\partial t} + \sum_{i=1}^{2} \frac{\partial v}{\partial x_i}\frac{d\xi_i}{dt} = \frac{\partial v}{\partial t} - \sum_{i=1}^{2} \frac{\partial v}{\partial x_i}\beta_i = \frac{\partial v}{\partial t} - \vec{\beta} \cdot \nabla v. \tag{1.17}$$

Se tomarmos $\vec{\beta}$ constante, a solução de (1.15) − (1.16) será dada por

$$\xi(x,t) = x - \vec{\beta}t.$$

Neste caso,

$$\nabla u(x,t) = \nabla v(\xi(x,t),t)$$

e

$$\Delta\left(|u(x,t)|^2 u(x,t)\right) = \Delta\left(|v(\xi(x,t),t)|^2 v(\xi(x,t),t)\right).$$

Portanto, usando (1.17) e o fato de $v(x,t)$ satisfazer (1.10), temos que

$$\frac{\partial u}{\partial t} + \vec{\beta}\cdot\nabla u - \Delta\left(|u|^2 u\right) = \frac{\partial v}{\partial t} - \vec{\beta}\cdot\nabla v + \vec{\beta}\cdot\nabla v - \Delta\left(|v|^2 v\right) = 0,$$

ou seja,

$$u(x,t) = v(x - \vec{\beta}t, t) \tag{1.18}$$

satisfará a equação não-linear de advecção-difusão (1.14), onde

$$v(x,t,a,\tau) = (t+\tau)^{-\frac{1}{3}}\left[\{a^2 - \frac{1}{18}(x^2+y^2)(t+\tau)^{-\frac{1}{3}}\}^+\right]^{\frac{1}{2}}. \tag{1.19}$$

Pela forma da solução dada por $(1.18)-(1.19)$, podemos observar que esta é não-negativa. Além disso, verificamos que se a condição inicial tiver suporte compacto, as soluções ao longo do tempo também o terão e este deve aumentar com o passar do tempo em virtude do termo $(t+\tau)^{-\frac{1}{3}}$ diminuir quando t aumenta. Este mesmo termo também é responsável pelo decaimento da solução.

Na Figura 1.1, vemos um corte paralelo ao eixo-x do gráfico de $(1.18)-(1.19)$ para $t = 0$. Nesta, observamos que o suporte da solução aumenta quando o parâmetro a aumenta. Além disso, a presença de uma região de variação brusca (ou derivadas grandes) da solução pode ser notada, fato que dificulta a resolução numérica de tais problemas, como veremos no próximo capítulo.

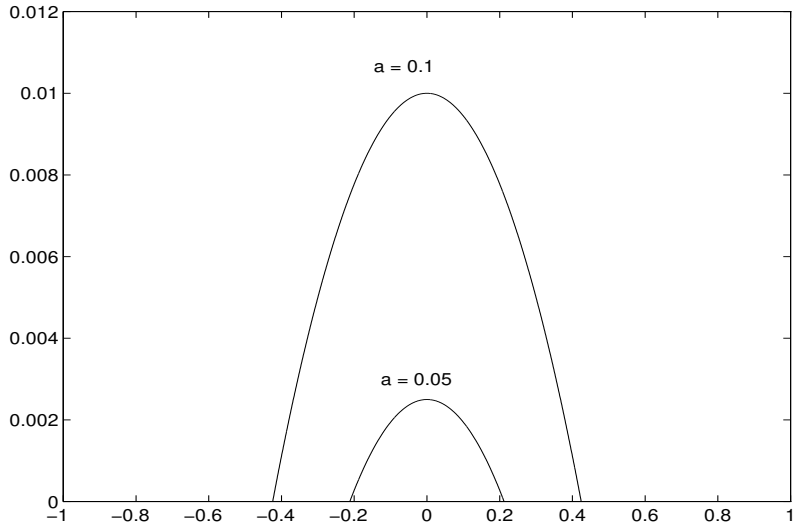

Figura 1.1: Soluções exatas - condição inicial.
Eixo horizontal: x, eixo vertical: u.

1.3 Conservação da Massa

Vamos retornar ao problema não-linear de advecção-difusão da seção anterior e que, como já dissemos, serve de modelo para o espalhamento de manchas de petróleo em superfícies aquosas. Pretendemos estabelecer uma propriedade a respeito da massa de sua solução para posteriormente usá-la para testar o comportamento de alguns métodos numéricos para este problema.

$$\frac{\partial u(x,t)}{\partial t} - \Delta((u(x,t))^3) + div(\vec{\beta}(x,t)u(x,t)) = f(x,t) \ , \ \ \Omega \times I \tag{1.20}$$

$$u(x,t) = g(x) \ , \ x \in \ \partial\Omega_- \ , \ \ t \in I \tag{1.21}$$

$$\frac{\partial u(x,t)}{\partial \vec{\eta}} = 0 \ , \ \ x \in \ \partial\Omega_+ \ , \ \ t \in I \tag{1.22}$$

$$u(x,0) = u_0(x) \ , \ x \in \Omega. \tag{1.23}$$

Vamos tomar a equação (1.20), reescrevendo $\Delta(u^3)$ como $div(3u^2\nabla u)$ e integrá-la num subdomínio $\tilde{\Omega} \subset \Omega$, para o caso especial $g = 0$ da condição de Dirichlet (1.21):

$$\int_{\tilde{\Omega}} \frac{\partial u}{\partial t} dx - \int_{\tilde{\Omega}} div(3u^2\nabla u)dx + \int_{\tilde{\Omega}} div(\vec{\beta}u)dx = \int_{\tilde{\Omega}} f dx. \tag{1.24}$$

Usando o teorema de divergência chegamos a

$$\frac{\partial}{\partial t}\int_{\tilde{\Omega}} u dx - \int_{\partial\tilde{\Omega}} 3u^2\nabla u \cdot \vec{\eta} ds + \int_{\partial\tilde{\Omega}} u\vec{\beta} \cdot \vec{\eta} ds = \int_{\tilde{\Omega}} f dx. \tag{1.25}$$

Tomando u com suporte compacto contido em $\tilde{\Omega}$, temos que

$$\frac{\partial}{\partial t}\int_{\tilde{\Omega}} u dx = \int_{\tilde{\Omega}} f dx,$$

isto é, o problema tem a importante propriedade de que a integral de u sobre o domínio é totalmente determinada pela correspondente integral de f.

Além disso, se $f = 0$ em (1.20):

$$\frac{\partial}{\partial t}\int_{\tilde{\Omega}} u(x,t)dx = 0, \tag{1.26}$$

que implica em uma característica de controle de massa que é importante na prática, e que deve ser garantida pelas aproximações numéricas.

Quando parte da fronteira $\partial\tilde{\Omega}$ coincidir com parte de $\partial\Omega$, as condições de contorno $(1.21) - (1.22)$ conduzirão (1.25), ainda com f identicamente nula, a

$$\frac{\partial}{\partial t}\int_{\tilde{\Omega}} u(x,t)dx = -\int_{\partial\tilde{\Omega}_+} u(x,t)\vec{\beta} \cdot \vec{\eta} ds, \tag{1.27}$$

onde $\partial\tilde{\Omega}_+ = \partial\Omega_+ \cap \partial\tilde{\Omega}$, levando a um decaimento da massa em virtude do sinal positivo de $\vec{\beta} \cdot \vec{\eta}$ na integral do lado direito de (1.27).

Desta forma, $(1.26) - (1.27)$ nos levam a concluir que a massa é conservada ao longo do tempo desde que a solução u não esteja em contato com $\partial\Omega$.

Observamos que esta propriedade continua valendo para problemas mais gerais, com termo de difusão não-linear da forma $-div(c(u)\nabla u(x,t))$, desde que o coeficiente de difusão $c(u)$ se anule quando $u = 0$.

Capítulo 2

Avaliação de Alguns Métodos de Elementos Finitos

2.1 Introdução

Neste capítulo, retornaremos ao modelo de espalhamento de manchas de petróleo sobre superfícies aquosas, apresentado no capítulo anterior, para experimentar vários métodos numéricos para aproximar sua solução, construídos por diferentes combinações entre as discretizações no espaço e no tempo, além da forma de linearização do problema. Nosso objetivo é explicitar as principais qualidades e defeitos de tais métodos, quando aplicados a este tipo de problema.

Iniciaremos recolocando o problema não-linear de advecção-difusão:

$$\frac{\partial u(x,t)}{\partial t} - c\Delta((u(x,t))^3) + \vec{\beta}(x,t)\cdot\nabla u(x,t) = f(x,t) \ \ , \ \ \Omega \times I \tag{2.1}$$

$$u(x,t) = g(x) \ , \ x \in \ \partial\Omega_- \ , \ \ t \in I \tag{2.2}$$

$$\frac{\partial u(x,t)}{\partial \vec{\eta}} = 0 \ , \ \ x \in \ \partial\Omega_+ \ , \ \ t \in I \tag{2.3}$$

$$u(x,0) = u_0(x) \ , \ x \in \Omega, \tag{2.4}$$

com $I = (0,T] \subset I\!R$ e $\Omega \subset I\!R^2$ com fronteira $\partial\Omega = \partial\Omega_+ \cup \partial\Omega_-$, onde

$$\partial\Omega_+ = \{ \ x \in \partial\Omega \ / \ \vec{\eta}(x)\cdot\vec{\beta}(x,t) \geq 0 \ \},$$

é também conhecido como bordo de saída do fluxo e

$$\partial\Omega_- = \{\ x \in \partial\Omega\ /\ \vec{\eta}(x) \cdot \vec{\beta}(x,t) < 0\ \},$$

que recebe o nome de bordo de entrada do fluxo. Podemos reparar que um coeficiente c, constante, foi introduzido com a finalidade de controlar o caráter da equação. Assim, por exemplo, o problema será predominantemente hiperbólico, desde que c assuma valores pequenos comparativamente com o tamanho do coeficiente de advecção $\vec{\beta}$.

2.2 O Método de Euler-Galerkin

Existem várias possibilidades de combinação entre métodos de linearização e discretização, no espaço e no tempo, para o problema $(2.1) - (2.4)$. Nosso principal objetivo é avaliar métodos de elementos finitos porque estes se adaptam de forma mais simples a problemas com geometrias complicadas - caso das regiões em que ocorrem os derramamentos de petróleo que desejamos prever - do que métodos de diferenças finitas. Entretanto, antes de atacar o problema com um método deste tipo, optamos por iniciar discretizando o tempo através de métodos de diferenças finitas e o espaço pelo método de Galerkin. Desta forma, a linearização pode ser feita de forma simples, aproveitando a discretização da variável temporal, como veremos a seguir.

2.2.1 Linearização e Semi-discretização no Tempo

Vamos definir $\Pi: \quad 0 = t_0 < t_1 < \cdots < t_N = T$ uma partição qualquer de $I = [0,T]$ com $I_n = (t_{n-1}, t_n)$ e $k_n = t_n - t_{n-1}$ o passo local no tempo.

Da mesma forma como em [39], vamos utilizar o método de Euler Regressivo para discretizar a equação (2.1) e ao mesmo tempo linearizá-la. Para isso, vamos olhar para o termo não-linear $\Delta(u^3)$ e reescrevê-lo como

$$\Delta(u^3) = div(\nabla u^3) = div(3u^2 \nabla u). \tag{2.5}$$

Se $u^n(x)$ é uma aproximação de $u(x,t_n)$, $n = 1, 2, \cdots, N$, podemos computar (2.5) considerando $(u(x,t))^2$ calculado no nível de tempo anterior da discretização, e portanto, conhecido. Desta forma, temos que resolver um problema linear a cada nível de tempo. Procuramos $u^n(x)$, $n = 1, 2, \cdots, N$, $x \in \Omega$, satisfazendo

$$\frac{u^n - u^{n-1}}{k_n} - \ div(3c(u^{n-1})^2 \nabla u^n) + \vec{\beta}(x,t) \cdot \nabla u^n = f(x,t_n) \tag{2.6}$$

e a condição inicial

$$u^0(x) = u_0(x). \tag{2.7}$$

Vale ressaltar que o método de Euler Regressivo é um algoritmo implícito; para utilizar o método de Crank-Nicolson, também implícito, para discretizar o tempo, trocamos a equação (2.6) por

$$\frac{u^n - u^{n-1}}{k_n} - \ div\left(3c(u^{n-1})^2\frac{(\nabla u^n + \nabla u^{n-1})}{2}\right) + \vec{\beta}(x, t_{n+\frac{1}{2}}) \cdot \frac{(\nabla u^n + \nabla u^{n-1})}{2} = f(x, t_{n+\frac{1}{2}}).$$

Segundo [56], este método pode apresentar oscilações nas vizinhanças de descontinuidades dos valores iniciais ou entre os valores iniciais e as condições de fronteira. Isto se explica porque a matriz de amplificação, utilizada na análise da estabilidade de tal método, pode apresentar autovalores negativos próximos de -1, levando a uma alternância de sinais de suas potências e conseqüente diminuição vagarosa de sua magnitude. Por isso, apesar de apresentar ordem de precisão maior ($O(k_n^2)$), optamos pelo método de Euler Regressivo para evitar esta possibilidade.

Para simplificar a exposição das idéias, de agora em diante, usaremos a seguinte notação:

$$\alpha(x) = 3c(u^{n-1}(x))^2. \tag{2.8}$$

A formulação variacional do problema (2.6)−(2.7) com condições de contorno (2.2)−(2.3) será descrita a seguir.

Considerando o espaço das funções-teste $H_0^1 = \{v \in \mathcal{H}^1(\Omega) \ / \ v \mid_{\partial\Omega_-} = 0\}$, multiplicando (2.6) por $v \in H_0^1$, para um dado nível de tempo n, integrando sobre Ω e usando o teorema de Green, tem-se que

$$\langle u^n, v\rangle + k_n a(u^n, v) = b(v), \ \ \forall\ v \in H_0^1, \tag{2.9}$$

sendo

$$a(u^n, v) = \int_\Omega 3c(u^{n-1}(x))^2 \nabla u^n(x) \cdot \nabla v(x) dx + \int_\Omega \vec{\beta}(x,t) \cdot \nabla u^n(x) v(x) dx,$$

$$b(v) = \langle k_n f + u^{n-1}, v\rangle = \int_\Omega k_n f(x, t_n) v(x) dx + \int_\Omega u^{n-1}(x) v(x) dx,$$

$$\langle u^n, v\rangle = \int_\Omega u^n(x) v(x) dx.$$

Impondo as condições de contorno fortemente, isto é, buscando a solução num conjunto cujas funções satisfaçam a condição de contorno dada em (2.2), devemos definir o conjunto de funções admissíveis como

$$H_g^1 = \{\ v \in \mathcal{H}^1(\Omega)\ /\ v\ |_{\partial\Omega_-} = g\ \}.$$

Assim, podemos enunciar o seguinte problema variacional:

$(\mathcal{V}_g)$ encontrar $u^n \in H_g^1,\ n = 1, 2, \cdots, N$, solução da equação

$$\langle u^n, v\rangle + k_n a(u^n, v) = b(v),\ \ \forall\ v \in H_0^1, \tag{2.10}$$

satisfazendo a condição inicial

$$u^0(x) = u_0(x). \tag{2.11}$$

Veremos a seguir uma maneira típica de garantir existência e unicidade da solução de problemas variacionais. Mas antes, introduziremos alguns conceitos.

Vamos considerar $(H, \langle\cdot,\cdot\rangle_H)$ um espaço de Hilbert com $\|\cdot\|_H$ a norma associada ao produto interno $\langle\cdot,\cdot\rangle_H$. Uma forma linear $b(\cdot)$ em H é dita contínua se existe $c_1 > 0$ tal que

$$|\ b(v)\ | \leq c_1\|v\|_H, \quad \forall\ v \in H.$$

De forma análoga, uma forma bilinear $\hat{a}(\cdot,\cdot)$ em H é dita contínua se existe $c_2 > 0$ tal que

$$|\ \hat{a}(v, w)\ | \leq c_2\|v\|_H\|w\|_H, \quad \forall\ v, w \in H$$

e dita coerciva se existe $c_3 > 0$ tal que

$$\hat{a}(v, v) \geq c_3\|v\|_H^2, \quad \forall\ v \in H.$$

Se $\hat{a}(u, v) = \hat{a}(v, u),\ \ \forall\ u, v \in H$, então dizemos que $\hat{a}(\cdot,\cdot)$ é uma forma bilinear simétrica.

No próximo teorema, enunciamos as condições para que um problema variacional tenha solução única.

Teorema de Lax-Milgram: Sejam H um espaço de Hilbert, $\hat{a}(\cdot,\cdot) : H \times H \to \Re$ uma forma bilinear contínua e coerciva e $b(\cdot) : H \to \Re$ uma forma linear contínua. Então, existe um único $u^* \in H$ solução da equação

$$\hat{a}(u, v) = b(v), \quad \forall\ v \in H,$$

além disso, vale a seguinte estimativa de estabilidade

$$\|u\|_H \leq \frac{c_1}{c_3}.$$

Se tomarmos $\hat{a}(\cdot,\cdot)$ simétrica, então a demonstração do teorema acima é uma aplicação do Teorema de Representação de Riesz. Em ambos os casos, simétrico ou não-simétrico, a demonstração pode ser encontrada em [49].

O problema variacional (2.10) não se enquadra nas hipóteses do teorema de Lax-Milgram porque H_g^1 é um subconjunto e não um subespaço de $\mathcal{H}^1(\Omega)$. Para contornar esta dificuldade, adaptam-se as idéias contidas em [1], por exemplo. Para isto, redefinimos o problema (2.10) da seguinte forma: dado um elemento $\hat{u}^n \in H_g^1$, fixo, porém arbitrário, podemos descrever H_g^1 por

$$H_g^1 = \{\ u^n \in \mathcal{H}^1(\Omega)\ /\ u^n = w^n + \hat{u}^n,\ \forall\ w^n \in H_0^1\ \}.$$

Reescrevendo (2.10) com u^n da forma acima temos

$$\hat{a}(w^n + \hat{u}^n, v) = b(v),$$

onde a forma bilinear $\hat{a}(\cdot,\cdot)$ é dada por

$$\hat{a}(u^n, v) = \langle u^n, v\rangle + k_n a(u^n, v).$$

Portanto, o problema variacional pode ser reescrito como

$(\mathcal{V})$ encontrar $w^n \in H_0^1,\ n = 1, 2, \cdots, N$, solução da equação

$$\hat{a}(w^n, v) = \hat{b}(v),\ \ \forall\ v \in H_0^1,$$

onde $\hat{b}(v) = b(v) - \hat{a}(\hat{u}^n, v)$.

2.2.2 Discretização no Espaço e no Tempo

Feita a formulação variacional do problema, passaremos ao método de Galerkin propriamente, ou seja, ao problema variacional discreto formulado em espaços de elementos finitos.

Vamos considerar que Ω seja um domínio poligonal e tomar os subespaços de dimensão finita consistindo de funções polinomiais por partes, $V_h \subset H_0^1$ dado por

$$V_h = \{\ v \in H_0^1\ /\ v\ |_K \in \mathcal{P}_r(K), \forall\ K \in T_h\ \},$$

e $V_h^g \subset H_g^1$ definido como

$$V_h^g = \{\ v \in H_g^1\ /\ v\ |_K \in \mathcal{P}_r(K), \forall\ K \in T_h\ \},$$

onde $T_h = \{K\}$ é uma triangularização de Ω e $\mathcal{P}_r(K)$ é o espaço dos polinômios de grau menor ou igual a r, definidos no elemento K. Denotaremos também $h_K \equiv$ diâmetro de K, que é o maior lado do triângulo K e

$$h = \max\{\ h_K\ /\ K \in T_h\ \}.$$

Então enunciamos o seguinte problema variacional discreto no espaço e no tempo, correspondente ao problema (2.10) − (2.11):

$(\mathcal{V}_h)$ encontrar $u_h^n \in V_h^g,\ n = 1, 2, \cdots N$, tal que

$$\langle u_h^n, v\rangle + k_n a(u_h^n, v) = b(v),\ \ \forall\ v \in V_h, \tag{2.12}$$

$$\langle u_h^0, v\rangle = \langle u_0, v\rangle,\ \ \forall\ v \in V_h. \tag{2.13}$$

Vamos explicitar a discretização dada pelo problema (2.12)−(2.13). Para isso, suporemos que na condição de contorno (2.2) tenhamos $g = 0$.

Seja $\{\Psi_1, \Psi_2, \cdots, \Psi_M\}$ uma base para V_h. Então, uma função $v \in V_h$ tem representação única

$$v(x) = \sum_{j=1}^{M} c_j \Psi_j(x),$$

enquanto que, para um dado nível de tempo n,

$$u_h^n(x) = \sum_{j=1}^{M} c_j^n \Psi_j(x).$$

Levando estas representações a (2.12) temos que

$$\sum_{j=1}^{M} c_j^n \langle \Psi_j, v\rangle + k_n \sum_{j=1}^{M} c_j^n a(\Psi_j, v) = b(v). \tag{2.14}$$

Como (2.14) vale para qualquer elemento de V_h, em particular vale para os elementos de sua base. Desta forma,

$$\sum_{j=1}^{M} c_j^n \left[\langle \Psi_j, \Psi_i\rangle + k_n a(\Psi_j, \Psi_i)\right] = b(\Psi_i),\ \ i = 1, 2, \cdots M. \tag{2.15}$$

Por exigência da estratégia de linearização, interpolamos a solução conhecida no nível de tempo anterior $n-1$ usando as próprias funções de base de V_h^g:

$$u_h^{n-1}(x) = \sum_{l=1}^{M} c_l^{n-1} \Psi_l(x).$$

Substituindo esta expressão em (2.15), e levando em consideração as definições da forma bilinear $a(\cdot,\cdot)$ e da forma linear $b(\cdot)$, temos

$$\sum_{j=1}^{M} c_j^n \left[\langle \Psi_j, \Psi_i \rangle + k_n \langle 3c(\sum_l c_l^{n-1}\Psi_l)^2 \nabla\Psi_j, \nabla\Psi_i \rangle + k_n \langle \vec{\beta}\cdot\nabla\Psi_j, \Psi_i \rangle \right] =$$

$$= \sum_{j=1}^{M} c_j^{n-1} \left[\langle \Psi_j, \Psi_i \rangle \right] + k_n \langle f, \Psi_i \rangle, \quad i = 1, 2, \cdots M,$$

que é um sistema linear com M equações e M incógnitas $c_1^n, c_2^n, \cdots, c_M^n$, que pode ser posto na forma matricial

$$A(c^{n-1})c^n = Bc^{n-1} + d, \tag{2.16}$$

lembrando que a matriz A, devido à forma da linearização, depende dos coeficientes já calculados $c_1^{n-1}, c_2^{n-1}, \cdots, c_M^{n-1}$ e tem seus elementos definidos por

$$A(c^{n-1})_{ij} = \langle \Psi_j, \Psi_i \rangle + k_n \langle 3c(\sum_l c_l^{n-1}\Psi_l)^2 \nabla\Psi_j, \nabla\Psi_i \rangle + k_n \langle \vec{\beta}\cdot\nabla\Psi_j, \Psi_i \rangle, \quad i,j = 1, 2, \cdots M.$$

enquanto

$$B_{ij} = \langle \Psi_j, \Psi_i \rangle, \quad i,j = 1, 2, \cdots M$$

e

$$d_i = k_n \langle f, \Psi_i \rangle, \quad i,j = 1, 2, \cdots M.$$

Sendo a condição inicial dada, isto é, sendo os coeficientes $c_1^0, c_2^0, \cdots, c_M^0$ calculados pela relação (2.13), podemos resolver o sistema linear (2.16) para cada nível de tempo.

Observamos que (2.16) de fato tem uma única solução, pelo menos quando $\operatorname{div}\vec{\beta} = 0$, que é o caso no qual estaremos interessados.

Para provar este resultado, basta mostrar que a única solução da equação $A(c^{n-1})z = 0$ é $z = (z_1, \ldots z_M) = 0$. Mas, denotando

$$u(x) = \sum_{j=1}^{M} z_j \Psi_j(x),$$

isto é equivalente a mostrar que $u = 0$ é a única solução do seguinte problema variacional:

$$\langle u, v\rangle + k_n a(u, v) = 0,\ \forall v \in V_h.$$

Para isto, lembramos que estamos considerando $g = 0$ e, portanto, podemos tomar $v = u \in V_h$ na identidade anterior para obter:

$$||u||^2 + k_n a(u, u) = 0,$$

sendo

$$\begin{aligned} a(u,u) &= \int_\Omega 3c(u^{n-1}(x))|\nabla u(x)|^2 dx + \int_\Omega \vec{\beta}(x,t_n)\cdot\nabla u(x)u(x)dx \\ &= \int_\Omega 3c(u^{n-1}(x))|\nabla u(x)|^2 dx + \tfrac{1}{2}\int_\Omega \vec{\beta}(x,t_n)\cdot\nabla u^2(x)dx \\ &= \int_\Omega 3c(u^{n-1}(x))|\nabla u(x)|^2 dx - \tfrac{1}{2}\int_\Omega \operatorname{div}\vec{\beta}(x,t_n)u^2(x)dx \\ &\quad + \int_{\partial\Omega^+} \vec{\beta}(x,t_n).n(x)u^2(x)dS(x) \\ &= \int_\Omega 3c(u^{n-1}(x))|\nabla u(x)|^2 dx + \int_{\partial\Omega^+} \vec{\beta}(x,t_n).n(x)u^2(x)dS(x) \geq 0, \end{aligned}$$

onde utilizamos o fato de que $\operatorname{div}\vec{\beta} = 0$, $g = 0$, $\vec{\beta}(x,t_n).n(x) \geq 0$ sobre $\partial\Omega^+$ e também que $c(u^{n-1}) \geq 0$. "$\|\cdot\|$" denota a norma-$\mathcal{L}_2$.

Deste resultado e da última igualdade concluímos que $u = 0$ e, portanto, (2.16) tem de fato uma única solução.

2.2.3 Condições de Estabilidade

Para estabelecer condições de estabilidade para o método de Galerkin aplicado ao problema $(2.6) - (2.7)$, novamente seguiremos os passos contidos em [39]. Iniciaremos considerando a seguinte desigualdade

$$2ab \leq a^2\epsilon + \frac{b^2}{\epsilon}, \quad \forall\ a, b \in I\!R\ \ e\ \ \epsilon \in I\!R_+^*. \tag{2.17}$$

Se tomarmos (2.13) com $v = u_h^0$ e levando em consideração (2.17) com $\epsilon = 1$, temos

$$\langle u_h^0, u_h^0\rangle = \langle u_0, u_h^0\rangle \leq \frac{1}{2}\|u_0\|^2 + \frac{1}{2}\|u_h^0\|^2,$$

onde "$\|\cdot\|$" denota a norma-$\mathcal{L}_2$.

Logo, concluímos que

$$\|u_h^0\| \leq \|u_0\|. \tag{2.18}$$

Para obter uma relação de estabilidade para o problema discreto (2.12) − (2.13), com a simplificação $f = 0$, tomamos $v = u_h^n$ em (2.12) e obtemos

$$\|u_h^n\|^2 - \langle u_h^{n-1}, u_h^n \rangle + a(u_h^n, u_h^n)k_n = 0.$$

Usando (2.17) com $\epsilon = 1$, concluímos que para $n = 1, 2, \cdots, N$

$$\frac{1}{2}\|u_h^n\|^2 - \frac{1}{2}\|u_h^{n-1}\|^2 + a(u_h^n, u_h^n)k_n \leq 0.$$

Somando entre 1 e $\overline{n}$, fixo, temos

$$\|u_h^{\overline{n}}\|^2 + 2\sum_{m=1}^{\overline{n}} a(u_h^m, u_h^m)k_m \leq \|u_h^0\|^2 \leq \|u_0\|^2.$$

Fazendo uso da coercividade de $a(\cdot, \cdot)$, chegamos a

$$\|u_h^{\overline{n}}\| \leq \|u_h^0\| \leq \|u_0\|, \;\; \overline{n} = 1, \cdots, N, \tag{2.19}$$

que é uma condição de estabilidade que relaciona a solução aproximada, em cada nível de tempo, com a projeção da condição inicial no espaço V_h e a condição inicial propriamente.

Podemos ainda obter uma outra estimativa de estabilidade envolvendo as primeiras derivadas da aproximação de Galerkin u_h. Esta será muito útil na análise dos resultados numéricos apresentados por este método. Para isso, necessitamos do seguinte resultado que se obtém diretamente da fórmula de Green, desde que $div(\vec{\beta}) = 0$:

$$\int_\Omega u\vec{\beta} \cdot \nabla u dx = \frac{1}{2}\int_{\partial\Omega} u^2 \vec{\beta} \cdot \vec{\eta} ds. \tag{2.20}$$

Tomando $v = u_h^n$ em (2.12) com $f = 0$ e usando (2.23), obtemos

$$k_n\langle 3c(u^{n-1})^2\nabla u_h^n, \nabla u_h^n\rangle + \frac{k_n}{2}\int_{\partial\Omega_+}(u_h^n)^2\vec{\beta}\cdot\vec{\eta}ds + \langle u_h^n, u_h^n\rangle = \langle u_h^{n-1}, u_h^n\rangle.$$

Levando em conta (2.17) com $\epsilon = 1$, temos que

$$k_n\|(3c(u^{n-1})^2)^{\frac{1}{2}}\nabla u_h^n\|^2 + \frac{k_n}{2}\int_{\partial\Omega_+}(u_h^n)^2\vec{\beta}\cdot\vec{\eta}ds + \frac{1}{2}\|u_h^n\|^2 \leq \frac{1}{2}\|u_h^{n-1}\|^2.$$

Lembrando que a integral sobre $\partial\Omega_+$ desta última desigualdade é não-negativa, porque $\vec{\beta}\cdot\vec{\eta} \geq 0$ em $\partial\Omega_+$, e usando (2.19)

$$k_n\|(3c(u^{n-1})^2)^{\frac{1}{2}}\nabla u_h^n\|^2 + \frac{1}{2}\|u_h^n\|^2 \leq \frac{1}{2}\|u_h^{n-1}\|^2 \leq \frac{1}{2}\|u_h^0\|^2 \leq \frac{1}{2}\|u_0\|^2. \tag{2.21}$$

Na estimativa (2.21) verificamos que, se a condição inicial u_0 for limitada, podemos obter um controle no comportamento das soluções aproximadas u_h^n, bem como em suas derivadas de primeira ordem. Entretanto, quando o termo $3c(u^{n-1})^2$, que funciona como coeficiente de difusão, se torna pequeno, o controle sobre a norma-$\mathcal{L}_2$ de ∇u_h^n se perde. Infelizmente, como as soluções aproximadas u_h^n têm suporte compacto, existem regiões em que o coeficiente de difusão diminui até se anular. Nestes locais, o problema torna-se puramente hiperbólico e a solução apresenta uma transição muito rápida, causando o aparecimento de oscilações espúrias na aproximação de Galerkin.

Nas simulações apresentadas na Seção 2.6, este comportamento indesejado poderá ser verificado. Para contornar este problema, nas próximas seções, introduziremos métodos de elementos finitos desenvolvidos para aliarem boa estabilidade e precisão.

2.3 Métodos Estabilizados de Elementos Finitos

A razão principal para a introdução dos métodos estabilizados de elementos finitos é que a aplicação do método de Galerkin a diversos problemas de matemática e engenharia conduz a aproximações numéricas deficientes. Tais métodos constituem uma metodologia sistemática para melhorar a estabilidade sem comprometer a precisão das soluções numéricas.

Operadores de advecção-difusão deixaram analistas numéricos perplexos por décadas. Historicamente, tais operadores eram tratados com métodos que apresentavam comprometimentos na estabilidade (diferenças finitas centrais) ou na precisão (*upwind*).

Nesta seção, continuaremos a utilizar o método de Euler Regressivo para a discretização no tempo, mas em conjunto com métodos estabilizados de elementos finitos, de forma a evitar as oscilações descritas no final da seção anterior.

2.3.1 Método Euler-*Streamline Diffusion*

Para introduzir o método *Streamline Diffusion*, iniciaremos olhando para uma equação linear de advecção-difusão como em [39]:

$$\frac{\partial u(x,t)}{\partial t} - c(x,t)\Delta(u(x,t)) + \vec{\beta}(x,t)\cdot\nabla u(x,t) = f(x,t).$$

Para este caso, a forma mais simples de contornar as dificuldades apresentadas pelo método de Galerkin no caso em que o coeficiente de difusão for pequeno, isto é, quando $c(x,t) << h$, é evitar esta situação. Isto pode ser feito decrescendo-se o parâmetro da triangularização h até que $c(x,t) > h$, mas esta estratégia pode se tornar impraticável se $c(x,t)$ assumir valores muito pequenos ou particularmente, se estivermos tratando um problema puramente hiperbólico, isto é, sem transporte difusivo ($c(x,t) \equiv 0$). É justamente neste caso que se encaixa a equação (2.1), após sua discretização e linearização via método de Euler Regressivo, uma vez que o correspondente coeficiente de difusão, neste caso, é dado por $\alpha(x) = 3c(u^{n-1})^2$ que assume valores pequenos e até nulos.

Por outro lado, podemos simplesmente acrescentar à equação linear acima um termo de difusão artificial $-\delta\Delta u$, para o qual $\delta = h - c(x,t)$. Esta é a idéia principal dos métodos de difusão artificial e que pode ser colocada em funcionamento para a equação não-linear (2.1). Porém este tipo de método, apesar de produzir soluções não oscilatórias, tem o defeito de introduzir uma perturbação na equação original que não permite que a solução aproximada tenha precisão melhor que $O(h)$.

A alternativa pode ser, no caso linear ou não, a introdução de difusão artificial somente na direção do campo de velocidades $\vec{\beta}$, através do termo $-\delta u_{\vec{\beta}\vec{\beta}}$, com $\delta = h - \alpha(x)$, (equação não-linear) ou $\delta = h - c(x,t)$ (equação linear), mas sem a violação da consistência entre a

equação diferencial e o esquema de discretização, onde estamos denotando

$$u_{\vec{\beta}} = \vec{\beta} \cdot \nabla u$$

e

$$u_{\vec{\beta}\vec{\beta}} = \vec{\beta} \cdot \nabla(\vec{\beta} \cdot \nabla u).$$

Isto corresponde a atribuir, em cada elemento, um peso maior aos nós de onde vem o fluxo definido por $\vec{\beta}$, como acontece com os métodos *upwind* de diferenças finitas.

Para obter tal efeito, vamos alterar o espaço das funções-teste tornando-o diferente do espaço de aproximação. Métodos com esta propriedade são classificados como métodos de Petrov-Galerkin. O método *Streamline Diffusion* é um destes métodos. Para estabelecê-lo, tomaremos funções-teste do tipo $w = v + \delta\vec{\beta} \cdot \nabla v$, $v \in V_h$, no problema $(2.12) - (2.13)$ que, após o uso do Teorema de Green, fica reescrito como:

$$\langle \frac{u_h^n - u_h^{n-1}}{k_n}, v + \delta\vec{\beta} \cdot \nabla v \rangle + \langle 3c(u^{n-1})^2 \nabla u_h^n, \nabla v \rangle - \langle div(3c(u^{n-1})^2 \nabla u_h^n), \delta\vec{\beta} \cdot \nabla v \rangle \; + \quad (2.22)$$

$$+ \langle \vec{\beta} \cdot \nabla u_h^n, v + \delta\vec{\beta} \cdot \nabla v \rangle = \langle f, v + \delta\vec{\beta} \cdot \nabla v \rangle, \; \forall \; v \in V_h, \;\; n = 1, 2, \cdots, N.$$

$$\langle u_h^0, v + \delta\vec{\beta} \cdot \nabla v \rangle = \langle u_0, v + \delta\vec{\beta} \cdot \nabla v \rangle, \; \forall \; v \in V_h, \quad (2.23)$$

sendo $\delta = \overline{c}h$ se $3c(u^{n-1})^2 < h, para \; \overline{c} > 0$ suficientemente pequena e $\delta = 0$ se $3c(u^{n-1})^2 \geq h$.

A escolha de δ é feita localmente, podendo variar de elemento para elemento, dependendo da relação entre a difusão e advecção nos mesmos.

Na prática, escolhemos tal parâmetro segundo a proposta de Johnson [41]:

$$\delta_k = \overline{c} \; \frac{\max\left(h_k - \frac{3c(u^{n-1})^2}{|\vec{\beta}_k|}, 0\right)}{\mid \vec{\beta}_k \mid}, \quad (2.24)$$

onde $\overline{c}$ é uma constante positiva no intervalo $[0.5, 1]$ e $\mid \vec{\beta}_k \mid$ é o maior valor do módulo do vetor $\vec{\beta}$, calculado entre os vértices do elemento k.

O termo $\delta\langle\vec{\beta} \cdot \nabla u_h^n, \vec{\beta} \cdot \nabla v\rangle$ é um transporte difusivo na direção de $\vec{\beta}$ e funciona como uma fonte de estabilização do método numérico porque aumenta a quantidade de difusão na direção do vetor $\vec{\beta}$.

O método (2.22) – (2.23) é equivalente, para cada nível de tempo, ao sistema linear nas variáveis $c_1^n, c_2^n, \cdots, c_M^n$ dado por

$$\sum_{j=1}^{M} c_j^n \left[\langle \Psi_j, \Psi_i \rangle + k_n \langle 3c(\sum_l c_l^{n-1}\Psi_l)^2 \nabla\Psi_j, \nabla\Psi_i \rangle + k_n \langle \vec{\beta}\cdot\nabla\Psi_j, \Psi_i \rangle \right] +$$

$$+ \sum_{j=1}^{M} c_j^n \left[\delta\langle \Psi_j, \vec{\beta}\cdot\nabla\Psi_i \rangle - \ k_n\delta\langle div(3c(\sum_l c_l^{n-1}\Psi_l)^2\nabla\Psi_j, \vec{\beta}\cdot\nabla\Psi_i \rangle + k_n\delta\langle \vec{\beta}\cdot\nabla\Psi_j, \vec{\beta}\cdot\nabla\Psi_i \rangle \right] =$$

$$= \sum_{j=1}^{M} c_j^{n-1} \left[\langle \Psi_j, \Psi_i + \delta\vec{\beta}\cdot\nabla\Psi_i \rangle \right] + k_n \langle f, \Psi_i + \delta\vec{\beta}\cdot\nabla\Psi_i \rangle, \quad i = 1, 2, \cdots M,$$

onde $\{\Psi_1, \Psi_2, \cdots, \Psi_M\}$ é uma base para o espaço de aproximação V_h. Podemos escrever também, de forma matricial,

$$A(c^{n-1})c^n = Bc^{n-1} + d.$$

Novamente lembramos que a matriz A, devido à forma da linearização, depende dos coeficientes já calculados $c_1^{n-1}, c_2^{n-1}, \cdots, c_M^{n-1}$. Além disso, podemos observar que A, B e d têm termos adicionais, se comparados ao sistema linear (2.16) do método Euler-Galerkin.

2.3.2 Método de Euler-Galerkin com Funções Bolha

Adaptando as idéias de Franca e Farhat [21], para problemas estacionários e lineares, aplicaremos o método de Galerkin com funções bolha ao modelo de espalhamento de manchas de petróleo em superfícies aquosas.

Tal método se baseia na mesma formulação variacional do problema contínuo (2.1) – (2.4), mas com espaço de aproximação maior. A idéia principal é escolher o espaço V_h adequadamente, de modo a aumentar a estabilidade do método de Galerkin. Isto pode ser conseguido pela adição de funções bolha ao espaço de elementos finitos e é equivalente a incluir um termo de estabilização do mesmo tipo do método *Streamline Diffusion*.

Uma função bolha, que denotaremos por φ_K, tem suporte contido em um único elemento e satisfaz $\varphi_K > 0$, $\forall\ x \in K$, $\varphi_K(x) = 0$, $\forall\ x \in \partial K$ e $\varphi_K = 1$ no baricentro do triângulo K, $\forall\ K \in T_h$. Como um exemplo, se o espaço de aproximação consistir de funções polinomiais

por partes, de ordem 1 ou 2, podemos tomar bolhas com expressão cúbica e apoiadas em um único elemento.

Em nossas simulações, utilizamos a seguinte base local linear no elemento padrão, isto é, o triângulo com vértices em $(0,0),(1,0)$ e $(0,1)$:

$$\lambda_1 = 1 - \xi - \eta, \qquad \lambda_2 = \xi, \qquad \lambda_3 = \eta$$

e a função bolha dada por

$$\varphi_K = 27\lambda_1\lambda_2\lambda_3.$$

Chamaremos de $\mathcal{B}(K)$ o espaço das funções bolha definidas sobre cada elemento da triangularização T_h.

Vamos considerar o subespaço

$$V_h^b = \{\ v \in H_0^1\ /\ v\mid_K \in \mathcal{P}_r(K) \oplus \mathcal{B}(K),\ \forall K \in T_h\ \}.$$

Discretizando o tempo pelo método de Euler Regressivo, estaremos procurando $u_h \in V_h^b$, aproximação de $u(x,t_n)$, $n = 1,2,\cdots,N,\ x \in \Omega$, satisfazendo

$$\langle \frac{u_h^n - u_h^{n-1}}{k_n}, v\rangle + \langle 3c(u^{n-1})^2\nabla u_h^n, \nabla v\rangle + \langle \vec{\beta}\cdot\nabla u_h^n, v\rangle = \langle f, v\rangle,\ \forall v \in V_h^b, \tag{2.25}$$

$$\langle u_h^0, v\rangle = \langle u_0, v\rangle,\ \forall v \in V_h^b. \tag{2.26}$$

Se tomarmos uma base $\{\Psi_1,\Psi_2,\cdots,\Psi_M\}$ para o espaço V_h, definido na Seção 2.2, e chamarmos $\Psi_{M+1} = \varphi_1,\ \Psi_{M+2} = \varphi_2,\ \cdots,\ \Psi_{M+Nel} = \varphi_{Nel}$, então $\{\Psi_1,\Psi_2,\cdots,\Psi_{M+Nel}\}$ será uma base para V_h^b, onde Nel indica o número de elementos da triangularização.

Desta forma, $(2.25)-(2.26)$ será equivalente a resolver, para cada nível de tempo, o sistema linear

$$\sum_{j=1}^{M+Nel} c_j^n \left[\langle\Psi_j,\Psi_i\rangle + k_n\langle 3c(\sum_l c_l^{n-1}\Psi_l)^2\nabla\Psi_j,\nabla\Psi_i\rangle + k_n\langle\vec{\beta}\cdot\nabla\Psi_j,\Psi_i\rangle\right] =$$

$$= \sum_{j=1}^{M+Nel} c_j^{n-1} \left[\langle \Psi_j, \Psi_i \rangle\right] + k_n \langle f, \Psi_i \rangle, \;\; i = 1, 2, \cdots M + nel,$$

com $M + Nel$ equações e $M + Nel$ incógnitas $c_1^n, c_2^n, \cdots, c_{M+Nel}^n$, com elementos definidos como em (2.16).

A seguir, vamos eliminar as bolhas do sistema, isto é, vamos isolar os coeficientes das funções bolha no sistema linear, para analisar melhor seu efeito sobre o problema.

Se em (2.25) tomarmos $v = \varphi_K(x)$ para $x \in K$ e $v = 0$ caso contrário, obtemos

$$\langle \frac{u_h^n - u_h^{n-1}}{k_n}, \varphi_K \rangle_K + \langle 3c(u^{n-1})^2 \nabla u_h^n, \nabla \varphi_K \rangle_K + \langle \vec{\beta} \cdot \nabla u_h^n, \varphi_K \rangle_K = \langle f, \varphi_K \rangle_K, \tag{2.27}$$

onde $\langle \cdot, \cdot \rangle_K$ indica o produto interno de $\mathcal{L}_2$ com integração sobre o elemento K.

Podemos decompor a solução u_h^n em sua parte de grau $\leq r$, $u_1^n \in V_h$, e sua parte gerada pelas bolhas, isto é,

$$u_h^n = u_1^n + \sum_K u_b^{nK} \varphi_K, \tag{2.28}$$

onde u_b^{nK} é o coeficiente da função bolha no elemento K.

Substituindo (2.28) em (2.27) e escrevendo $\alpha = 3c(u^{n-1})^2$ temos

$$\langle u_1^n, \varphi_K \rangle_K + k_n \langle \alpha \nabla u_1^n, \nabla \varphi_K \rangle_K + k_n \langle \vec{\beta} \cdot \nabla u_1^n, \varphi_K \rangle_K +$$

$$+ u_b^{nK} [\langle \varphi_K, \varphi_K \rangle_K + k_n \langle \alpha \nabla \varphi_K, \nabla \varphi_K \rangle_K + k_n \langle \vec{\beta} \cdot \nabla \varphi_K, \varphi_K \rangle_K] \; =$$

$$= \; k_n \langle f, \varphi_K \rangle_K + \langle u_h^{n-1}, \varphi_K \rangle_K.$$

Entretanto, para $\forall \; w_1 \in V_h$ temos, usando a fórmula de Green

$$\langle \nabla w_1, \nabla \varphi_K \rangle_K = -\langle \Delta w_1, \varphi_K \rangle_K + \int_{\partial K} \varphi_K \nabla w_1 \cdot \vec{\eta} ds = -\langle \Delta w_1, \varphi_K \rangle_K. \tag{2.29}$$

E então,

$$u_b^{nK} = \frac{\langle k_n f + u_h^{n-1} - u_1^n - k_n \vec{\beta} \cdot \nabla u_1^n + k_n \Delta u_1^n, \varphi_K \rangle_K}{\langle \varphi_K + k_n \alpha \nabla \varphi_K + k_n \vec{\beta} \cdot \nabla \varphi_K, \varphi_K \rangle_K}. \tag{2.30}$$

Este procedimento pode ser repetido para cada elemento $K \in T_h$. Se em (2.25) tomarmos $v = v_1^n \in V_h$ e usarmos (2.28), teremos

$$\langle u_1^n, v_1^n \rangle + k_n \langle \alpha \nabla u_1^n, \nabla v_1^n \rangle + k_n \langle \vec{\beta} \cdot \nabla u_1^n, v_1^n \rangle +$$

$$+ \sum_K u_b^{nK} [\langle \varphi_K, v_1^n \rangle_K + k_n \langle \alpha \nabla \varphi_K, \nabla v_1^n \rangle_K + k_n \langle \vec{\beta} \cdot \nabla \varphi_K, v_1^n \rangle_K] \; = \tag{2.31}$$

$$= \; k_n \langle f, v_1^n \rangle_K + \langle u_h^{n-1}, v_1^n \rangle.$$

Então, o problema variacional resultante (2.31) é equivalente a usar o método de Galerkin com mais um termo que em vista de (2.30) e da fórmula de Green pode ser reescrito como

$$\sum_K u_b^{nK} [\langle \varphi_K + k_n \vec{\beta} \cdot \nabla \varphi_K, v_1^n \rangle_K + k_n \langle \alpha \nabla \varphi_K, \nabla v_1^n \rangle_K] \; =$$

$$= \sum_K \frac{\langle k_n f + u_h^{n-1} - u_1^n - k_n \vec{\beta} \cdot \nabla u_1^n + k_n \Delta u_1^n, \varphi_K \rangle_K}{\langle \varphi_K + k_n \alpha \nabla \varphi_K + k_n \vec{\beta} \cdot \nabla \varphi_K, \varphi_K \rangle_K} \langle \varphi_K, v_1^n - k_n \vec{\beta} \cdot \nabla v_1^n - k_n \Delta v_1^n \rangle_K.$$

Usando os resultados de [2], este termo adicional ainda pode ser escrito como

$$\sum_K \frac{\langle k_n f + u_h^{n-1} - u_1^n - k_n \vec{\beta} \cdot \nabla u_1^n + k_n \Delta u_1^n, v_1^n - k_n \vec{\beta} \cdot \nabla v_1^n - k_n \Delta v_1^n \rangle_K}{\langle \varphi_K + k_n \alpha \nabla \varphi_K + k_n \vec{\beta} \cdot \nabla \varphi_K, \varphi_K \rangle_K} \overline{C} \| \varphi_K \|_K^2,$$

onde $\overline{C}$ é uma constante positiva.

Isto sugere uma alteração do espaço das funções-teste, ou seja, a inclusão de funções bolha funciona como um método tipo Petrov-Galerkin para estabilizar o problema variacional discreto.

Uma comparação entre os métodos Euler-*Streamline Diffusion* e Euler-Galerkin com funções bolha, mostra que ambos apresentaram resultados quantitativamente iguais, nas simulações que realizamos (Seção 2.6), fato que já era esperado em vista da estreita relação entre métodos do tipo Petrov-Galerkin e métodos de Galerkin mais bolhas, como podemos conferir em [18].

De forma mais geral, a escolha de um ou outro pode se basear no esforço computacional que cada um demanda. Enquanto o primeiro tem mais termos em sua forma bilinear, causando um maior número de integrações a serem calculadas para a formação do sistema linear para cada nível de tempo, o segundo resulta na construção de um sistema linear de dimensão maior, desde que estejamos usando interpolação polinomial de mesma ordem para ambos.

2.4 Método *Streamline Diffusion* Aplicado a Problemas Dependentes do Tempo

Nesta seção, introduziremos uma nova forma para discretizar $(2.1)-(2.4)$, onde por simplicidade tomaremos $g = 0$ em (2.2), que não passa pela semi-discretização do tempo descrita na Seção 2.2, mas utiliza a linearização descrita naquela seção. Para que seja possível computar soluções discretas a cada nível de tempo, tomaremos elementos finitos que envolvam as variáveis espaciais e temporais simultaneamente, abandonando assim o método de Euler para a discretização no tempo. Trata-se do método *Streamline Diffusion*, considerando elementos tridimensionais envolvendo espaço e tempo.

O método que descreveremos a seguir pode ser usado para melhorar a estabilidade, porém, sem o devido cuidado, pode levar a um sistema linear de dimensão exageradamente grande. A razão para isso é que o uso de funções-teste contínuas (na variável temporal) relaciona todos os níveis de tempo simultaneamente. Uma forma para evitar esta dificuldade, e decrescer o tamanho do correspondente sistema linear, é trabalhar em faixas de tempo S_n com a ajuda das funções de interpolação que serão contínuas no espaço e descontínuas no tempo, na fronteira comum entre duas faixas.

Na seqüência deste trabalho, nos referiremos a este método como Petrov-Galerkin Descontínuo no Tempo ou **PGDT**, para facilitar.

Seguindo as idéias contidas em [39] e adaptando-as para o problema não-linear de nosso interesse, tomamos $\Pi : \quad 0 = t_0 < t_1 < \cdots < t_N = T$ uma partição qualquer de $I = [0, T]$ e introduzimos as faixas de tempo S_n definidas por

$$S_n = \{(x,t) \ / \ x \in \Omega, \ t_{n-1} < t < t_n\},$$

para $n = 1, 2, \cdots, N$. Para cada n, seja V_h^n um subespaço de elementos finitos de $\mathcal{H}^1(S_n)$ e

$$V_h^{0n} = \{v \in V_h^n \ / \ v(x,t) = 0, x \in \partial\Omega_-\}.$$

Se aplicarmos o método *Streamline Diffusion* sucessivamente em cada faixa S_n, impondo a condição inicial para $t = t_{n-1}$ fracamente e as condições de contorno fortemente, obtemos o seguinte método:

$(\mathcal{V}_h)$ encontrar $u_h^n \in V_h^{0n}, \ n = 1, 2, \cdots N$, tal que

$$\langle \frac{\partial u_h^n}{\partial t} + \vec{\beta} \cdot \nabla u_h^n, v + \delta(\frac{\partial v}{\partial t} + \vec{\beta} \cdot \nabla v)\rangle^n + \langle 3c(u_-^{n-1})^2 \nabla u_h^n, \nabla v\rangle^n -$$

$$- \langle div(3c(u_-^{n-1})^2 \nabla u_h^n), \delta(\frac{\partial v}{\partial t} + \vec{\beta} \cdot \nabla v)\rangle^n + \langle\langle u_{h+}^n, v_+\rangle\rangle^{n-1} \ = \tag{2.32}$$

$$= \langle f, v + \delta(\frac{\partial v}{\partial t} + \vec{\beta} \cdot \nabla v)\rangle^n + \langle\langle u_{h-}^{n-1}, v_+\rangle\rangle^{n-1}, \ \forall \ v \in V_h^{0n},$$

onde $\delta = \overline{c}h$ se $3c(u_-^{n-1})^2 < h, \ \overline{c} > 0$ suficientemente pequena e $\delta = 0$ se $3c(u_-^{n-1})^2 \geq h$,

$$\langle w, v\rangle^n = \int_{S_n} w(x,t)v(x,t)dxdt,$$

$$\langle\langle w, v\rangle\rangle^n = \int_{\Omega} w(x,t_n)v(x,t_n)dxdt,$$

$$v_+(x,t) = \lim_{s \to 0^+} v(x, t+s),$$

$$v_-(x,t) = \lim_{s \to 0^-} v(x, t+s),$$

e $u_-^0 = u_0$, a condição inicial.

Nesta versão do método *Streamline Diffusion* para problemas de evolução, as funções-teste são dadas por

$$v + \delta(\frac{\partial v}{\partial t} + \vec{\beta} \cdot \nabla v),$$

já que a variável tempo é interpretada em conjunto com as variáveis espaciais.

Na prática, escolhemos δ segundo a proposta de Johnson [41]:

$$\delta_k = \overline{c}\ \frac{\max\left(h_k - \frac{3c(u^{n-1})^2}{|(1,\vec{\beta}_k)|}, 0\right)}{\mid (1, \vec{\beta}_k) \mid}, \tag{2.33}$$

onde $\overline{c}$ é uma constante positiva no intervalo $[0.5, 1]$ e $\mid (1, \vec{\beta}_k) \mid$ é o maior valor do módulo do vetor $(1, \vec{\beta})$, calculado entre os vértices do elemento k. Tal vetor representa o campo de velocidades, uma vez que o fluxo advectivo é dado, neste caso, por $\dfrac{\partial u}{\partial t} + \vec{\beta} \cdot \nabla u$.

Relembramos que esta é uma escolha local, ou seja, o parâmetro δ pode variar de elemento para elemento, dependendo da relação entre a difusão e advecção nos mesmos.

Sejam $\{\Psi_1, \Psi_2, \cdots, \Psi_M\}$, uma base para o espaço V_h, definido na Seção 2.2 por

$$V_h = \{\ v \in H_0^1\ /\ v\mid_K \in \mathcal{P}_r(K), \forall\ K \in T_h\ \},$$

e a seguinte base $\{\lambda_1, \lambda_2\}$ para o espaço dos polinômios de grau ≤ 1, definidos sobre o intervalo (t_{n-1}, t_n):

$$\lambda_1(t) = \frac{t_n - t}{t_n - t_{n-1}}, \quad \lambda_2(t) = \frac{t - t_{n-1}}{t_n - t_{n-1}}.$$

Podemos construir uma base $\{\Phi_1, \Phi_2, \cdots, \Phi_{2M}\}$ para V_h^{0n} através das relações:

$$\Phi_1(x,t) = \lambda_1(t)\Psi_1(x),\ \Phi_2(x,t) = \lambda_1(t)\Psi_2(x),\ \cdots, \Phi_M(x,t) = \lambda_1(t)\Psi_M(x),$$

$$\Phi_{M+1}(x,t) = \lambda_2(t)\Psi_1(x),\ \Phi_{M+2}(x,t) = \lambda_2(t)\Psi_2(x),\ \cdots, \Phi_{2M}(x,t) = \lambda_2(t)\Psi_M(x).$$

Então, sendo a solução aproximada em S_n dada por $u_h^n(x,t) = \sum_{j=1}^{2M} c_j^n \Phi_j(x,t)$, o problema (2.32) é equivalente ao seguinte sistema linear de ordem $2M$ nos coeficientes c_j^n:

$$\sum_{j=1}^{2M} c_j^n \left[\langle \frac{\partial \Phi_j}{\partial t} + \vec{\beta} \cdot \nabla \Phi_j, \Phi_i + \delta(\frac{\partial \Phi_i}{\partial t} + \vec{\beta} \cdot \nabla \Phi_i) \rangle^n + \langle 3c(\sum_l c_l^{n-1} \Phi_l)^2 \nabla \Phi_j, \nabla \Phi_i \rangle^n \right] +$$

$$+ \sum_{j=1}^{2M} c_j^n \left[- \langle div(3c(\sum_l c_l^{n-1} \Phi_l)^2 \nabla \Phi_j), \delta(\frac{\partial \Phi_i}{\partial t} + \vec{\beta} \cdot \nabla \Phi_i) \rangle^n + \langle \langle \Phi_{j+}, \Phi_{i+} \rangle \rangle^{n-1} \right] =$$

$$= \sum_{j=1}^{2M} c_j^{n-1} \left[\langle \langle \Phi_{j-}, \Phi_{i+} \rangle \rangle^{n-1} \right] + \langle f, \Phi_i + \delta(\frac{\partial \Phi_i}{\partial t} + \vec{\beta} \cdot \nabla \Phi_i) \rangle^n, \quad i = 1, 2, \cdots 2M.$$

Resolvido o sistema linear, tomamos a aproximação para $u(x, t_n)$, $n = 1, \cdots, N$, pela seguinte expressão

$$u^n(x) = u_h^n(x, t_n) = \sum_{j=M+1}^{2M} c_j^n \Psi_{j-M}(x),$$

isto é, utilizamos os coeficientes da interface superior da faixa S_n.

2.5 Método Semi-Lagrangiano

Introduziremos mais um método numérico para o problema de advecção-difusão (2.1)−(2.4). Trata-se do método Semi-Lagrangiano, bastante utilizado em modelos para a previsão do tempo, e que se baseia na aproximação por diferenças finitas das curvas características da equação a ser discretizada.

Vamos iniciar com a equação (2.1) escrita da seguinte forma

$$\frac{\partial u(x,t)}{\partial t} - c\Delta(u^3(x,t)) + \vec{\beta}(x,t) \cdot \nabla u(x,t) = f(x,t) \ \ , \ \ \Omega \times I. \tag{2.34}$$

Como de costume, seja uma partição $\Pi: \ 0 = t_0 < t_1 < \cdots < t_N = T$ de $I = [0, T]$ com $I_n = (t_{n-1}, t_n)$ e $k_n = t_n - t_{n-1}$. Se linearizarmos a equação (2.34) através de um atraso no tempo da parte não-linear, como já havíamos feito anteriormente, com $f = 0$ para facilitar a exposição, podemos reescrevê-la como

$$\frac{\partial u^n(x)}{\partial t} - c\, div(3(u^{n-1}(x))^2 \nabla u^n(x)) + \vec{\beta}^n(x) \cdot \nabla u^n(x) = 0. \tag{2.35}$$

Seguindo [48], uma aproximação Semi-Lagrangiana para (2.35) pode ser escrita como

$$\frac{u^n(x) - u^{n-1}(x-s)}{k_n} = \frac{c}{2}\left[div(3(u^{n-1}(x))^2 \nabla u^n(x)) + div(3(u^{n-1}(x-s))^2 \nabla u^{n-1}(x-s))\right], \tag{2.36}$$

onde o passo

$$s(x) = k_n \vec{\beta}(x - \frac{s(x)}{2}, t_{n-1} + \frac{k_n}{2}) \tag{2.37}$$

foi obtido por aproximação das curvas características através de um esquema de diferenças finitas centradas e com erro de segunda ordem.

Tomaremos uma dada malha e assumiremos o conhecimento dos valores de u para o tempo t_{n-1} e de $\vec{\beta}$ para o tempo $(t_{n-1} + \frac{k_n}{2})$ em todos os seus pontos.

Uma solução Semi-Lagrangiana para (2.34) é dada pelo seguinte algoritmo:

i) Resolvemos (2.37), que define $s(x)$ de forma implícita, iterativamente por

$$s_{l+1}(x) = k_n \vec{\beta}(x - \frac{s_l}{2}, t_{n-1} + \frac{k_n}{2}).$$

ii) Avaliamos

$$div[3(u^{n-1}(x))^2 \nabla u^{n-1}(x)]$$

nos pontos da malha e utilizamos uma fórmula de interpolação para calcular

$$u^{n-1}(x-s) + k_n \frac{c}{2} div[3(u^{n-1}(x-s))^2 \nabla u^{n-1}(x-s)].$$

iii) Resolvemos a equação

$$[u^n - k_n \frac{c}{2} div(3(u^{n-1})^2 \nabla u^n)] \mid_x = [u^{n-1} + k_n \frac{c}{2} div(3(u^{n-1})^2 \nabla u^{n-1})] \mid_{(x-s)} .$$

Observamos que, no passo (i) do algoritmo pode ser necessária a utilização de uma fórmula de interpolação para avaliar $\vec{\beta}$, caso este não seja conhecido fora dos nós da malha.

O passo (ii) é o responsável pela construção do lado direito da equação resolvida em (iii). Em ambos, métodos de diferenças finitas ou elementos finitos podem ser utilizados.

Para a realização da interpolação necessária nesta construção, surge uma dificuldade adicional: a localização do ponto $(x - s)$ na malha.

A maioria dos autores sugere a utilização de malhas de diferenças finitas, que tornam este trabalho muito mais simples, entretanto, pela própria característica de nosso trabalho, optamos pelo método de Galerkin.

Para isso, utilizamos a seguinte estratégia de localização de um dado ponto dentre os elementos de uma triangularização.

2.5.1 Localização em uma Malha de Elementos Finitos

Para descrevê-la, devemos conhecer de antemão a maior e a menor coordenadas x e y de cada triângulo da malha. Estes dados serão armazenados nos seguintes vetores:

xmax - vetor que contém a maior coordenada x de cada um dos elementos da malha;

ymax - vetor que contém a maior coordenada y de cada um dos elementos da malha;

xmin - vetor que contém a menor coordenada x de cada um dos elementos da malha;

ymin - vetor que contém a menor coordenada y de cada um dos elementos da malha.

A idéia principal da estratégia de localização de um ponto, que denotaremos por $(\tilde{x}, \tilde{y})$, consiste em, numa etapa inicial, considerarmos uma aproximação grosseira de cada elemento triangular k que compõe a malha, pela utilização de retângulos dados por

$$[xmin(k), xmax(k)] \times [ymin(k), ymax(k)].$$

Nesta fase, se pelo menos uma das seguintes condições não for satisfeita, então certamente o ponto $(\tilde{x}, \tilde{y})$ não pertencerá ao elemento k da triangularização: $xmin(k) < \tilde{x} < xmax(k)$ ou $ymin(k) < \tilde{y} < ymax(k)$.

Assim, investigando rapidamente cada um destes retângulos, podemos descartar os elementos que certamente não contêm o ponto $(\tilde{x}, \tilde{y})$ e ainda selecionar triângulos candidatos, isto é, que podem conter $(\tilde{x}, \tilde{y})$, para serem analisados com mais cuidado.

Numa segunda etapa, cada triângulo candidato com vértices $(x_1, y_1), (x_2, y_2)$ e (x_3, y_3), é levado em um triângulo padrão com vértices em $(0, 0), (1, 0)$ e $(0, 1)$, por meio da seguinte transformação linear:

$$\bar{x} = x_1 + (x_2 - x_1)\tilde{x} + (x_3 - x_1)\tilde{y},$$

$$\bar{y} = y_1 + (y_2 - y_1)\tilde{x} + (y_3 - y_1)\tilde{y}.$$

Por esta tranformação, calculamos o ponto $(\bar{x}, \bar{y})$ no sistema de coordenadas deste triângulo padrão, correspondente a $(\tilde{x}, \tilde{y})$.

Desta forma, fica fácil verificar se o ponto $(\bar{x}, \bar{y})$ pertence ao triângulo neste novo sistema de coordenadas, e, conseqüentemente, se $(\tilde{x}, \tilde{y})$ pertence ao triângulo em questão, bastando para isso que as seguintes condições sejam satisfeitas ao mesmo tempo: $\bar{x} + \bar{y} \leq 1$ e $\bar{x} \geq 0$ e $\bar{y} \geq 0$.

A fase inicial do algoritmo é bastante importante por poder ser realizada com rapidez, enquanto a segunda etapa apresenta um número maior de tarefas, sendo portanto mais demorada. Por isso, se realizássemos a busca utilizando somente a segunda fase, o algoritmo causaria o comprometimento do tempo de execução total do método Semi-Lagrangiano.

2.6 Experimentos Numéricos

Nesta seção, apresentamos simulações com o problema (2.1) − (2.4) utilizando os métodos numéricos introduzidos até aqui, visando tornar claras suas vantagens e desvantagens em tal tipo de aplicação.

2.6.1 Simulações com um Problema Teste

Nesta subseção, utilizaremos a solução exata (1.18) − (1.19) para testar e comparar alguns dos métodos apresentados até aqui.

Para isso, seja o domínio dado por $\Omega = [-3.5, 3.5] \times [-1, 1]$, com fronteiras

$$\partial\Omega_- = \{\ (x, y)\ /\ x = -3.5,\ y \in [-1, 1]\ \},$$

$$\partial\Omega_+ = \{\ (x, y)\ /\ x = 3.5,\ y \in [-1, 1]\ \} \cup$$

$$\cup\ \{\ (x, y)\ /\ y = -1,\ x \in [-3.5, 3.5]\ \}\ \cup\ \{\ (x, y)\ /\ y = 1,\ x \in [-3.5, 3.5]\ \}$$

e o conjunto de dados apresentado na Tabela 2.1.

Tal domínio pode ser encontrado na Figura 2.1, onde também apresentamos uma das triangularizações do tipo $(L_1(T_h))$ utilizadas nas próximas simulações. Nesta figura, o eixo horizontal representa o eixo-x e o vertical representa o eixo-y. Daqui para frente, por todo o texto, esta será a nossa convenção, sempre que não houver aviso contrário.

Observamos porém que, sendo este problema simétrico, poderíamos utilizar tal propriedade para economizar elementos na malha. Entretanto, como nas simulações dos próximos capítulos esta propriedade geralmente não estará presente, não faremos uso desta simplificação.

Cenário 1: Canal Aberto com Solução Exata

Parâmetro	**Valor**
c	1.0
$\vec{\beta}(x,t)$	$(1,0)$
f	0.0
g	0.0
k (Passo no tempo)	5×10^{-2}
$u_0(x,y)$	$\left[\max\{a^2 - \frac{1}{18}(x^2+y^2), 0\}\right]^{\frac{1}{2}}$
Elementos (Primeira ordem)	5888
Nós	3057
h_{min}	0.068

Tabela 2.1: Dados Relativos ao Cenário 1.

Na tabela acima, definimos o parâmetro h_{min} como

$$h_{min} = \min\{\ h_K\ /\ K \in T_h\ \},$$

isto é, o menor diâmetro dos elementos triangulares K da respectiva malha.

Já sabemos do Capítulo 1 que este problema tem por solução

$$u(x,t) = v(x - \vec{\beta}t, t),$$

sendo

$$v(x,t) = (t+1)^{-\frac{1}{3}} \left[\max\{a^2 - \frac{1}{18}(x^2+y^2)(t+1)^{-\frac{1}{3}}, 0\}\right]^{\frac{1}{2}},$$

ou seja, com o parâmetro $\tau = 1$, desde que tomemos o cuidado para que a constante a seja suficientemente pequena de forma a não permitir que o suporte de u toque as fronteiras do domínio.

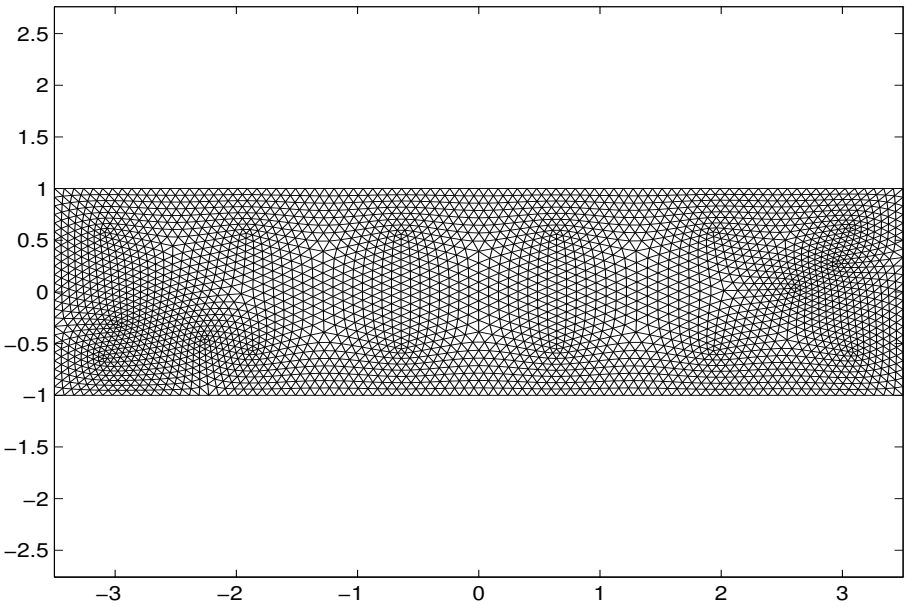

Figura 2.1: Domínio e Malha de Elementos Finitos para o Cenário 1.

As Tabelas 2.2−2.4 apresentam comparações entre a solução exata e as soluções numéricas para os dados do Cenário 1. Lembramos que o parâmetro $\overline{c}$ aparece na construção das funções-teste para os métodos **PGDT** e Euler-*Streamline Diffusion* e funciona como um controle da quantidade de difusão de cada método e que a escolha de tal parâmetro pode ser encontrada nas equações (2.33) e (2.24), respectivamente.

Na Tabela 2.2, verificamos que o decaimento da solução é mais rápido para os métodos que utilizam Euler para a discretização do tempo, enquanto na Tabela 2.3 podemos observar que o aumento de $\overline{c}$, seja para o método Euler-*Streamline Diffusion* quanto para **PGDT**, torna a solução "menos negativa", porque aumenta a difusão artificial gerada pelo método. Apesar disso, o erro absoluto entre as soluções exata e numérica, utilizando a norma-infinito, não diminui, como pode ser visto na Tabela 2.4.

Nestas mesmas tabelas, verificamos que o método Semi-Lagrangiano - tal qual os métodos que utilizam discretização de diferenças finitas para o tempo - também apresentou efeito difusivo maior que o desejado, porém, o erro absoluto entre a solução exata e a solução numérica calculada por este método, seja por interpolação linear ou quadrática, é da mesma ordem da solução de **PGDT**, que obteve os melhores resultados entre os métodos comparados.

Método	**t = 0.5**	**t = 1**	**t = 2**	**t = 3**
solução exata	0.1747	0.1586	0.1386	0.1260
Euler-*Streamline Diffusion* ($\overline{c} = 0.5$)	0.1532	0.1323	0.1118	0.0985
Euler-*Streamline Diffusion* ($\overline{c} = 1.0$)	0.1519	0.1298	0.1105	0.0951
PGDT ($\overline{c} = 0.5$)	0.1764	0.1582	0.1390	0.1263
PGDT ($\overline{c} = 1.0$)	0.1764	0.1581	0.1388	0.1261
Semi-Lagrangiano (interpolação linear)	0.1698	0.1534	0.1304	0.1185
Semi-Lagrangiano (interpolação quadrática)	0.1734	0.1553	0.1359	0.1248

Tabela 2.2: Máximos das soluções - $a = 0.2$.

Método	**t = 0.5**	**t = 1**	**t = 2**	**t = 3**
solução exata	0.0	0.0	0.0	0.0
Euler-*Streamline Diffusion* ($\overline{c} = 0.5$)	-0.115	-0.103	-0.104	-0.101
Euler-*Streamline Diffusion* ($\overline{c} = 1.0$)	-0.105	-0.099	-0.101	-0.085
PGDT ($\overline{c} = 0.5$)	-0.096	-0.088	-0.093	-0.084
PGDT ($\overline{c} = 1.0$)	-0.084	-0.088	-0.091	-0.069
Semi-Lagrangiano (interpolação linear)	-0.099	-0.095	-0.091	-0.088
Semi-Lagrangiano (interpolação quadrática)	-0.093	-0.088	-0.082	-0.084

Tabela 2.3: Mínimos das soluções - $a = 0.2$.

Método	**t = 0.5**	**t = 1**	**t = 2**	**t = 3**
Euler-*Streamline Diffusion* ($\overline{c} = 0.5$)	0.00561	0.00588	0.00645	0.00838
Euler-*Streamline Diffusion* ($\overline{c} = 1.0$)	0.00561	0.00588	0.00645	0.00838
PGDT ($\overline{c} = 0.5$)	0.00266	0.00278	0.00287	0.00445
PGDT ($\overline{c} = 1.0$)	0.00266	0.00278	0.00287	0.00445
Semi-Lagrangiano (interpolação linear)	0.00299	0.00308	0.00365	0.00492
Semi-Lagrangiano (interpolação quadrática)	0.00267	0.00280	0.00295	0.00455

Tabela 2.4: Erro absoluto entre as soluções - $a = 0.2$.

Para verificar o comportamento do erro, em relação ao refinamento da malha, apresentamos nas Tabelas $2.5 - 2.8$ simulações com diferentes malhas de elementos finitos, preservando os demais dados do Cenário 1. Tais resultados mostram o decaimento do erro, em todos os métodos apresentados, em função do refinamento da malha. Enfatizamos que o método **PGDT** obteve os melhores resultados nestas comparações.

h_{min}	$\mathbf{t = 0.5}$	$\mathbf{t = 1}$	$\mathbf{t = 2}$	$\mathbf{t = 3}$
0.068	0.00561	0.00588	0.00645	0.00838
0.052	0.00512	0.00533	0.00588	0.00625
0.037	0.00475	0.00504	0.00546	0.00591
0.018	0.00365	0.00402	0.00431	0.00487

Tabela 2.5: Erro - Euler-*Streamline Diffusion* ($\overline{c} = 0.5$).

h_{min}	$\mathbf{t = 0.5}$	$\mathbf{t = 1}$	$\mathbf{t = 2}$	$\mathbf{t = 3}$
0.068	0.00266	0.00278	0.00287	0.00445
0.052	0.00212	0.00243	0.00259	0.00396
0.037	0.00188	0.00225	0.00241	0.00356
0.018	0.00115	0.00147	0.00171	0.00211

Tabela 2.6: Erro - PGDT ($\overline{c} = 0.5$).

h_{min}	$\mathbf{t = 0.5}$	$\mathbf{t = 1}$	$\mathbf{t = 2}$	$\mathbf{t = 3}$
0.068	0.00299	0.00308	0.00365	0.00492
0.052	0.00237	0.00259	0.00278	0.00443
0.037	0.00209	0.00239	0.00259	0.00374
0.018	0.00148	0.00169	0.00192	0.00235

Tabela 2.7: Erro - Semi-Lagrangiano (interpolação linear).

h_{min}	$\mathbf{t = 0.5}$	$\mathbf{t = 1}$	$\mathbf{t = 2}$	$\mathbf{t = 3}$
0.068	0.00267	0.00280	0.00295	0.00455
0.052	0.00223	0.00256	0.00270	0.00405
0.037	0.00199	0.00234	0.00249	0.00365
0.018	0.00122	0.00158	0.00182	0.00219

Tabela 2.8: Erro - Semi-Lagrangiano (interpolação quadarática).

Nas Figuras $2.2 - 2.4$ comparamos o comportamento do suporte das soluções exata e numérica, **PGDT** e Euler-*Streamline Diffusion*, respectivamente, para os dados do Cenário 1 com $a = 0.25$. Podemos observar como a solução aproximada por este último método se espalhou excessivamente.

Tal comportamento pode ser diminuído se refinarmos a malha utilizada, mesmo assim, as soluções obtidas via Euler-*Streamline Diffusion* ou Semi-Lagrangiano apresentaram espalhamento muito maior que o desejado.

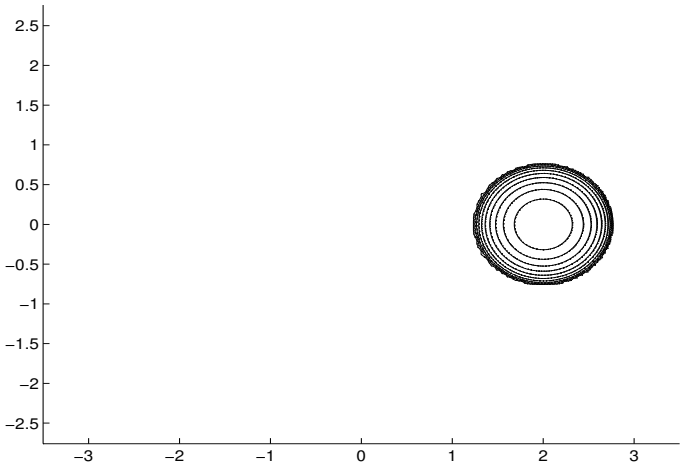

Figura 2.2: Solução Exata - $t = 2$.

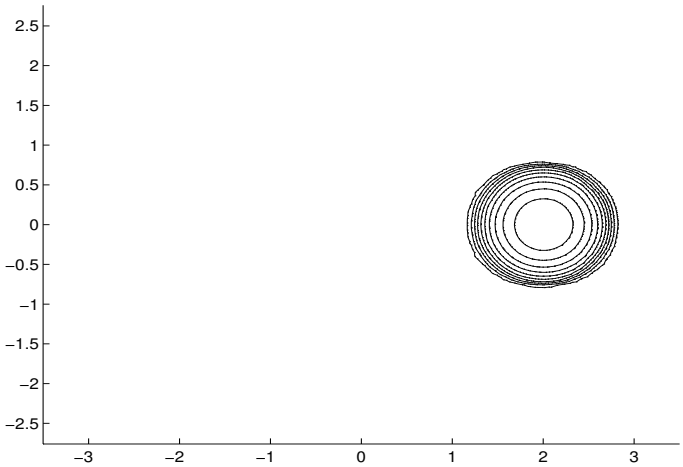

Figura 2.3: Solução de **PGDT** - $t = 2$.

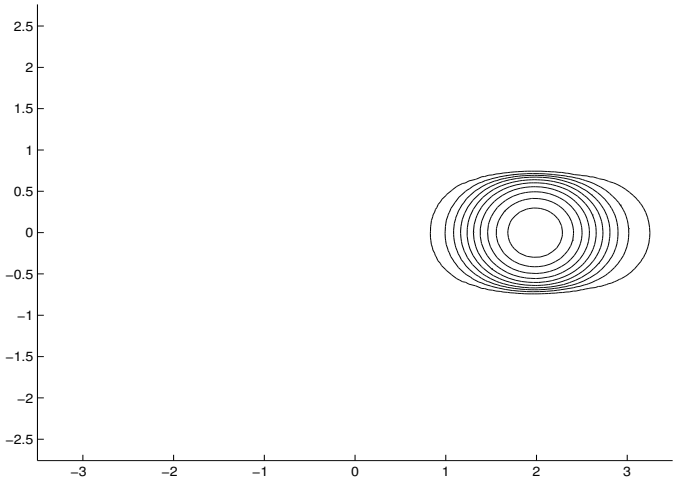

Figura 2.4: Solução de Euler-*Streamline Diffusion* - $t = 2$.

Na Tabela 2.9, apresentamos o tempo médio de execução de cada método, necessário para a resolução de um nível de tempo com todos os dados do Cenário 1. Todas as medidas de tempo foram realizadas num Pentium-II 300 MHz com 64 Mb de memória RAM.

Nesta tabela, podemos verificar que o tempo de CPU está diretamente ligado à dimensão do sistema linear resultante da discretização. Desta forma, a inclusão das bolhas não melhorou o resultado alcançado pelo método Euler-*Streamline Diffusion* e ainda levou visível desvantagem em tempo de CPU. Por isso, a partir da próxima seção deste trabalho optaremos apenas pela exibição dos resultados obtidos pelo método Euler-*Streamline Diffusion*.

Porém, isto não significa que devamos abandonar as bolhas. Através da implementação de um mecanismo conhecido como condensação estática ([20]), o desempenho de tal método pode ser melhorado pela eliminação, em cada submatriz que compõe o sistema linear, das variáveis correspondentes às funções bolha, uma vez que estas estão desacopladas das demais variáveis porque tais funções têm suporte em apenas um elemento.

Ainda examinando a Tabela 2.9, verificamos que o método **PGDT** apresenta tempo de processamento pouco competitivo, se comparado com o método Euler-*Streamline Diffusion*. Isto se explica pela necessidade de associar a cada nó da malha duas aproximações de u: $u_+(x,t)$ e $u_-(x,t)$, limites à direita e à esquerda de u na variável t, devido à descontinuidade no tempo das funções do espaço de aproximação e das funções-teste, levando à duplicação da dimensão de seus sistemas lineares.

Assim, os métodos **PGDT**, Euler-Galerkin com funções bolha e Semi-Lagrangiano demandam um tempo maior de processamento em comparação com o método Euler-*Streamline Diffusion*.

Método	Tempo de CPU	Dimensão
Euler-*Streamline Diffusion* ($\overline{c} = 0.5$)	3.1 s	3057
Euler-Galerkin com funções bolha	13.9 s	8945
PGDT ($\overline{c} = 0.5$)	10.2 s	6114
Semi-Lagrangiano (interpolação linear)	9.1 s	3057
Semi-Lagrangiano (interpolação quadrática)	10.5 s	3057

Tabela 2.9: Comparação entre os tempos de CPU.

2.6.2 Simulações em um Canal Aberto

O segundo conjunto de parâmetros para simulações numéricas é apresentado a seguir. Trata-se do mesmo domínio Ω do Cenário 1, com os demais dados definidos na tabela seguinte:

Cenário 2: Canal Aberto

Parâmetro	Valor
c	1×10^{-3}
$\vec{\beta}(x,t)$	$(1-y^2, 0)$
f	0.0
g	0.0
k (Passo no tempo)	5×10^{-2}
$u_0(x,y)$	$0.1exp(-16((x+2.5)^2+y^2))$
Elementos (Primeira ordem)	5888
Nós	3057
h_{min}	0.068

Tabela 2.10: Dados Relativos ao Cenário 2.

Nas simulações apresentadas nesta subseção, utilizamos $\overline{c} = 0.5$ nos métodos Euler-*Streamline Diffusion* e **PGDT**. Lembramos que tal parâmetro aparece na construção das funções-teste destes métodos, conforme as escolhas apresentadas em (2.24) e (2.33), respectivamente. Em ambos os métodos, $\overline{c}$ funciona como um controle da quantidade de difusão introduzida pelos mesmos.

Nas Figuras 2.5 e 2.6, apresentamos um corte paralelo ao eixo-x na solução, para simulações com os métodos Euler-Galerkin e Euler-*Streamline Diffusion*, respectivamente, utilizando os dados do Cenário 2. Na primeira, verificamos o aparecimento de oscilações na região de mudança de tipo da equação, enquanto na segunda tais oscilações não estão mais presentes.

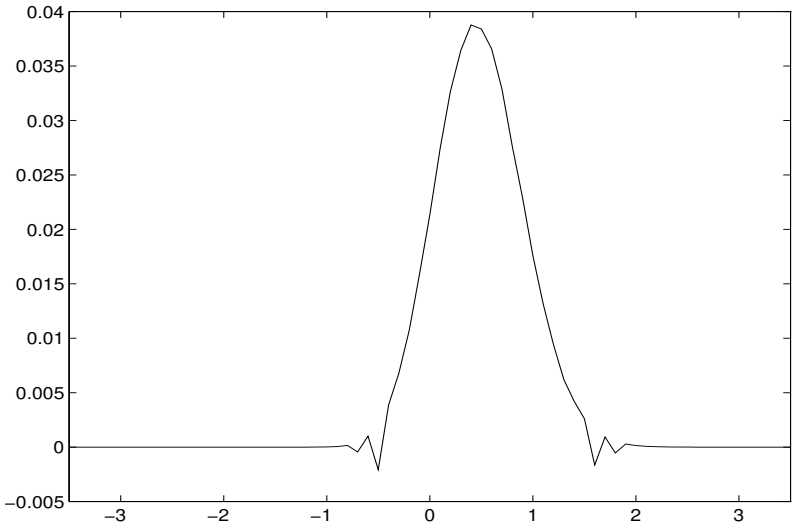

Figura 2.5: Corte na Solução de Euler-Galerkin - $t = 4.5$.
Eixo horizontal: x, eixo vertical: u.

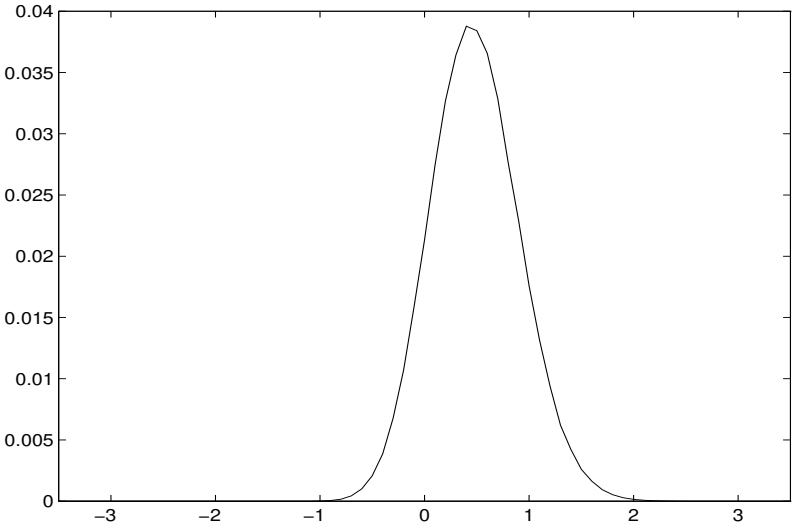

Figura 2.6: Corte na Solução de Euler-*Streamline Diffusion* - $t = 4.5$.
Eixo horizontal: x, eixo vertical: u.

Cabe ressaltar que, embora nosso objetivo não seja este, refinamentos da malha podem diminuir e até eliminar, em alguns casos, as oscilações presentes na Figura 2.5. Nesta simulação, por exemplo, se utilizarmos o método Euler-Galerkin com $h_{min} = 0.018$ ao invés de $h_{min} = 0.068$, podemos obter a estabilidade necessária. Entretanto, nosso intuito continua sendo trocar a necessidade de refinamentos custosos pelo uso de métodos projetados para este tipo específico de situação.

As Figuras 2.7-2.9 apresentam as curvas de nível da evolução da solução do problema (2.1)−(2.4) computada pelos método Euler-*Streamline Diffusion*, Euler-Galerkin com funções bolha e **PGDT**, respectivamente, para os tempos $t = 0,\ 3.0,\ 4.5$ e 6.0.

Nestes experimentos, o campo de velocidades $\vec{\beta}$ corresponde a um escoamento laminar bidimensional entre placas paralelas. Observando as Figuras 2.7 e 2.8, verificamos que à medida que o tempo aumenta, a parte frontal da mancha de óleo toma a forma parabólica ditada pela primeira componente do campo de velocidades, entretanto, o mesmo não acontece com sua região anterior.

Este comportamento indica excesso de difusão artificial nos métodos Euler-*Streamline Diffusion* e Euler-Galerkin com funções bolha, como já havíamos verificado nas simulações com o Cenário 1, e não ocorre nas simulações com o método **PGDT**, como pode ser visto na Figura 2.9. Além disso, verificamos que o suporte das soluções apresentadas nas Figuras 2.7 e 2.8 é muito maior se comparado ao suporte mostrado pela Figura 2.9, evidenciando mais uma vez que o método **PGDT** acrescenta menos difusão para atingir a estabilização.

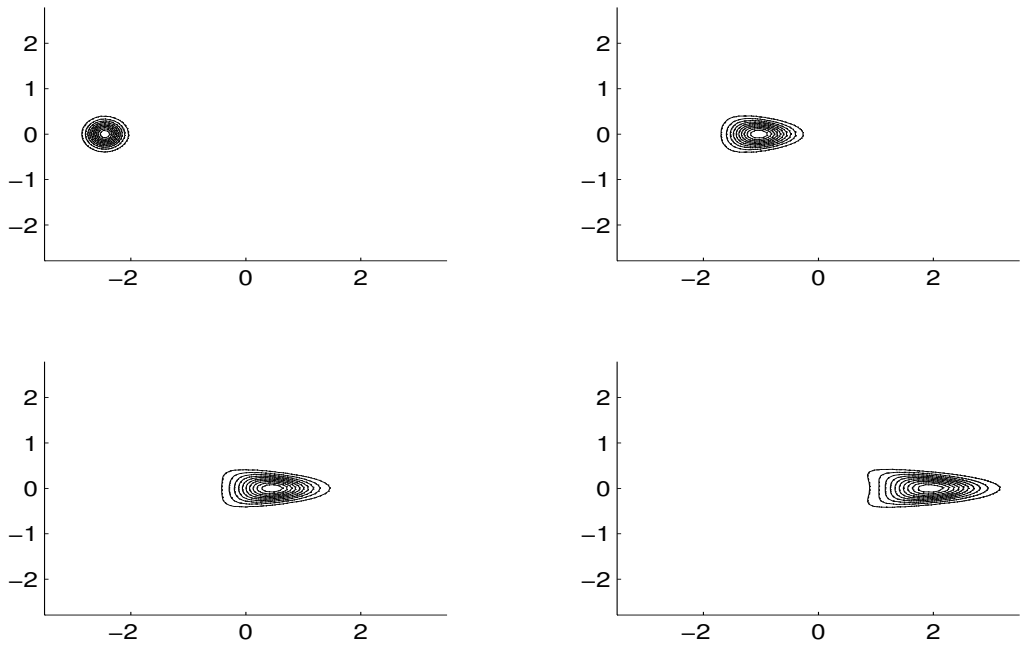

Figura 2.7: Euler-*Streamline Diffusion* para o Cenário 2 - $t = 0,\ 3.0,\ 4.5$ e 6.0.

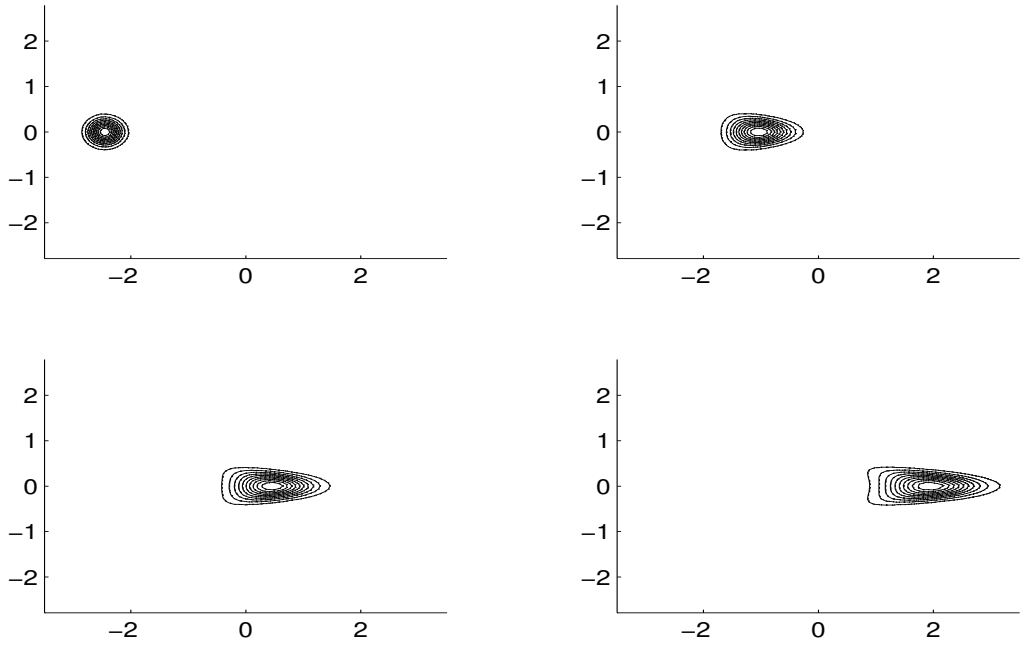

Figura 2.8: Euler-Galerkin com Funções Bolha para o Cenário 2 - $t = 0,\ 3.0,\ 4.5$ e 6.0.

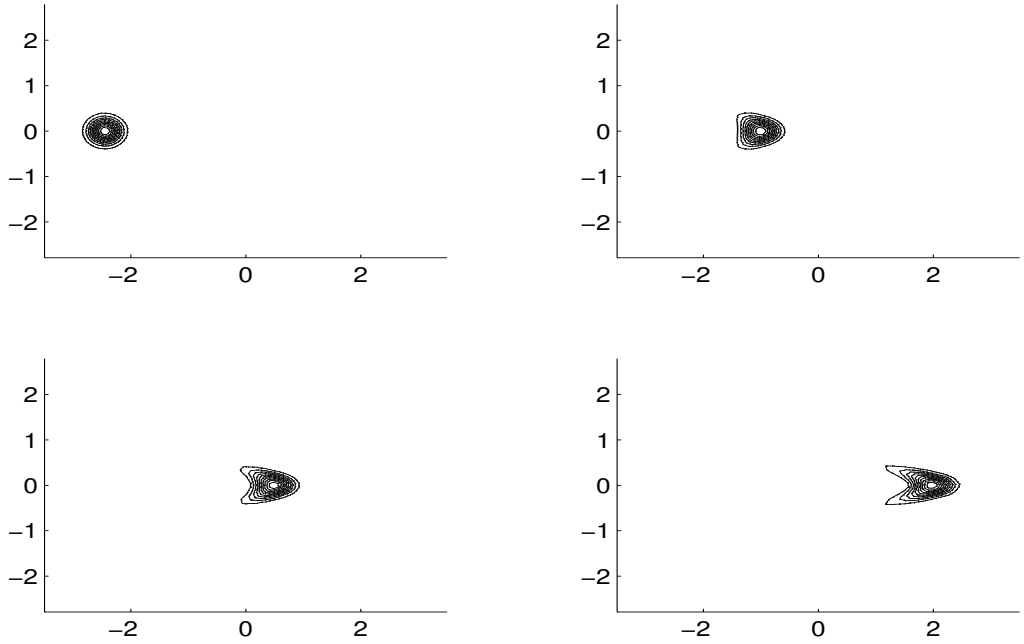

Figura 2.9: Método $\mathbf{PGDT}$ para o Cenário 2 - $t = 0,\ 3.0,\ 4.5$ e 6.0.

Novamente, comparando os resultados das Figuras 2.7 e 2.8 observamos claramente que estes são iguais, e lembramos que este fato já era esperado em virtude da equivalência entre os métodos Euler-*Streamline Diffusion* e Euler-Galerkin com funções bolha.

2.6.3 Simulações em um Canal com Ilha

O terceiro conjunto de parâmetros para simulações numéricas está definido a seguir:

Cenário 3: Canal com Ilha

Parâmetro	**Valor**
c	1×10^{-3}
$\vec{\beta}(x,t)$	$\left(1 - \frac{2x^2r^2}{(x^2+y^2)^2} + \frac{r^2}{(x^2+y^2)}\ ,\ -\frac{2xyr^2}{(x^2+y^2)^2}\right)$
f	0.0
g	0.0
k (Passo no tempo)	5×10^{-2}
$u_0(x,y)$	$0.1exp(-16((x+2.5)^2+y^2))$
Elementos (Primeira ordem)	3520
Nós	1840
h_{min}	0.089

Tabela 2.11: Dados Relativos ao Cenário 3.

Neste caso, o domínio Ω é composto pelo retângulo dado por $[-3.5, 3.5] \times [-1, 1]$ e um obstáculo dado pela circunferência de centro na origem e raio $r = 0.25$. Suas fronteiras são descritas por

$$\partial\Omega_- = \{\ (x,y)\ /\ x = -3.5,\ y \in [-1,1]\ \},$$

$$\partial\Omega_+ = \{\ (x,y)\ /\ x^2 + y^2 = r^2\ \}\ \cup\ \{\ (x,y)\ /\ x = 3.5,\ y \in [-1,1]\ \} \cup$$

$$\cup\ \{\ (x,y)\ /\ y = -1,\ x \in [-3.5, 3.5]\ \}\ \cup\ \{\ (x,y)\ /\ y = 1,\ x \in [-3.5, 3.5]\ \}.$$

Na Figura 2.10, encontramos o domínio do Cenário 3 juntamente com uma das triangularizações utilizadas. Observamos que sua malha é mais refinada que a do Cenário 1 apenas em volta do obstáculo, apresentado menos elementos que a malha do primeiro cenário.

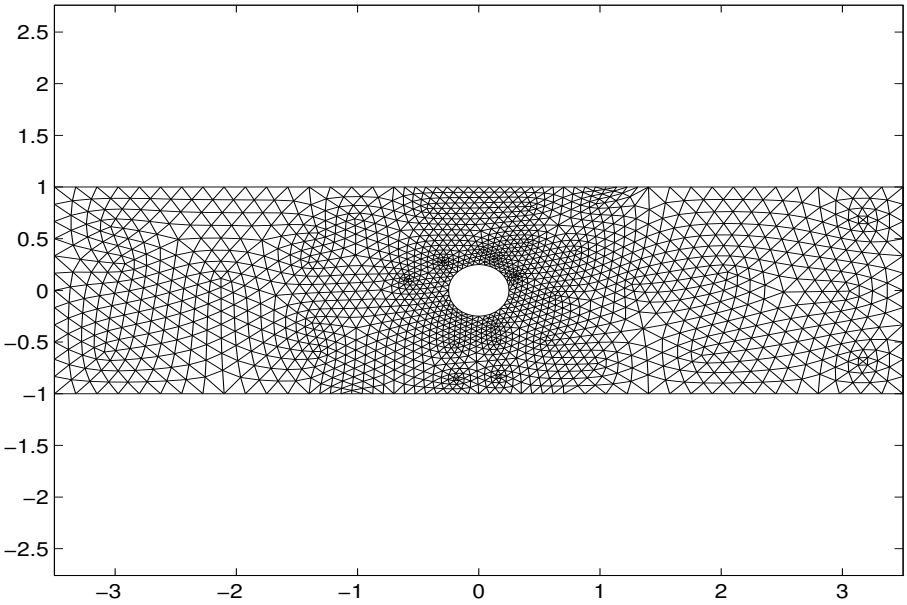

Figura 2.10: Domínio e Malha de Elementos Finitos para o Cenário 3.

As Figuras 2.11-2.13 apresentam as curvas de nível da evolução da solução do problema (2.1)−(2.4) computada pelos método Euler-*Streamline Diffusion*, Euler-Galerkin com funções bolha e **PGDT**, respectivamente, para os tempos $t = 0,\ 2.0,\ 3.0$ e 5.0. Como nas simulações anteriores, utilizamos $\overline{c} = 0.5$ nos métodos Euler-*Streamline Diffusion* e **PGDT**.

Podemos reparar que o campo de velocidades dado por

$$\vec{\beta} = \left(1 - \frac{2x^2r^2}{(x^2+y^2)^2} + \frac{r^2}{(x^2+y^2)} \ , \ -\frac{2xyr^2}{(x^2+y^2)^2} \right),$$

tem a propriedade de ser quase constante e paralelo ao eixo-x longe do obstáculo e tangente nas suas proximidades, de forma a contornar a ilha de raio r.

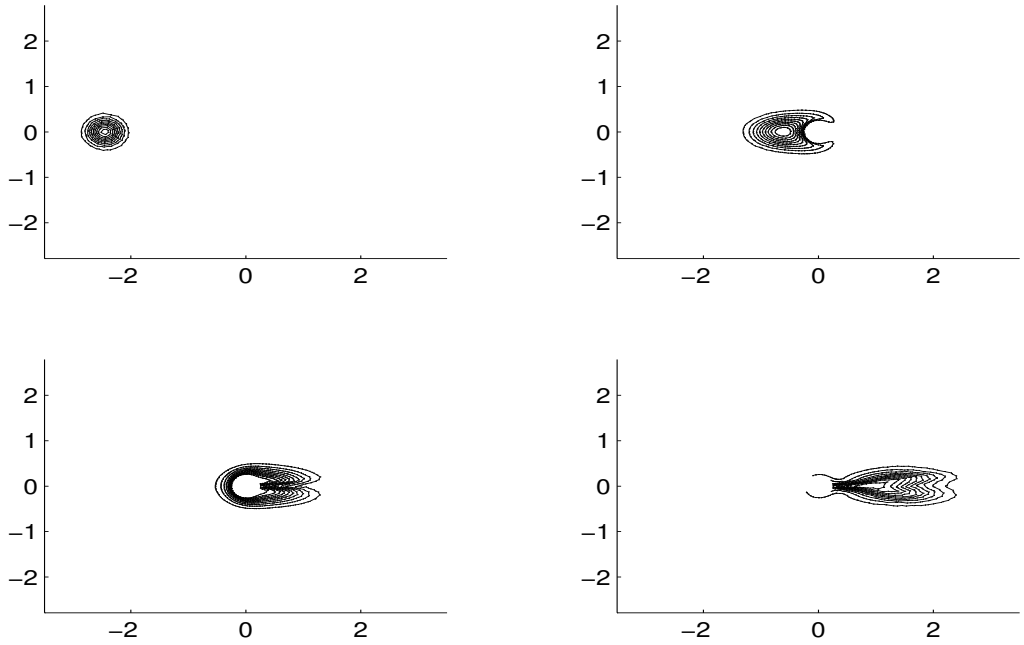

Figura 2.11: Euler-*Streamline Diffusion* para o Cenário 3 - $t = 0,\ 2.0,\ 3.0$ e 5.0.

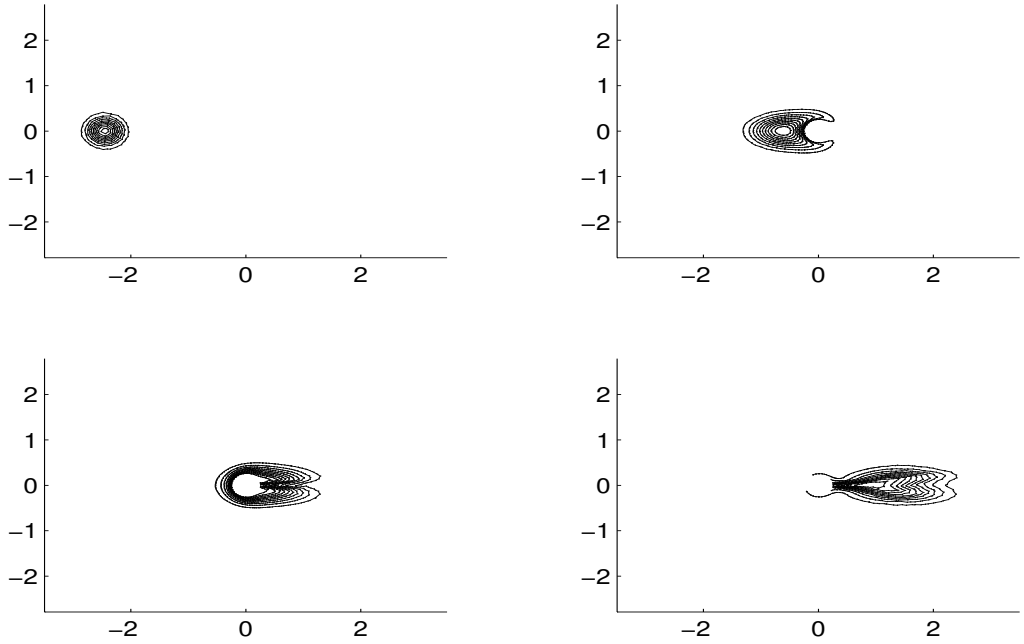

Figura 2.12: Euler-Galerkin com Funções Bolha para o Cenário 3 - $t = 0,\ 2.0,\ 3.0$ e 5.0.

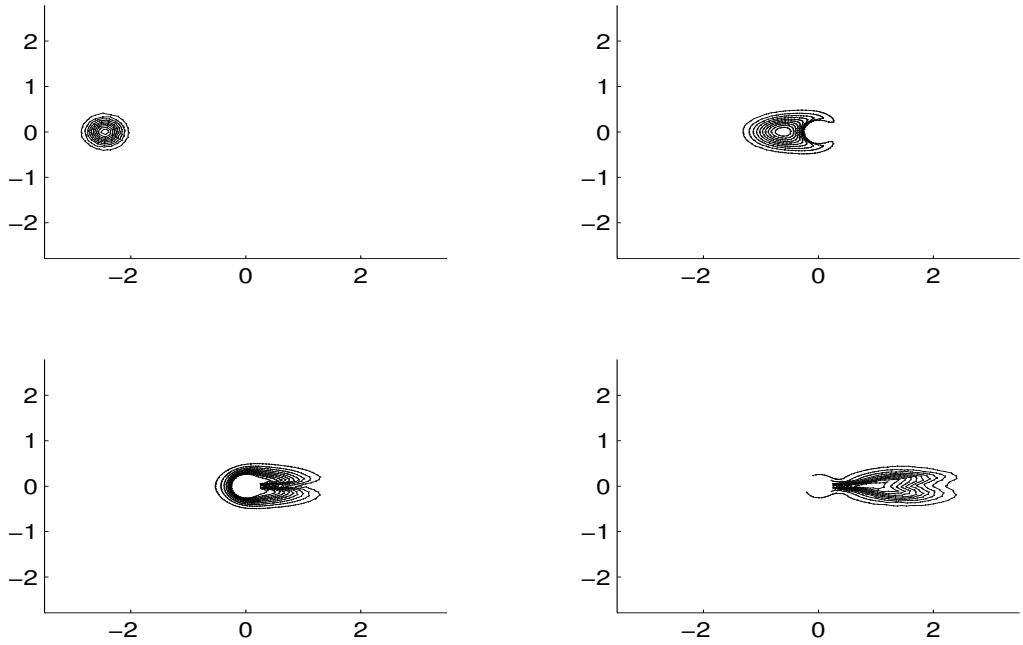

Figura 2.13: Método **PGDT** para o Cenário 3 - $t = 0,\ 2.0,\ 3.0$ e 5.0.

Estas figuras não nos permitem tirar maiores conclusões. Aparentemente a ilha foi contornada sem problemas. Entretanto, uma verificação na Tabela 2.12 nos mostra que o valor máximo atingido pela solução, em cada nível de tempo, é menor nos casos em que os métodos de Euler ou Semi-Lagrangiano foram aplicados, também indicando presença de maior difusão artificial nestes tipos de discretização, embora este último tenha apresentado resultados mais próximos do esperado no caso da utilização de interpolação com funções de base quadráticas.

Na mesma tabela, ressaltamos o comportamento idêntico entre os métodos Euler-Galerkin com funções bolha e Euler-*Streamline Diffusion*.

Método	$\mathbf{t = 1}$	$\mathbf{t = 2}$	$\mathbf{t = 3}$	$\mathbf{t = 4}$	$\mathbf{t = 5}$
Euler-*Streamline Diffusion* ($\overline{c} = 0.5$)	0.073	0.060	0.047	0.040	0.035
Euler-Galerkin com funções bolha	0.073	0.060	0.047	0.040	0.035
PGDT ($\overline{c} = 0.5$)	0.090	0.085	0.076	0.072	0.066
Semi-Lagrangiano (interpolação linear)	0.082	0.074	0.069	0.060	0.055
Semi-Lagrangiano (interpolação quadrática)	0.085	0.079	0.072	0.065	0.059

Tabela 2.12: Valores máximos das soluções aproximadas - Cenário 3.

Método	Tempo de CPU	Dimensão
Euler-*Streamline Diffusion* ($\overline{c} = 0.5$)	1.9 s	1840
Euler-Galerkin com funções bolha	8.3 s	5360
PGDT ($\overline{c} = 0.5$)	6.1 s	3680
Semi-Lagrangiano (interpolação linear)	5.5 s	1840
Semi-Lagrangiano (interpolação quadrática)	6.3 s	1840

Tabela 2.13: Comparação entre os tempos de CPU.

Examinando a Tabela 2.13, onde apresentamos o tempo médio de execução de cada método necessário para a resolução de um nível de tempo com os dados do Cenário 3, novamente verificamos que o tempo de CPU está diretamente ligado à dimensão do sistema linear resultante da discretização, assim, verificamos que o método **PGDT** apresenta tempo de processamento pouco competitivo, se comparado ao método Euler-*Streamline Diffusion*.

As simulações realizadas até aqui nos levam a concluir que o método **PGDT** alcançou os melhores resultados dentre os métodos estudados, já que ele consegue alcançar estabilidade numérica com a menor quantidade de difusão artificial. Porém, o tempo de CPU deste método mostrou-se exagerado, podendo ser melhorado. Tal aspecto, assim como a propriedade de conservação da massa serão abordados no próximo capítulo.

Antes, porém, estaremos examinando, na proxima seção, outras maneiras de linearizar o problema.

2.7 Outras Possibilidades de Linearização

Em todos os métodos de aproximação para o problema de espalhamento de manchas de óleo, apresentados neste trabalho, realizamos a linearização do termo difusivo da equação (2.1) por atraso no tempo.

Nesta seção, descreveremos outras formas clássicas para linearizar tal equação e apresentaremos comparações entre os resultados obtidos por estas formas e a linearização que utilizamos até aqui.

2.7.1 Linearização por Relaxações Sucessivas

Para descrever uma outra maneira de linearizar (2.1), resolveremos a equação (2.6), que corresponde à sua linearização e discretização no tempo pelo método de Euler Regressivo, e corrigiremos a solução através de relaxações sucessivas. Este algoritmo de linearização, também conhecido como **SOR**, fica então descrito da seguinte forma:

Para $n = 1, 2, \cdots, N$

$$\frac{u^n - u^{n-1}}{k_n} - c\, div(3(u^{n-1})^2 \nabla u^n) + \vec{\beta}(x, t_n) \cdot \nabla u^n = f(x, t) \tag{2.38}$$

$l = 1$
$w^0 = u^n$
Repetir

$$\frac{w^l - u^{n-1}}{k_n} - c\, div(3(w^{l-1})^2 \nabla w^l) + \vec{\beta}(x, t_n) \cdot \nabla w^l = f(x, t) \tag{2.39}$$

$w^l = \theta w^l + (1 - \theta) w^{l-1}$
$l = l + 1$

até que $\|w^l - w^{l-1}\|_2 < Precisão$.
$u^n = w^l$
$n = n + 1$.

Aqui, $\|\cdot\|_2$ representa a norma euclidiana de $\mathbb{R}^n$ e $\theta \in [0, 1]$ é o parâmetro de relaxação. Lembramos que u^0 é a condição inicial do problema e, portanto, é conhecida.

Para resolver (2.38) e (2.39) , ambas com condições de contorno $(2.2) - (2.3)$, podemos aplicar qualquer dos métodos numéricos descritos nas seções anteriores deste capítulo. No final desta seção, apresentaremos comparações entre as formas de linearização em conjunto com o método *Streamline Diffusion*, escolhido de forma a exemplificar o procedimento.

2.7.2 O Método de Newton

Para aplicar o método de Newton, vamos retomar a equação (2.6) sem o atraso no tempo de sua parte não-linear:

$$\frac{u^n - u^{n-1}}{k_n} - c\, div(3(u^n)^2 \nabla u^n) + \vec{\beta}(x, t_n) \cdot \nabla u^n = f(x, t). \tag{2.40}$$

Desta forma, a cada nível de tempo, estamos interessados em resolver o problema

$$P(u^n) = 0, \tag{2.41}$$

onde

$$P(u^n) = -k_n c\ div(3(u^n)^2 \nabla u^n) + k_n \vec{\beta}(x, t_n) \cdot \nabla u^n + u^n - u^{n-1} - k_n f(x, t).$$

O método de Newton gera a seguinte seqüência para resolver (2.40) :

Para $l = 0, 1, 2, \cdots$

$$P^{'}(u^{nl})[w^{l+1}] = -P(u^{nl}), \tag{2.42}$$

$$u^{n(l+1)} = w^{l+1} + u^{nl}, \tag{2.43}$$

onde a notação u^{nl} significa a função u calculada no n-ésimo tempo e na l-ésima iteração de Newton.

Apresentaremos as definições e cálculos seguintes com intuito didático, isto é, de forma a propiciar que este trabalho atinja um público maior.

A derivada direcional do operador P, avaliada no ponto u e na direção de v é definida por

$$P^{'}(u)[v] = \frac{d}{d\lambda}\{P(u + \lambda v)\} \mid_{\lambda=0} .$$

Para o operador P de nosso problema, podemos calcular a derivada direcional, que será dada por

$$P^{'}(u)[v] = -3k_n c\ div(2vu\nabla u + u^2 \nabla v) + k_n \vec{\beta}(x, t_n) \cdot \nabla v + v. \tag{2.44}$$

Desta forma, podemos escrever:

Para $l = 0, 1, 2, \cdots$

$$-3k_n c\ div(2w^{l+1}u^{nl}\nabla u^{nl} + (u^{nl})^2\nabla w^{l+1}) + k_n\vec{\beta}(x,t_n)\cdot\nabla w^{l+1} + w^{l+1} \ =$$

$$= \ 3k_n c\ div((u^{nl})^2\nabla u^{nl}) - k_n\vec{\beta}(x,t_n)\cdot\nabla u^{nl} - u^{nl} + u^{n-1} + k_n f(x,t) \qquad (2.45)$$

$$u^{n(l+1)} = w^{l+1} + u^{nl}.$$

Levando em consideração a condição inicial (2.4), podemos escrever a formulação variacional discreta de (2.42), usando o método *Streamline Diffusion*:

Para $n = 1, 2, \cdots, N$,

$(\mathcal{V}_h)$ encontrar $w_h^{l+1} \in V_h^g,\ l = 0, 1, 2, \cdots$, tal que

$$-3k_n c\langle div\{2w_h^{l+1}u_h^{nl}\nabla u_h^{nl} + (u_h^{nl})^2\nabla w_h^{l+1}\} + k_n\vec{\beta}\cdot\nabla w_h^{l+1} + w_h^{l+1}, v + \delta\vec{\beta}\cdot\nabla v\rangle \ =$$

$$= \ 3k_n c\langle div((u_h^{nl})^2\nabla u_h^{nl}) - k_n\vec{\beta}\cdot\nabla u_h^{nl} - u_h^{nl} + u_h^{n-1} + k_n f, v + \delta\vec{\beta}\cdot\nabla v\rangle,\ \forall\ v \in V_h,$$

$$\langle u_h^0, v + \delta\vec{\beta}\cdot\nabla v\rangle = \langle u_0, v + \delta\vec{\beta}\cdot\nabla v\rangle,\ \forall\ v \in V_h,$$

onde $\delta = \overline{c}h$ se $3c(u^{n-1})^2 < h,\ \overline{c} > 0$ suficientemente pequena e $\delta = 0$ se $3c(u^{n-1})^2 \geq h$.

Lembramos que, na prática, escolhemos δ localmente segundo a proposta de Johnson [41]:

$$\delta_k = \overline{c}\ \frac{\max\left(h_k - \frac{3c(u^{n-1})^2}{|\vec{\beta}_k|}, 0\right)}{\mid \vec{\beta}_k \mid},$$

onde $\overline{c}$ é uma constante positiva no intervalo $[0.5, 1]$ e $\mid \vec{\beta}_k \mid$ é o maior valor do módulo do vetor $\vec{\beta}$, calculado entre os vértices do elemento k.

2.7.3 Comparações e Conclusões

Nas próximas tabelas, comparamos as linearizações utilizando o atraso no tempo via método de Euler Regressivo, método de Newton e relaxações sucessivas (SOR), todas em conjunto com o método *Streamline Diffusion* com $\bar{c} = 0.5$, indicados por Euler-SD, Newton-SD e SOR-SD, respectivamente, com os dados descritos no Cenário 2, definido na seção anterior.

O erro absoluto entre as soluções aproximadas por dois métodos diferentes, indicado nas tabelas, se refere a

$$\|U - V\|_\infty,$$

enquanto o erro relativo é dado por

$$\frac{\|U - V\|_\infty}{\|U\|_\infty},$$

onde U e V são vetores que representam as soluções em cada nó da malha e $\|\cdot\|_\infty$ é a norma-infinito para vetores de $\mathbb{R}^n$.

Métodos	**Erro Absoluto**	**Erro Relativo**
Euler-SD × Newton-SD	1.97×10^{-6}	3.28×10^{-5}
Euler-SD × SOR-SD	4.57×10^{-6}	7.60×10^{-5}
SOR -SD × Newton-SD	4.42×10^{-6}	7.35×10^{-5}

Tabela 2.14: Erros entre os métodos de linearização - $t = 1$.

Métodos	**Erro Absoluto**	**Erro Relativo**
Euler-SD × Newton-SD	6.47×10^{-7}	1.84×10^{-5}
Euler-SD × SOR-SD	1.64×10^{-6}	4.66×10^{-5}
SOR -SD × Newton-SD	1.31×10^{-6}	3.72×10^{-5}

Tabela 2.15: Erros entre os métodos de linearização - $t = 4$.

Métodos	**Erro Absoluto**	**Erro Relativo**
Euler-SD × Newton-SD	3.92×10^{-7}	1.35×10^{-5}
Euler-SD × SOR-SD	1.10×10^{-6}	3.79×10^{-5}
SOR -SD × Newton-SD	8.25×10^{-7}	2.85×10^{-5}

Tabela 2.16: Erros entre os métodos de linearização - $t = 6$.

A análise das Tabelas $2.14 - 2.16$ mostra que as diferentes formas de linearização conduziram a resultados muito próximos. Este fato se repete com maior intensidade ao realizarmos o refinamento da malha, conforme pode ser visto nas Tabelas $2.17 - 2.18$. Ressaltamos que este também é o comportamento dos demais métodos de discretização descritos até aqui, quando associados aos métodos de Newton ou **SOR**.

Desta forma, na seqüência deste trabalho, optaremos por continuar a utilizar a linearização por atraso no tempo, principalmente devido ao menor esforço computacional necessário para a sua execução.

Em vista dos resultados alcançados até aqui, o algoritmo composto por este tipo de linearização em conjunto com o método **PGDT**, será modificado no próximo capítulo de forma a aumentar sua eficiência.

h_{min}	**Erro Absoluto**	**Erro Relativo**
0.068	3.92×10^{-7}	1.35×10^{-5}
0.052	3.54×10^{-7}	1.14×10^{-5}
0.037	2.98×10^{-7}	1.02×10^{-5}
0.018	2.03×10^{-7}	9.96×10^{-6}

Tabela 2.17: Erros entre os métodos Euler-SD × Newton-SD - $t = 6$.

h_{min}	**Erro Absoluto**	**Erro Relativo**
0.068	1.10×10^{-6}	3.79×10^{-5}
0.052	9.92×10^{-7}	3.64×10^{-5}
0.037	8.78×10^{-7}	3.12×10^{-5}
0.037	8.12×10^{-7}	2.97×10^{-5}

Tabela 2.18: Erros entre os métodos Euler-SD × SOR-SD - $t = 6$.

Capítulo 3

Um Método *Streamline Diffusion* Descontínuo no Tempo com Controle de Massa

Neste capítulo, pretendemos introduzir algumas modificações no algoritmo do método **PGDT** que, dentre os métodos examinados até aqui, conseguiu o melhor desempenho em relação à qualidade das aproximações.

A primeira dessas modificações está diretamente ligada à análise realizada na Seção 1.3, e que nos levou a concluir que o problema (1.20) – (1.23), descrito novamente a seguir, tem a importante propriedade de que a integral de u sobre o domínio é totalmente determinada pela correspondente integral de f, o que confere ao problema uma característica de balanço (conservação) da massa total da equação.

A segunda modificação se refere à forma com que o algoritmo foi implementado. Levando em consideração o suporte compacto da solução do problema (1.20) – (1.23), isto é, o fato de que a solução se anula fora de um conjunto fechado e limitado contido em Ω, desenvolvemos um mecanismo que nos permite acompanhá-la ao longo do tempo, resolvendo o problema apenas numa parte do domínio computacional, isto é, num subdomínio que contenha o suporte da solução. Com isto, em cada nível de tempo o sistema linear resultante apresenta sensível diminuição de sua ordem e, conseqüentemente, o tempo de execução do algoritmo decresce também. É importante ressaltar que, ambas as modificações são contribuições originais deste trabalho.

Além disso, fechando o capítulo, incorporamos ao problema um termo de degradação, também não-linear, de forma a incluir nas simulações a possibilidade de desintegração de parte da mancha de petróleo através deste mecanismo.

3.1 Controle da Massa

Iniciaremos recolocando o problema não-linear de advecção-difusão:

$$\frac{\partial u(x,t)}{\partial t} - c\Delta((u(x,t))^3) + \vec{\beta}(x,t)\cdot\nabla u(x,t) = f(x,t) \ \ , \ \ \Omega\times I,$$

$$u(x,t) = g(x) \ , \ x \in \ \partial\Omega_- \ , \ \ t\in I,$$

$$\frac{\partial u(x,t)}{\partial \vec{\eta}} = 0 \ , \ \ x \in \ \partial\Omega_+ \ , \ \ t\in I,$$

$$u(x,0) = u_0(x) \ , \ x\in\Omega.$$

Na Seção 1.3, concluímos que

$$\frac{\partial}{\partial t}\int_\Omega u dx = \int_\Omega f dx,$$

que implica em, no caso em que $f\equiv 0$,

$$\frac{\partial}{\partial t}\int_\Omega u(x,t) dx = 0,$$

isto é, a massa da solução deve ser conservada ao longo do tempo, enquanto u não entrar em contato com o bordo $\partial\Omega$.

Tal característica de balanço de massa é obtida *a priori*, na construção do método de elementos finitos, e deve ser garantida pelas aproximações numéricas. Entretanto, esta propriedade permanece válida somente quando consideramos integrações exatas. O uso de quadraturas numéricas, aliado às características geométricas do problema, pode causar a perda desta qualidade

Através de algumas simulações, vamos examinar o comportamento de alguns dos métodos apresentados até aqui para a discretização de tal problema, com relação a esta propriedade.

Nas figuras a seguir, podemos observar gráficos da massa em função do número de passos no tempo. As Figuras 3.1 e 3.2 se referem aos dados do Cenário 2, definido na Subseção 2.6.2, respectivamente para os métodos Euler-*Streamline Diffusion* e **PGDT**, enquanto as Figuras 3.3 e 3.4 dizem respeito aos dados do Cenário 3, definido na Subseção 2.6.3, para os mesmos métodos.

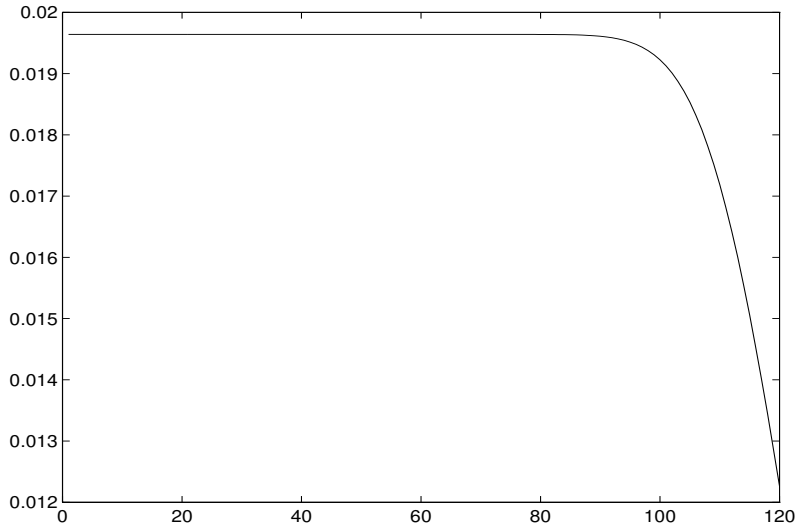

Figura 3.1: Massa - Método Euler-*Streamline Diffusion* para o Cenário 2.
Eixo horizontal: passos no tempo, eixo vertical: massa.

Observamos pelas Figuras 3.1 e 3.2 que tanto Euler-*Streamline Diffusion* quanto **PGDT** obtiveram bom comportamento em relação à conservação da massa. O decaimento ocorrido após 100 iterações no tempo se explica pelo fato da solução ter atingido o bordo de saída do domínio computacional, ou seja, a região considerada para a simulação numérica. O método de Euler-Galerkin com funções bolha também apresentou bom comportamento, embora não esteja ilustrado aqui por repetir os resultados alcançados com o método Euler-*Streamline Diffusion*, conforme as observações do Capítulo 2.

Já na Tabela 3.1, encontramos a relação entre a massa inicial (mi) da solução, calculada no tempo $t = 0$, e a massa final (mf), calculada no tempo em que ocorre o maior desvio com respeito à massa inicial, para os dados do Cenário 1, definido na Subseção 2.6.1.

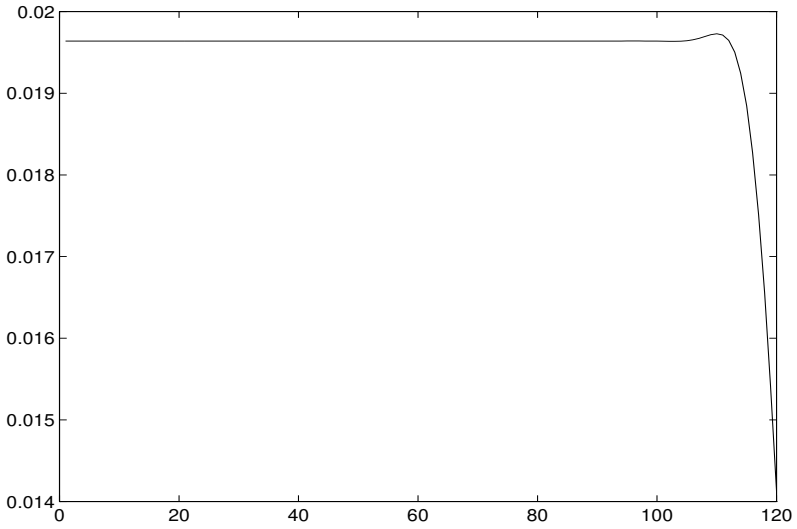

Figura 3.2: Massa - Método **PGDT** para o Cenário 2.
Eixo horizontal: passos no tempo, eixo vertical: massa.

Podemos observar que a performance dos métodos é bem parecida, exceto no caso em que utilizamos interpolação linear no método Semi-Lagrangiano em que ocorreu uma perda brusca de massa, mesmo sendo o domínio referente ao Cenário 1 um canal aberto, como no Cenário 2.

Método	**mf/mi**
PGDT ($\overline{c} = 0.5$)	1.00623
Euler-*Streamline Diffusion* ($\overline{c} = 0.5$)	1.00938
Semi-Lagrangiano (interpolação linear)	0.81537
Semi-Lagrangiano (interpolação quadrática)	1.01254

Tabela 3.1: Relação entre a Massa Inicial (mi) e a Massa Final (mf) - Cenário 1, $a = 0.25$.

Nas simulações seguintes, apresentadas nas Figuras 3.3 e 3.4, avaliaremos os métodos Euler-*Streamline Diffusion* e **PGDT** numa situação em que ocorre interação entre a solução do problema e as fronteiras internas do domínio.

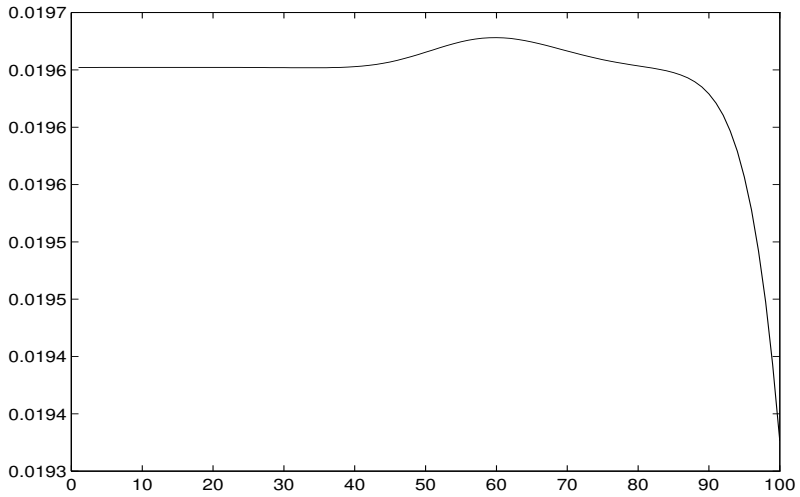

Figura 3.3: Massa - Método Euler-*Streamline Diffusion* para o Cenário 3.
Eixo horizontal: passos no tempo, eixo vertical: massa.

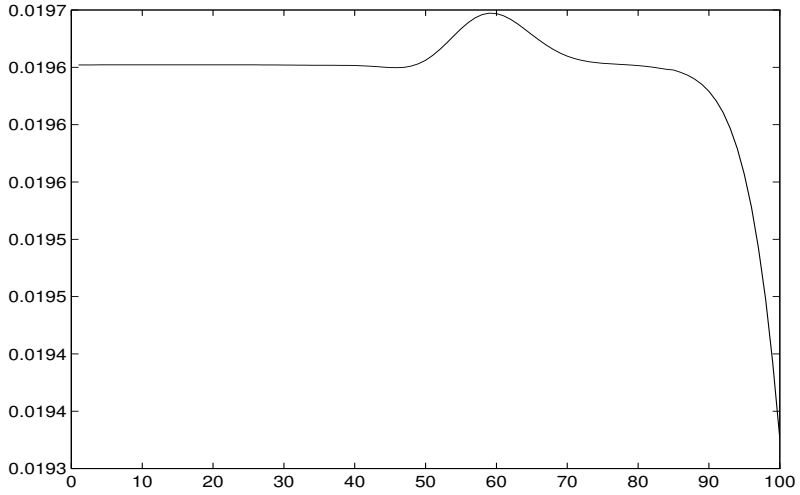

Figura 3.4: Massa - Método **PGDT** para o Cenário 3.
Eixo horizontal: passos no tempo, eixo vertical: massa.

Podemos observar claramente que, quando a mancha de petróleo interage com as fronteiras da ilha, a conservação da massa é significantemente violada por ambos os métodos, tendo sido pior para o método **PGDT**. Tais figuras mostram um crescimento da massa durante a passagem da mancha pelo obstáculo, com um posterior decaimento até atingir os níveis

anteriores. Ao final de 100 iterações, novamente o decaimento se deve ao fato da mancha estar saindo do domínio considerado.

Tendo em vista a equação (1.27), lembrando que $f \equiv 0$,

$$\frac{\partial}{\partial t}\int_{\tilde{\Omega}} u(x,t)dx = -\int_{\partial\tilde{\Omega}_+} u(x,t)\vec{\beta}\cdot\vec{\eta}ds$$

e o campo de velocidades

$$\vec{\beta} = \left(\, 1 - \frac{2x^2r^2}{(x^2+y^2)^2} + \frac{r^2}{(x^2+y^2)}\ ,\ -\frac{2xyr^2}{(x^2+y^2)^2}\right),$$

construído de forma a tangenciar o obstáculo presente no domínio, concluímos que

$$\int_{\partial\tilde{\Omega}_+} u(x,t)\vec{\beta}(x)\cdot\vec{\eta}ds = 0.$$

Então, pela construção das fronteiras da ilha, a propriedade

$$\frac{\partial}{\partial t}\int_{\Omega} u(x,t)dx = 0$$

continua válida, ou seja, a massa deveria se manter constante também durante a passagem da solução pelo obstáculo.

Continuamos investigando as possíveis causas desta variação indesejada da massa, assim como alternativas para contornar esta dificuladade. Nas simulações seguintes, desejamos verificar o efeito de refinamentos da malha, na região em torno da ilha, na conservação da massa. Para isso, na Tabela 3.2 apresentamos o quociente mf/mi, somente para o método **PGDT** com os dados do Cenário 3, com algumas malhas. Nesta tabela, indicamos h_{min} definido por

$$h_{min} = \min\{\ h_K\ /\ K \in T_h\ \},$$

isto é, o menor diâmetro dos elementos triangulares K, como já havíamos feito no capítulo anterior.

h_{min}	**mf/mi**
0.089	1.012359
0.044	1.005142
0.033	1.004236
0.021	1.003729
0.010	1.002976

Tabela 3.2: Comportamento da Massa em Relação ao Refinamento da Malha - **PGDT**.

Analisando tal tabela, podemos observar que a diminuição do parâmetro h_{min}, melhora a relação mf/mi, mas isto ocorre muito vagarosamente e certamente causa um sensível aumento do trabalho computacional. Novamente, gostaríamos de ressaltar que nosso objetivo é a eleboração de estratégias eficientes, sem a demanda de malhas extremamente finas. Por isso, cotinuamos as investigações em busca das causas e soluções de tais variações indesejadas da massa.

Realizando novas simulações com o mesmo método, procuramos descobrir se o tipo de geometria do problema tem influência em tais resultados, ou seja, na conservação da massa. Para isso, observamos que a passagem da mancha de óleo pelo obstáculo se divide em duas fases: na primeira ocorre um afunilamento do canal, que em uma segunda etapa volta a se alargar. Sendo assim, apresentamos as simulações seguintes com canais que se estreitam e se alargam, respectivamente.

As Tabelas $3.4 - 3.7$ apresentam os dados para tais simulações, enquanto as Figuras $3.5 - 3.8$ mostram os respectivos domínios. A relação entre (mi) e (mf), para cada tipo de canal utilizado, pode ser encontrada na Tabela 3.3.

Tipo de Canal	**mf/mi**
Canal Convergente - Cenário 4	1.008295
Canal Convergente Simétrico - Cenário 5	1.008738
Canal Divergente - Cenário 6	0.998902
Canal Divergente Simétrico - Cenário 7	0.998856

Tabela 3.3: Comportamento da Massa em Relação à Geometria do Canal - **PGDT** .

Cenário 4: Canal Convergente

Parâmetro	**Valor**
c	1×10^{-3}
$\vec{\beta}(x,t)$	$(1,0)$
f	0.0
g	0.0
k (Passo no tempo)	5×10^{-2}
$u_0(x,y)$	$0.1exp(-16((x+0.5)^2+y^2))$
Elementos (Primeira ordem)	4496
Nós	2327
h_{min}	0.037

Tabela 3.4: Dados Relativos ao Cenário 4.

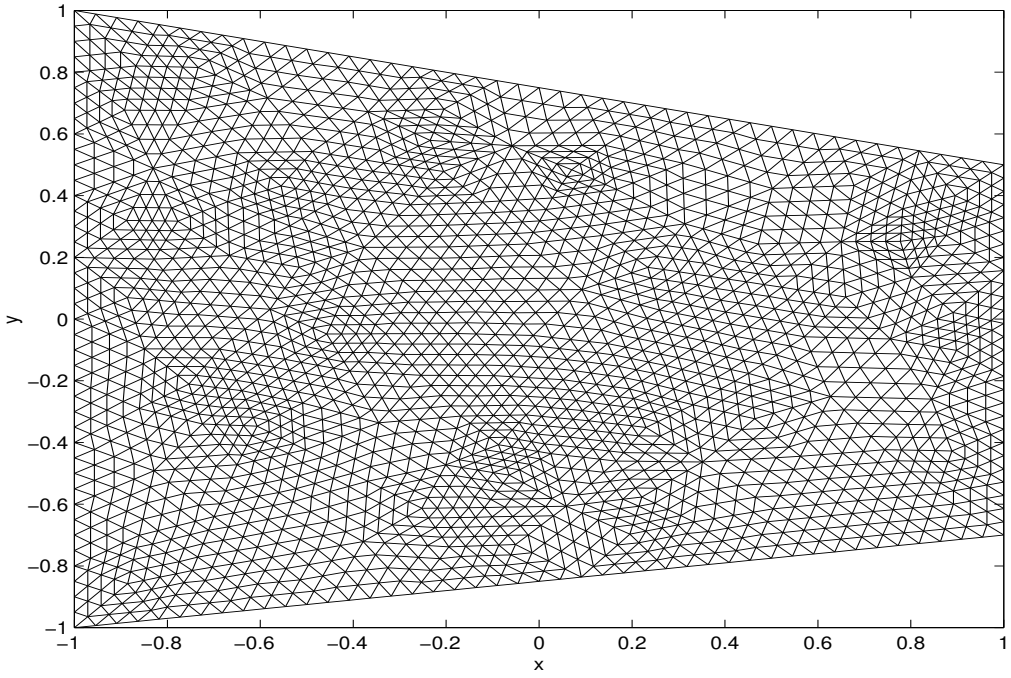

Figura 3.5: Domínio e Malha de Elementos Finitos para o Cenário 4.

Cenário 5: Canal Convergente Simétrico

Parâmetro	**Valor**
c	1×10^{-3}
$\vec{\beta}(x,t)$	$(1,0)$
f	0.0
g	0.0
k (Passo no tempo)	5×10^{-2}
$u_0(x,y)$	$0.1exp(-16((x+0.5)^2+y^2))$
Elementos (Primeira ordem)	4208
Nós	2179
h_{min}	0.037

Tabela 3.5: Dados Relativos ao Cenário 5.

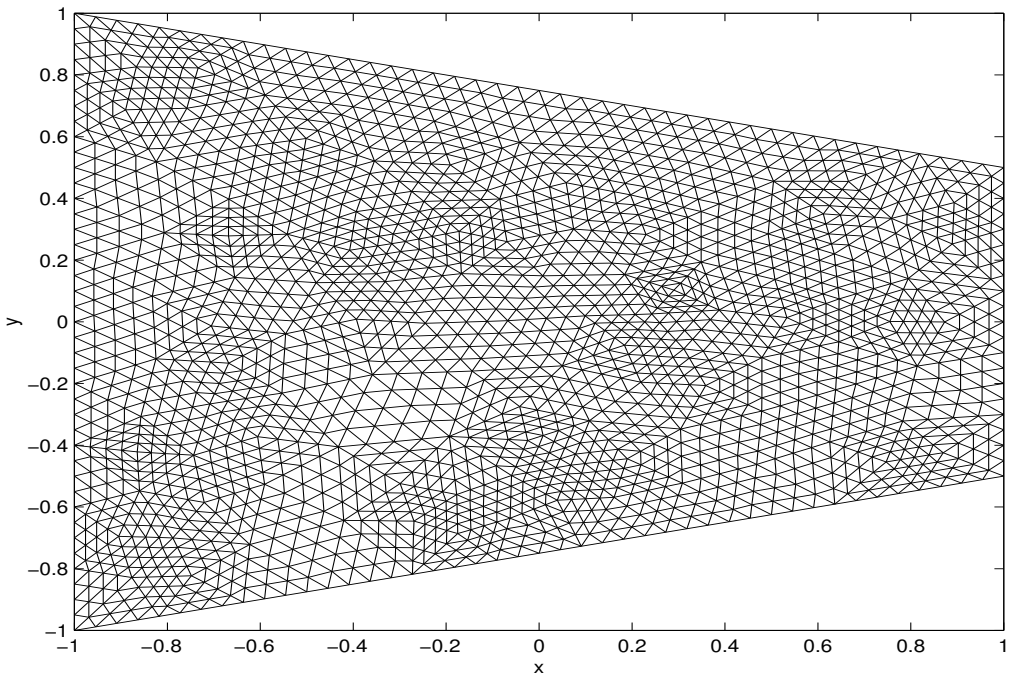

Figura 3.6: Domínio e Malha de Elementos Finitos para o Cenário 5.

Cenário 6: Canal Divergente

Parâmetro	**Valor**
c	1×10^{-3}
$\vec{\beta}(x,t)$	$(1,0)$
f	0.0
g	0.0
k (Passo no tempo)	5×10^{-2}
$u_0(x,y)$	$0.1exp(-16((x+0.5)^2+y^2))$
Elementos (Primeira ordem)	4240
Nós	2199
h_{min}	0.038

Tabela 3.6: Dados Relativos ao Cenário 6.

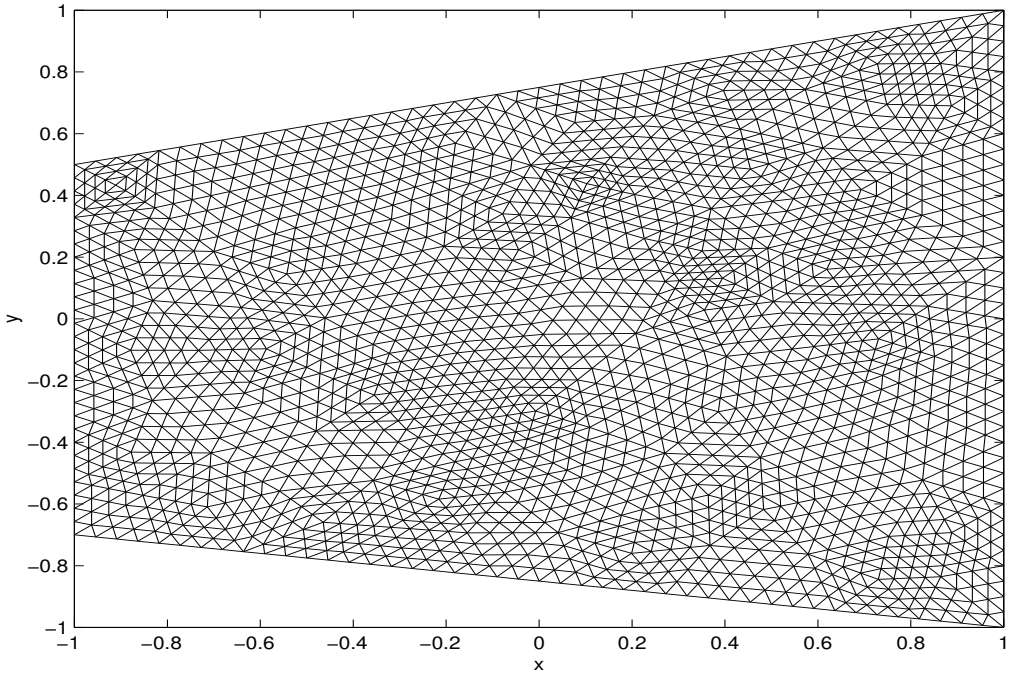

Figura 3.7: Domínio e Malha de Elementos Finitos para o Cenário 6.

Cenário 7: Canal Divergente Simétrico

Parâmetro	**Valor**
c	1×10^{-3}
$\vec{\beta}(x,t)$	$(1,0)$
f	0.0
g	0.0
k (Passo no tempo)	5×10^{-2}
$u_0(x,y)$	$0.1exp(-16((x+0.5)^2+y^2))$
Elementos (Primeira ordem)	3920
Nós	2035
h_{min}	0.039

Tabela 3.7: Dados Relativos ao Cenário 7.

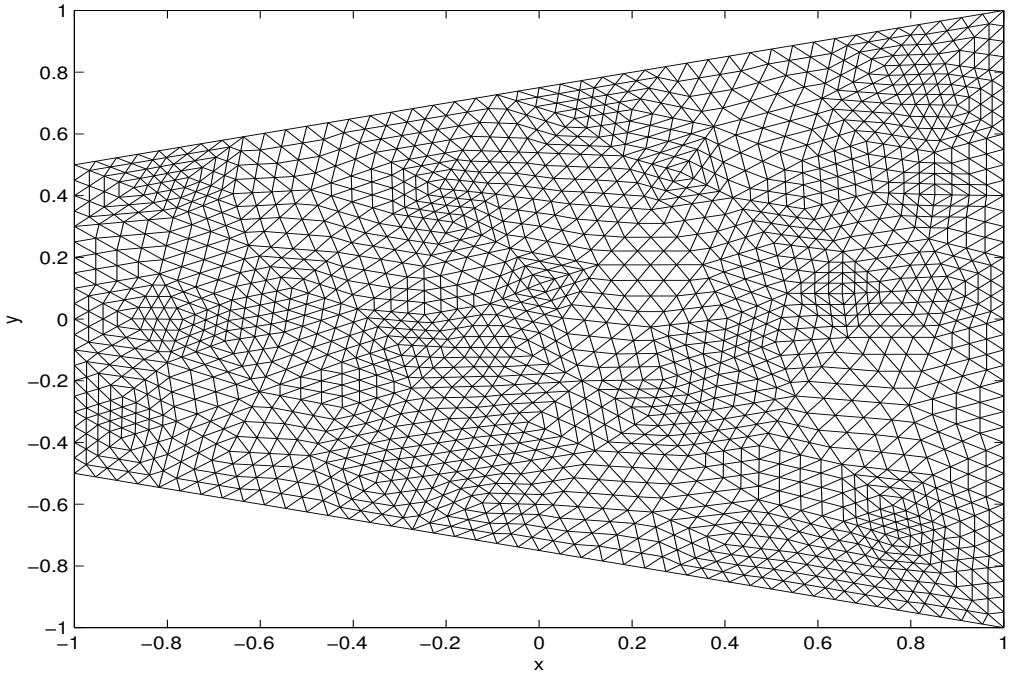

Figura 3.8: Domínio e Malha de Elementos Finitos para o Cenário 7.

Comparando os resultados obtidos nos Cenários $4-7$ (Tabela 3.3) com os resultados do Cenário 3 (Figura 3.4), notamos que a variação da massa apresenta comportamento análogo. Na parte convergente, ou de afunilamento dos canais, ocorre crescimento da massa e na região divergente ocorre decrescimento da mesma. Além disso, na mesma tabela podemos verificar que a simetria do canal não interfere neste comportamento.

Salientamos que os domínios dos Cenários $4-7$, apesar de serem relativamente simples, foram suficientes para nos remeter à violação da propriedade de balanço de massa da solução. Estas simulações nos levaram a acreditar que tal dificuldade está diretamente ligada à geometria do domínio em questão.

Tendo em vista nossos objetivos - simulações em cenários reais com geometrias complexas - necessitamos desenvolver uma forma de solucionar este problema.

Para melhorar a performance dos métodos com respeito ao balanço de massa, apresentamos a seguinte proposta de alteração e que constitui uma das contribuições deste trabalho: introduziremos um termo adicional ao lado direito da equação que define a formulação variacional do método e que irá forçar a solução aproximada a obedecer uma versão discretizada da equação de balanço (1.25).

Esta idéia original será detalhada a seguir para o método **PGDT**, em vista de todas as conclusões obtidas até agora sobre tal método.

Vamos tomar (1.25) com $\tilde{\Omega} = \Omega$ e utilizar as condições de contorno, isto é,

$$\frac{\partial}{\partial t}\int_{\Omega} u dx + \int_{\partial\Omega} u\vec{\beta}\cdot\vec{\eta} ds \; = \int_{\Omega} f dx. \tag{3.1}$$

Chamando de M_n a massa total calculada no tempo t_n:

$$M_n = \int_{\Omega} u(x, t_n) dx,$$

e discretizando (3.1) por diferenças finitas, obtemos a seguinte relação para prever a massa total em cada nível de tempo:

$$M_n = M_{n-1} - k_n \int_{\partial\Omega} u(x, t_n)\vec{\beta}(x)\cdot\eta(\vec{x}) ds + k_n \int_{\Omega} f(x, t_n) dx. \tag{3.2}$$

Por outro lado, seja a massa associada com a solução aproximada por elementos finitos dada por

$$\tilde{M_n} = \int_{\Omega} u^n(x) dx. \tag{3.3}$$

Então, para forçar a solução de elementos finitos a se aproximar da massa total verdadeira, introduzimos ao lado direito de (2.40) o seguinte termo:

$$-\epsilon(\tilde{M}_n - M_n)\max\{u^{n-1}(x), 0\}, \tag{3.4}$$

para $x \in \Omega$, ϵ uma constante estritamente positiva e com M_n e $\tilde{M}_n$ calculados respectivamente por (3.2) e (3.3).

Assim, este termo funciona como sorvedouro de massa se $\tilde{M}_n > M_n$ ou fonte se $\tilde{M}_n < M_n$. Quando $\tilde{M}_n = M_n$, o termo não altera a solução.

Obtemos então o seguinte problema para aproximar a solução:

$(\mathcal{V}_h)$ encontrar $u_h^n \in V_h^{0n},\ n = 1, 2, \cdots N$, tal que para todo $v \in V_h^{0n}$ valha

$$\langle \frac{\partial u_h^n}{\partial t} + \vec{\beta}\cdot\nabla u_h^n, v + \delta(\frac{\partial v}{\partial t} + \vec{\beta}\cdot\nabla v)\rangle^n + \langle 3c(u_-^{n-1})^2\nabla u_h^n, \nabla v\rangle^n -$$

$$- \langle div(3c(u_-^{n-1})^2\nabla u_h^n), \delta(\frac{\partial v}{\partial t} + \vec{\beta}\cdot\nabla v)\rangle^n + \langle\langle u_{h+}^n, v_+\rangle\rangle^{n-1} \ =$$

$$= \ \langle f - \epsilon(\tilde{M}_n - M_n)\max\{u_-^{n-1}, 0\}, v + \delta(\frac{\partial v}{\partial t} + \vec{\beta}\cdot\nabla v)\rangle^n + \langle\langle u_{h-}^{n-1}, v_+\rangle\rangle^{n-1},$$

com o parâmetro δ calculado como em (2.33).

Como veremos a seguir, esta adaptação, que convencionamos chamar de Petrov-Galerkin Descontínuo no Tempo com Controle de Massa, ou **PGDT-CM** para simplificar, melhorou significantemente os resultados do método **PGDT**, inclusive em simulações no Cenário 3.

Na Tabela 3.8, relacionamos o quociente mf/mi para simulações realizadas com o método **PGDT-CM**, usando $\epsilon = 5 \times 10^{-3}$ como parâmetro para o controle da massa, nos cenários apresentados até aqui. Os resultados de tal tabela podem ser comparados com os das Tabelas 3.1 e 3.3 para concluirmos que o método **PGDT-CM** apresentou melhores resultados em todos os cenários nos quais realizamos simulações.

Tipo de Canal	mf/mi
Canal Aberto - Cenário 1	1.000057
Canal Aberto - Cenário 2	1.000021
Canal com Ilha - Cenário 3	1.000063
Canal Convergente - Cenário 4	1.000072
Canal Convergente Simétrico - Cenário 5	1.000080
Canal Divergente - Cenário 6	0.999985
Canal Divergente Simétrico - Cenário 7	0.999943

Tabela 3.8: Comportamento da Massa em Relação à Geometria do Canal - **PGDT-CM**.

Para ilustrar, na Figura 3.9 encontramos o gráfico do comportamento da massa ao longo do tempo, para os dados do Cenário 3. Enquanto os métodos Euler-*Streamline Diffusion* e **PGDT** não conservam a massa durante a interação com a ilha (Figuras 3.3 e 3.4, respectivamente), o método **PGDT-CM** obedece tal propriedade.

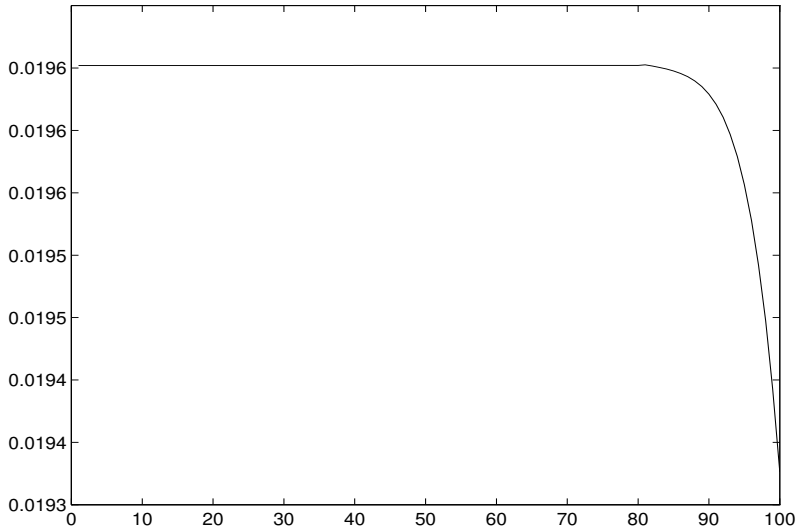

Figura 3.9: **PGDT-CM** - Cenário 3 - $\epsilon = 5 \times 10^{-3}$.
Eixo horizontal: passos no tempo, eixo vertical: massa.

Além de apresentar os melhores resultados nos testes efetuados até aqui, o método **PGDT-CM** ainda mostrou máximos muito próximos aos valores máximos exatos das soluções, em comparação com os demais métodos, para simulações realizadas no Cenário 1. Isso pode ser visto na Tabela 3.9 e evidencia que a incorporação deste mecanismo de balanço não alterou a boa qualidade das aproximações numéricas de tal método.

Método	**t = 0.5**	**t = 1**	**t = 1.5**	**t = 2**	**t = 2.5**	**t = 3**
solução exata	0.2184	0.1984	0.1842	0.1733	0.1646	0.1574
Euler-*Streamline Diffusion*	0.2064	0.1748	0.1554	0.1492	0.1289	0.1135
PGDT	0.2163	0.1945	0.1835	0.1744	0.1623	0.1558
PGDT-CM ($\epsilon = 5 \times 10^{-3}$)	0.2181	0.1988	0.1839	0.1739	0.1641	0.1570

Tabela 3.9: Máximos das soluções - $a = 0.25$.

Finalmente, na Tabela 3.10 verificamos o comportamento do método **PGDT-CM** com relação ao refinamento da malha do Cenário 3 e podemos observar a diminuição do desvio da massa, devido a tal tipo de refinamento, resultado esperado e bastante importante na determinação da qualidade da técnica de discretização utilizada.

h_{min}	**mf/mi**
0.089	1.000063
0.044	1.000041
0.033	1.000032
0.021	1.000022
0.010	1.000009

Tabela 3.10: Massa em Função do Refinamento da Malha - **PGDT-CM**.

Concluímos, então, que a estratégia que desenvolvemos para forçar o controle da massa funcionou bem em todos os casos que estudamos. Assim, nas simulações que realizaremos no próximo capítulo, em domínios com geometrias bem mais complexas e que, portanto, tendem a apresentar dificuldades maiores, aplicaremos tal técnica como forma de obter soluções mais confiáveis. Antes disso, proporemos uma maneira de diminuir o custo computacional de tal método.

3.2 Uma Estratégia de Localização e o Aumento da Eficiência do Método PGDT-CM

Nesta seção, estamos interessados em examinar o custo computacional, medido em tempo de processamento, de alguns dos métodos apresentados até agora.

Mais precisamente, proporemos uma modificação na forma com que os algoritmos **PGDT** e **PGDT-CM** foram implementados, levando em consideração o suporte compacto da solução do problema de espalhamento, para tornar tais métodos mais competitivos neste item.

Iniciaremos relembrando os resultados obtidos nas Tabelas 2.9 e 2.13, onde apresentamos o tempo médio de execução de cada método, necessário para a resolução de um nível de tempo. Tais resultados nos indicaram que o método **PGDT**, apesar de apresentar as melhores aproximações numéricas dentre os métodos que analisamos, não apresentou bom comportamento, tendo atingido tempo de CPU três vezes maior que o tempo gasto pelo método mais rápido: Euler-*Streamline Diffusion*.

Nestas tabelas, pudemos verificar a estreita relação entre o tempo de CPU e a dimensão do sistema linear resultante da discretização. Por isso, o método **PGDT** demandou mais tempo de processamento, principalmente, devido à dimensão de seu sistema linear, a cada nível de tempo.

Gostaríamos também de observar que a correção da massa que apresentamos na seção anterior, método **PGDT-CM**, pouco contribuiu para o aumento do tempo de processamento, elevando-o de 6.1 s para 6.2 s, para os dados do Cenário 3, e de 10.2 s para 10.4 s, para os dados do Cenário 1.

A estratégia original que desenvolvemos visa atacar este ponto crucial: a ordem do sistema linear. Levando em consideração o suporte compacto da solução, tal estratégia nos permite acompanhá-la ao longo do tempo, resolvendo o problema apenas numa parte do domínio computacional, isto é, num subdomínio que contenha o suporte da solução num determinado nível de tempo e no tempo subseqüente.

Com isto, em cada iteração no tempo o sistema linear resultante apresenta sensível diminuição de sua ordem e, conseqüentemente, o tempo de execução do algoritmo decresce.

A idéia geométrica do processo consiste em, conhecida a solução num determinado instante, prever a possível localização na iteração seguinte e resolver o sistema linear apenas num conjunto contendo os elementos nos quais as soluções nestes dois tempos subseqüentes não se anulem.

Para descrevê-la com mais detalhes, consideremos uma dada malha de elementos finitos descrevendo a região de interesse. Nesta malha, definimos:

Nel - número de elementos da malha;

Nver - número de vértices de cada elemento;

maximo - valor máximo atingido pela solução u_h^n;

xmax - vetor que contém a maior coordenada x de cada um dos elementos da malha;

ymax - vetor que contém a maior coordenada y de cada um dos elementos da malha;

xmin - vetor que contém a menor coordenada x de cada um dos elementos da malha;

ymin - vetor que contém a menor coordenada y de cada um dos elementos da malha;

lista - conjunto de índices dos elementos que pertencem ao subdomínio que contém o suporte da solução u_h^n;

listanew - conjunto de índices dos elementos que pertencem ao subdomínio que conterá o suporte da solução u_h^{n+1};

(x_{ij}, y_{ij}) - coordenadas do i-ésimo vértice do j-ésimo elemento.

Com estas definições, podemos colocar o algoritmo, que passamos a chamar de **PGDT-CM-SUB**, da seguinte forma (no Apêndice A podemos encontrar a listagem completa):

Inicialização: Montagem dos vetores **xmax**, **ymax**, **xmin** e **ymin**.

Esta fase é realizada uma única vez, antes do início das iterações no tempo, enquanto os **Passos** seguintes devem ser realizados em cada faixa de tempo. Seu principal objetivo é descrever de maneira grosseira os elementos da malha, considerando-os como se fossem retângulos. Esta primeira aproximação dos triângulos garante que o tempo de busca deste algoritmo não se torne impraticável. Para a montagem destes vetores, percorremos todos os elementos da malha e analisamos seus vértices.

Passo 1: Cálculo do valor máximo (**maximo**) atingido pela solução u_h^n.

Novamente percorremos todos os elementos da malha para o cálculo da variável **maximo**, que tem por objetivo servir como parâmetro para a obtenção dos elementos que contêm o suporte da solução.

Passo 2: Determinação do subdomínio que contém o suporte da solução u_h^n.

Conhecido o maior valor atingido por u_h^n, verificamos quais nós da malha apresentam, pelo menos, um milésimo do valor máximo e marcamos os elementos que os contêm, pela inclusão de seus índices ao conjunto **lista**.

Tal passo pode ser descrito da seguinte forma:

Para $j = 1, 2, \cdots, Nel$

Para $i = 1, \cdots, Nver$

Se $u_h^n(x_{ij}, y_{ij}) \geq 10^{-3}$**maximo**, colocar o índice j no conjunto **lista**.

Passo 3: Previsão do subdomínio que conterá o suporte da solução u_h^{n+1}.

A idéia básica deste passo é separar os dois efeitos do problema de espalhamento. Primeiro a advecção e em seguida a difusão.

Inicialmente, para cada cada vértice (x_{ij}, y_{ij}) da malha, encontramos o seu respectivo ponto $(\tilde{x}_{ij}, \tilde{y}_{ij})$, origem de (x_{ij}, y_{ij}), considerando apenas o efeito de transporte (lembrando que k_n é o passo no tempo):

Para $j = 1, 2, \cdots, Nel$

Para $i = 1, \cdots, Nver$

$\tilde{x}_{ij} = x_{ij} - \vec{\beta}_1(x_{ij}, y_{ij})k_n$

$\tilde{y}_{ij} = y_{ij} - \vec{\beta}_2(x_{ij}, y_{ij})k_n.$

Em seguida, todos os pontos origem devem ser localizados em seus respectivos elementos $\tilde{k}$, incorporando o índice j ao conjunto **listanew** se $\tilde{k}$ já pertencer ao conjunto **lista**.

Esta é a parte crucial do algoritmo e pode levar muito tempo, se não for executada com cuidado, podendo comprometer a desejada diminuição do tempo de CPU que buscamos. A localização do ponto $(\tilde{x}_{ij}, \tilde{y}_{ij})$ num elemento da malha foi detalhada na Subseção 2.5.1, para as interpolações necessárias ao método Semi-Lagrangiano.

Aqui só relembraremos que a idéia para isto consiste em, numa etapa inicial, considerarmos uma aproximação grosseira de cada elemento triangular k que compõe a malha, pela utilização de retângulos dados por

$$[xmin(k), xmax(k)] \times [ymin(k), ymax(k)].$$

Assim, encontramos elementos candidatos a pertencer ao conjunto **listanew**.

Tais candidatos serão examinados com todo o cuidado necessário, por meio de transformações lineares que levam cada triângulo candidato em um triângulo-padrão com vértices em $(0,0),(1,0)$ e $(0,1)$.

De forma resumida, podemos escrever:

Para $j = 1, 2, \cdots, Nel$

Para $i = 1, \cdots, Nver$

Localizar $(\tilde{x}_{ij}, \tilde{y}_{ij})$ no elemento $\tilde{k}$

Se o elemento $\tilde{k}$ pertencer ao conjunto **lista**, colocar j no conjunto **listanew**.

Na Figura 3.10, encontramos uma ilustração de malha de elementos finitos com o respectivo subdomínio, obtido até este ponto do algoritmo, contendo o suporte da solução.

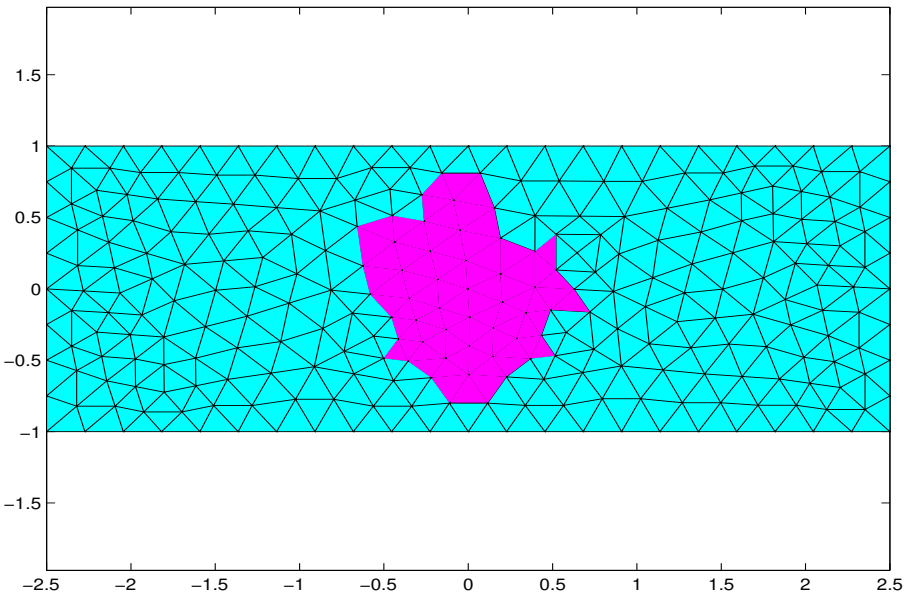

Figura 3.10: Subdomínio Contendo o Suporte da Solução.

Finalizando este passo, incluímos o efeito difusivo do operador, aumentando o subdomínio no qual resolveremos o problema. Isto deve ser feito porque o mecanismo de previsão, que utilizamos na primeira parte deste passo, leva em consideração apenas a advecção, desconsiderando o aumento do suporte da solução em decorrência da difusão. Então, introduzimos

uma camada de novos elementos no subdomínio, contendo os elementos da malha que fazem fronteira com os triângulos deste conjunto.

Isto pode ser visto na Figura 3.11, onde as partes demarcadas correspondem ao subdomínio onde o problema será efetivamente resolvido, num determinado nível de tempo.

Em nossas experiências, percebemos que a inclusão de apenas uma camada é suficiente para simular o efeito difusivo no caso do problema não-linear de espalhamento que temos tratado o tempo todo. Entretanto, mais camadas de triângulos podem ser necessárias se, por exemplo, estivermos resolvendo um problema com difusão linear, por causa do aumento maior do suporte da solução.

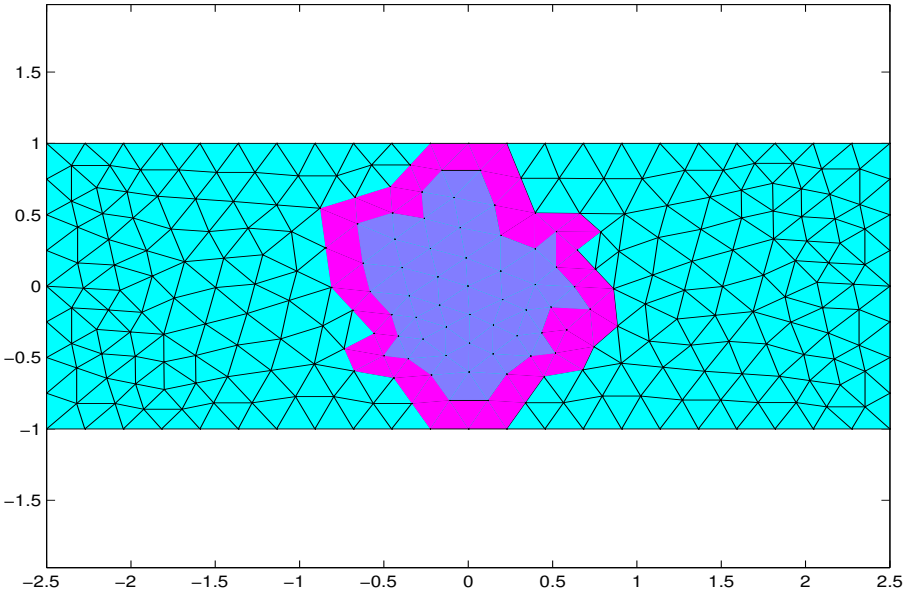

Figura 3.11: Subdomínio Contendo o Suporte da Solução para 2 Níveis Subseqüentes de Tempo.

Passo 4: Montagem do sistema linear.

Utilizando os conjuntos **lista** e **listanew**, montamos o sistema linear resultante do método **PGDT-CM**, ou seja, montamos o sistema somente com as variáveis pertencentes aos elementos cujos índices estão presentes nestes conjuntos, considerando as demais variáveis nulas.

Na Tabela 3.11, podemos observar os tempos de CPU do método **PGDT-CM-SUB** em simulações realizadas nos cenários definidos até aqui. Se a compararmos com as Tabelas 2.9 e 2.13, vemos claramente a diminuição do tempo médio de execução, que passou a ser muito próximo do tempo fornecido pelo método Euler-*Streamline Diffusion*, mostrando quão efetiva pode ser esta estratégia.

Tipo de Canal	Tempo de CPU	Nós na Rede
Canal Aberto - Cenário 1	4.1 s	3057
Canal Aberto - Cenário 2	4.2 s	3057
Canal com Ilha - Cenário 3	2.5 s	1840
Canal Convergente - Cenário 4	3.2 s	2327
Canal Convergente Simétrico - Cenário 5	2.9 s	2179
Canal Divergente - Cenário 6	3.0 s	2199
Canal Divergente Simétrico - Cenário 7	2.8 s	2035

Tabela 3.11: Tempos de CPU - **PGDT-CM-SUB**.

Com mais esta modificação, o método **PGDT-CM-SUB** passa a aliar confiabilidade e eficiência. Desta forma, segundo nosso ponto de vista, temos em mãos um algoritmo apropriado para realizar novas simulações numéricas, em situações mais complexas. Antes, porém, examinaremos novamente o modelo e introduziremos mais um termo à equação não-linear de advecção-difusão.

3.3 Modelo de Espalhamento com Degradação

Finalizando este capítulo, visando tornar o problema mais próximo da realidade, vamos modificar o modelo de espalhamento pela inclusão de um termo não-linear de degradação à equação (2.1). Assim, o problema fica reescrito como

$$\frac{\partial u(x,t)}{\partial t} - c\Delta((u(x,t))^3) + \vec{\beta}(x,t)\cdot\nabla u(x,t) + c_2\sigma(u)u(x,t) = f(x,t) \ \ , \ \ \Omega\times I \tag{3.5}$$

$$u(x,t) = g(x) \ , \ x \in \ \partial\Omega_- \ , \ \ t\in I \tag{3.6}$$

$$\frac{\partial u(x,t)}{\partial \vec{\eta}} = 0 \ , \ \ x \in \ \partial\Omega_+ \ , \ \ t\in I \tag{3.7}$$

$$u(x,0) = u_0(x) \ , \ x\in\Omega, \tag{3.8}$$

onde c_2 é uma constante positiva e a função $\sigma(u)$ corresponde à área exposta da mancha.

Salientamos que a escolha de c_2 constante é uma simplificação. Na verdade, este é um parâmetro que depende da composição do óleo.

A superfície definida por $u(x,t)$, tem sua área dada por

$$\sigma(u) = \int_\Omega \sqrt{1 + (u_x)^2 + (u_y)^2} dxdy. \tag{3.9}$$

A incorporação de tal termo ao problema, permite que parte da mancha de petróleo se desintegre por evaporação. Desta forma, ao aproximarmos a solução deste problema, o método de discretização deve ter sua quantidade de difusão artificial controlada, para que o aumento do suporte da solução não seja demasiado a ponto de causar uma degradação maior que a esperada, levando à desintegração precoce da mancha de petróleo. Mas felizmente, este ponto já foi resolvido neste capítulo e no anterior, pelo uso do método **PGDT-CM-SUB**.

Para seu tratamento numérico, este termo será linearizado como $\sigma(u^{n-1})u^n$, isto é por atraso no tempo, como já havíamos feito anteriormente com a não-linearidade do termo difusivo.

Trabalhando da mesma forma como fizemos antes, obtemos o seguinte problema para aproximar a solução de $(3.5) - (3.8)$:

$(\mathcal{V}_h)$ encontrar $u_h^n \in V_h^{0n},\ n = 1, 2, \cdots N$, tal que para todo $v \in V_h^{0n}$ valha

$$\langle \frac{\partial u_h^n}{\partial t} + \vec{\beta}\cdot\nabla u_h^n + c_2\sigma(u_-^{n-1})u_h^n, v + \delta(\frac{\partial v}{\partial t} + \vec{\beta}\cdot\nabla v)\rangle^n + \langle 3c(u_-^{n-1})^2\nabla u_h^n, \nabla v\rangle^n -$$

$$- \langle div(3c(u_-^{n-1})^2\nabla u_h^n), \delta(\frac{\partial v}{\partial t} + \vec{\beta}\cdot\nabla v)\rangle^n + \langle\langle u_{h+}^n, v_+\rangle\rangle^{n-1} =$$

$$= \langle f - \epsilon(\tilde{M}_n - M_n)\max\{u_-^{n-1}, 0\}, v + \delta(\frac{\partial v}{\partial t} + \vec{\beta}\cdot\nabla v)\rangle^n + \langle\langle u_{h-}^{n-1}, v_+\rangle\rangle^{n-1},$$

onde $\delta = \overline{c}h$ se $3c(u_-^{n-1})^2 < h,\ \overline{c} > 0$ suficientemente pequena e $\delta = 0$ se $3c(u_-^{n-1})^2 \geq h$. As definições de M_n e $\tilde{M}_n$ podem ser encontradas na Seção 3.1.

Como podemos notar, este problema variacional já inclui a previsão e correção da massa da solução, entretanto, neste caso a equação que define seu balanço é dada por

$$M_n(1 + k_n c_2 \sigma(u^{n-1})) = M_{n-1} - k_n \int_{\partial\Omega} u(x, t_n)\vec{\beta}(x) \cdot \eta(\vec{x})ds + k_n \int_{\Omega} f(x, t_n)dx,$$

levando a um esperado decaimento da massa, mesmo no caso de simulações sem interações com as fronteiras e com fonte f identicamente nula (veja a Figura 3.13).

A Figura 3.12 apresenta o resultado de uma simulação do método **PGDT-CM-SUB**, aplicado ao problema $(3.5)-(3.8)$ com os dados do Cenário 2, mas com campo de velocidades constante $\vec{\beta} = (1, 0)$, $\epsilon = 5 \times 10^{-3}$ e $c_2 = 1 \times 10^{-1}$ como coeficiente de degradação. Nesta tabela são mostradas as curvas de nível da solução para $t = 2.0,\ 3.0,\ 5.0$ e 6.0, respectivamente. Já na Figura 3.13, encontramos o gráfico da massa da solução em função do número de passos no tempo, para a mesma simulação.

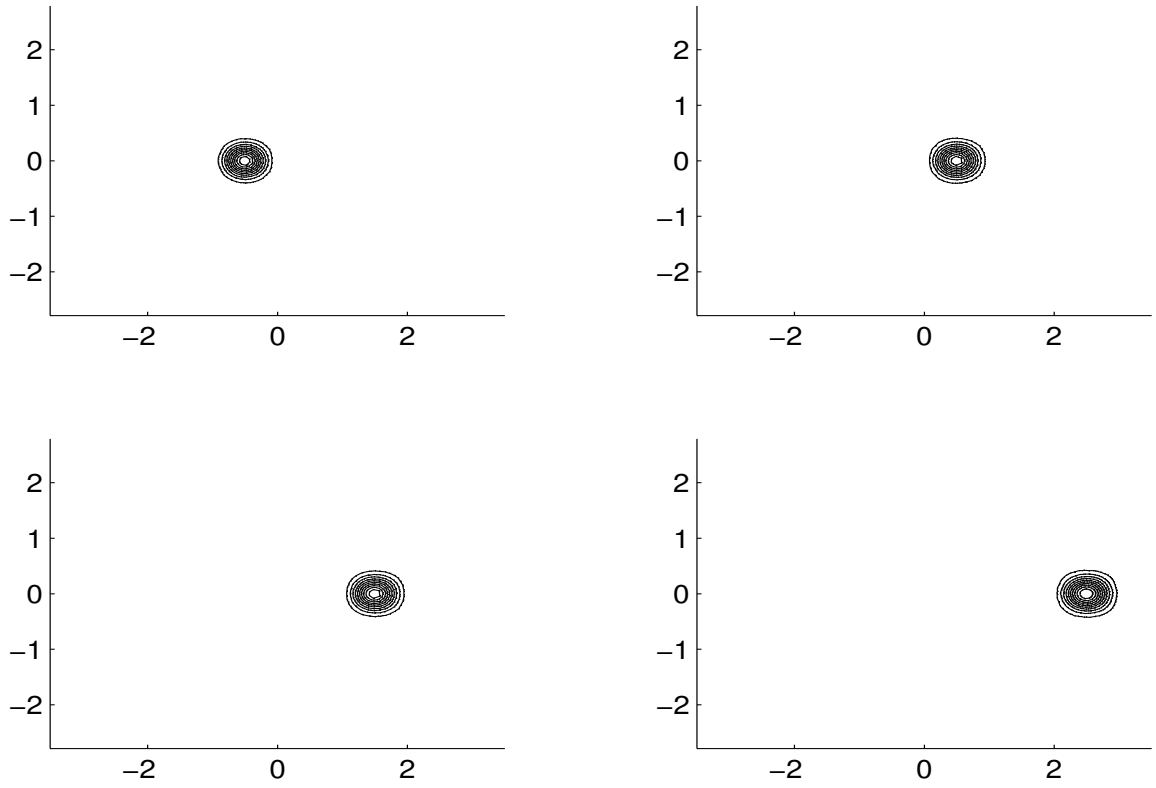

Figura 3.12: Modelo com Degradação - Método **PGDT-CM-SUB** - $t = 2.0,\ 3.0,\ 5.0$ e 6.0.

Observando a Figura 3.12, temos a impressão de que nada está acontecendo. Mas, ao analisarmos o gráfico da Figura 3.13 verificamos o decaimento rápido da massa. Isto é

um sinal de que a constante de degradação $c_2 = 1 \times 10^{-1}$ desintegrou o petróleo muito rapidamente.

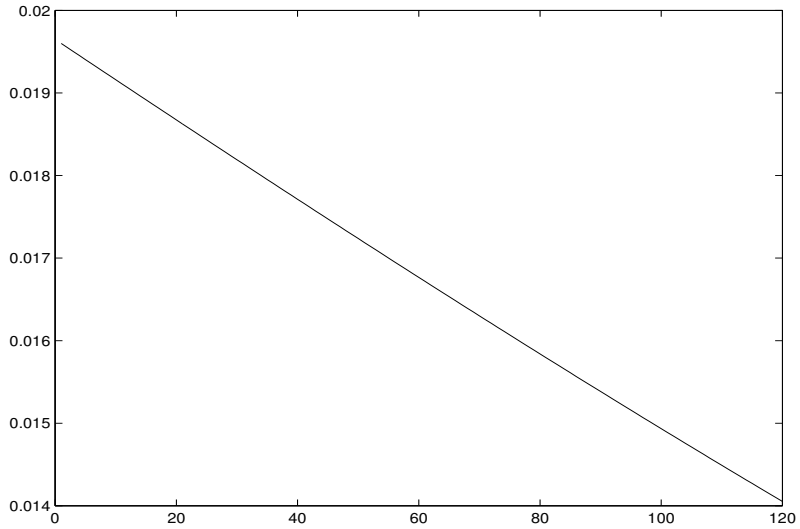

Figura 3.13: Massa no Modelo com Degradação - **PGDT-CM-SUB**.
Eixo horizontal: passos no tempo, eixo vertical: massa.

Neste ponto, acreditamos termos concluído o nosso objetivo principal: isolar o estágio referente à obtenção de soluções numéricas para as equações que descrevem o modelo de espalhamento de manchas de petróleo, propondo um método confiável e rápido **PGDT-CM-SUB**. Sendo assim, estamos prontos para caminhar um pouco mais na direção de problemas de derramamento de petróleo com dados mais realistas. Esta abordagem será introduzida no próximo capítulo.

Capítulo 4

Uma Proposta de Procedimento para a Previsão do Espalhamento de Manchas de Petróleo

Neste capítulo, pretendemos resolver problemas de espalhamento de manchas de petróleo em situações mais gerais, descrevendo os derramamentos de óleo de forma mais completa. Para isso, incorporamos aos programas computacionais o cálculo do escoamento incompressível utilizando as equações de Navier-Stokes. Desta forma, o escoamento da água deixa de ser dado diretamente, passando a ser uma das incógnitas do problema, que agora transformou-se num sistema contendo as equações de Navier-Stokes juntamente com a equação não-linear de advecção-difusão que governa o espalhamento, propriamente.

Para realizar o cálculo do escoamento, nos deparamos com duas possibilidades: utilizar uma implementação própria ou recorrer a softwares prontos. Só como exemplo, podemos citar o *FEATFLOW* (www.featflow.de) desenvolvido e oferecido gratuitamente por um grupo de pesquisadores da Universidade de Dortmund, na Alemanha, além do *FEMLAB* (www.femlab.com), um toolbox do Matlab.

Preferimos recorrer a uma implementação própria, em vista das dificuldades de acoplar tais softwares aos nossos programas, devido à utilização de linguagens e conceitos de programação diferentes.

Desta forma, iniciamos este capítulo apresentando um simulador para a resolução de tais equações e em seguida, nos dedicamos à validação de seu código pela aplicação em problemas clássicos da literatura - escoamento entre placas paralelas e cavidade quadrada.

Entretanto, gostaríamos de ressaltar que tal simulador não tem o objetivo de resolver rigorosamente os escoamentos. Para isso, novas pesquisas serão necessárias. Com esta implementação temos apenas o intuito de produzir previsões do campo de velocidades para a realização de simulações numéricas de derramamentos de petróleo, na última seção deste capítulo, tendo como cenário a Baía de Guanabara, no Estado do Rio de Janeiro.

4.1 Equações de Navier-Stokes

Iniciaremos introduzindo as equações de Navier-Stokes, modelo matemático apropriado para descrever escoamentos bidimensionais de fluidos viscosos e incompressíveis.

$$\rho\frac{\partial\vec{u}}{\partial t} \; - \; \mu\Delta\vec{u} \; + \; \rho(\; \vec{u}\cdot\nabla\;)\vec{u} \; + \; \nabla p \; = \; \rho\vec{f} \;\; , \;\; \Omega\times I \tag{4.1}$$

$$div\;\vec{u} \; = \; 0 \;\; , \;\; \Omega\times I. \tag{4.2}$$

Aqui, Ω é um domínio limitado do $I\!R^2$ onde se dá o fluxo e $\vec{x} = (x, y)$ é um ponto neste domínio; $I = (0, T] \subset I\!R$ é um intervalo de tempo, $p(\vec{x}, t)$ é a pressão, ρ representa a densidade do fluido e μ a sua viscosidade. A força externa, por unidade de massa, se escreve como

$$\vec{f}(\vec{x}) \; = \; \begin{pmatrix} f_x(\vec{x}) \\ f_y(\vec{x}) \end{pmatrix}$$

e o campo de velocidades procurado é dado por

$$\vec{u}(\vec{x}, t) \; = \; \begin{pmatrix} u(\vec{x}, t) \\ v(\vec{x}, t) \end{pmatrix}.$$

Para completar o problema, condições iniciais $u = u_0(x, y)$ e $v = v_0(x, y)$ satisfazendo (4.2) e condições de contorno devem ser impostas.

Utilizando a notação,

$$(\; \vec{u}\cdot\nabla\;)\vec{u} \; = \; \left(\begin{pmatrix} u \\ v \end{pmatrix} \cdot \begin{pmatrix} \partial/\partial x \\ \partial/\partial y \end{pmatrix} \right) \begin{pmatrix} u \\ v \end{pmatrix},$$

reescrevemos $(4.1) - (4.2)$ como

$$\rho\frac{\partial u}{\partial t} \; - \; \mu\;\Delta u \; + \; \rho\;\vec{u}\cdot\nabla u \; + \; \frac{\partial p}{\partial x} \; = \; \rho f_x \;\; , \;\; \Omega\times I \tag{4.3}$$

$$\rho\frac{\partial v}{\partial t} \; - \; \mu\,\Delta v \; + \; \rho\,\vec{u}\cdot\nabla v \; + \; \frac{\partial p}{\partial y} \; = \; \rho f_y \;\; , \;\; \Omega\times I \tag{4.4}$$

$$\frac{\partial u}{\partial x} \; + \; \frac{\partial v}{\partial y} \; = \; 0 \;\; , \;\; \Omega\times I. \tag{4.5}$$

Para que possamos adimensionalizar as equações $(4.3)-(4.5)$ conjuntamente com o problema não-linear de advecção-difusão, resecrevemos a equação de espalhamento como

$$\frac{\partial h}{\partial t} - c\Delta(h^3) + \vec{u}\cdot\nabla h + c_2\sigma(h)h = f_1, \tag{4.6}$$

utilizando a variável h para designar a altura da mancha, de óleo de modo a evitar confusão com o campo de velocidades $\vec{u}$.

4.2 Adimensionalização

Várias vezes queremos reproduzir experimentos físicos de grandes dimensões em um laboratório. Para isso, devemos combinar os diversos parâmetros que caracterizam tais fenômenos de forma adequada, dando origem a quantidades adimensionais e que nos permitam relacionar fenômenos de grandes e pequenas escalas. Como pretendemos simular problemas de advecção-difusão em conjunto com o cálculo dos fluxo, devemos adimensionalizar as equações que descrevem os dois problemas da mesma forma.

Iniciaremos com a equação (4.6), introduzindo as seguintes constantes como quantidades de referência: H é altura característica, V a velocidade e L o comprimento. E as seguintes variáveis adimensionais:

$$h^* = \frac{h}{H}, \quad t^* = \frac{Vt}{L}, \quad \vec{u^*} = \frac{\vec{u}}{V}, \quad x^* = \frac{x}{L}, \quad y^* = \frac{y}{L}.$$

Levando essas variáveis a (4.6), chegamos a

$$\frac{HV}{L}\frac{\partial h^*}{\partial t^*} - c\frac{H^3}{L^2}\Delta^*(h^{*3}) + \frac{HV}{L}\vec{u^*}\cdot\nabla^* h^* + c_2 H\sigma^* h^* = f_1,$$

ou, após algumas manipulações a

$$\frac{\partial h^*}{\partial t^*} - c\frac{H^2}{LV}\Delta^*(h^{*3}) + \vec{u^*}\cdot\nabla^* h^* + c_2\frac{L}{V}\sigma^* h^* = \frac{L}{HV}f_1,$$

em que os operadores ∇^* e Δ^* são tomados com respeito a x^* e y^*.

Procederemos da mesma maneira com a equação de Navier-Stokes na forma (4.1), reescrevendo-a como

$$\frac{\partial \vec{u}}{\partial t} \; - \; \frac{\mu}{\rho}\,\Delta\vec{u} \; + \; (\;\vec{u}\cdot\nabla\;)\vec{u} \; + \frac{1}{\rho}\,\nabla p \; = \vec{f}. \tag{4.7}$$

Definimos

$$\vec{x^*} = \frac{\vec{x}}{L}, \quad t^* = \frac{Vt}{L}, \quad \vec{u^*} = \frac{\vec{u}}{V}, \quad p^* = \frac{p}{V^2}, \quad \vec{f^*} = \frac{L}{V^2}\vec{f}.$$

Estas novas variáveis conduzem (4.7) à sua forma adimensional:

$$\frac{\partial \vec{u^*}}{\partial t^*} \; - \; \frac{\mu}{\rho VL}\,\Delta^*\vec{u^*} \; + \; (\;\vec{u^*}\cdot\nabla^*\;)\vec{u^*} \; + \; \nabla^* p^* \; = \; \frac{L}{V^2}\vec{f},$$

ou

$$\frac{\partial \vec{u^*}}{\partial t^*} \; - \; \frac{1}{Re}\,\Delta^*\vec{u^*} \; + \; (\;\vec{u^*}\cdot\nabla^*\;)\vec{u^*} \; + \; \nabla^* p^* \; = \; \vec{f^*},$$

sendo Re o Número de Reynolds dado por

$$Re = \frac{\rho VL}{\mu},$$

que é o parâmetro que controla a relação entre a difusão e a advecção neste problema.

A força adimensional $\vec{f^*}$ também pode ser escrita como

$$\vec{f^*} = \left(\frac{1}{Fr}\right)^2 \frac{\vec{f}}{\parallel \vec{f} \parallel} = \frac{L}{V^2}\vec{f},$$

onde Fr, conhecido como Número de Froude, é dado por

$$Fr = \frac{V}{\sqrt{L \parallel \vec{f} \parallel}}.$$

4.3 Linearização e Discretização

Para discretizar as equações de Navier-Stokes, a forma apresentada em $(4.1) - (4.2)$ seria adequada desde que desejássemos impor condições de contorno de Dirichlet na velocidade e uma condição sobre a pressão, num único ponto, isto é, um nível de referência para a pressão. Mas nosso objetivo é impor também condições do tipo *stress* nulo, que dependem da formulação variacional do problema e implicam em deixar o fluido livre de forças na saída do fluxo.

Para isso, será necessário tomar as equações de Navier-Stokes na seguinte forma equivalente a $(4.1) - (4.2)$:

$$\rho\frac{\partial\vec{u}}{\partial t} \; - \; div(\sigma_1) \; + \; \rho(\; \vec{u}\cdot\nabla\;)\vec{u} \; = \; \vec{f}, \tag{4.8}$$

na qual $\sigma_1 = -p(\vec{x},t)I + 2\mu D$ é o tensor de *stress*, sendo D dado por

$$D \; = \; \frac{1}{2}(L + L^t) \; = \; \frac{1}{2}\begin{pmatrix} 2\frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} & 2\frac{\partial v}{\partial y} \end{pmatrix}, \tag{4.9}$$

sendo $L = \nabla\vec{u}$ o gradiente do vetor $\vec{u}$, definido como

$$\nabla\vec{u} = \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{pmatrix}.$$

Utilizando a definição de σ_1, podemos concluir que

$$-div(\sigma_1) = \nabla p - \mu\Delta\vec{u}. \tag{4.10}$$

Para obtermos a formulação variacional do problema, multiplicamos (4.8) pelas funções-teste $\vec{\varphi} = \begin{pmatrix}\varphi_1 \\ \varphi_2\end{pmatrix}$, $\varphi_1, \varphi_2 \in H_0^1$, que definiremos a seguir, integramos sobre Ω e utilizamos o teorema de divergência, de forma que

$$\int_\Omega div(\sigma_1)\vec{\varphi}dx = \int_{\partial\Omega}\sigma_1\cdot\vec{\eta}\vec{\varphi}ds - \int_\Omega \sigma_1\cdot\nabla\vec{\varphi}dx. \tag{4.11}$$

Neste ponto, impomos condições de contorno do tipo *stress* nulo, isto é, $\sigma_1 \cdot \vec{\eta} = 0$ ou condições de Dirichlet em cada bordo do domínio, de forma que a integral sobre $\partial\Omega$ da equação acima se anule.

Desta forma, usando (4.10) temos que

$$\int_\Omega (\nabla p - \mu\Delta\vec{u})\vec{\varphi}dx = -\int_\Omega div(\sigma_1)\vec{\varphi}dx = \int_\Omega \sigma_1 \cdot \nabla\vec{\varphi}dx, \tag{4.12}$$

tomando o cuidado de interpretar "$\cdot$" como o produto interno entre os vetores-linha correspondentes de cada matriz presente no integrando de (4.12).

Podemos notar pela definição de σ_1 que impor este tipo de condição significa estabelecer um relação entre a pressão e a velocidade no bordo do domínio, não sendo portanto necessário impor outra condição sobre a pressão. Além disso, esta é uma condição de contorno natural.

Por outro lado,

$$\sigma_1 \cdot \nabla\vec{\varphi} = \begin{pmatrix} -p & 0 \\ 0 & -p \end{pmatrix} \cdot \begin{pmatrix} \frac{\partial\varphi_1}{\partial x} & \frac{\partial\varphi_1}{\partial y} \\ \frac{\partial\varphi_2}{\partial x} & \frac{\partial\varphi_2}{\partial y} \end{pmatrix} + \mu \begin{pmatrix} 2\frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} & 2\frac{\partial v}{\partial y} \end{pmatrix} \cdot \begin{pmatrix} \frac{\partial\varphi_1}{\partial x} & \frac{\partial\varphi_1}{\partial y} \\ \frac{\partial\varphi_2}{\partial x} & \frac{\partial\varphi_2}{\partial y} \end{pmatrix} =$$

$$= \begin{pmatrix} -p\frac{\partial\varphi_1}{\partial x} \\ -p\frac{\partial\varphi_2}{\partial y} \end{pmatrix} + \mu \begin{pmatrix} 2\frac{\partial u}{\partial x}\frac{\partial\varphi_1}{\partial x} + (\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x})\frac{\partial\varphi_1}{\partial y} \\ (\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x})\frac{\partial\varphi_2}{\partial x} + 2\frac{\partial v}{\partial y}\frac{\partial\varphi_2}{\partial y} \end{pmatrix} =$$

$$= \begin{pmatrix} -p\frac{\partial\varphi_1}{\partial x} \\ -p\frac{\partial\varphi_2}{\partial y} \end{pmatrix} + \mu \begin{pmatrix} \frac{\partial u}{\partial x}\frac{\partial\varphi_1}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial\varphi_1}{\partial y} \\ \frac{\partial v}{\partial x}\frac{\partial\varphi_2}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial\varphi_2}{\partial y} \end{pmatrix} + \mu \begin{pmatrix} \frac{\partial u}{\partial x}\frac{\partial\varphi_1}{\partial x} + \frac{\partial v}{\partial x}\frac{\partial\varphi_1}{\partial y} \\ \frac{\partial u}{\partial y}\frac{\partial\varphi_2}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial\varphi_2}{\partial y} \end{pmatrix} =$$

$$= -p \begin{pmatrix} \frac{\partial\varphi_1}{\partial x} \\ \frac{\partial\varphi_2}{\partial y} \end{pmatrix} + \mu \begin{pmatrix} \nabla u \cdot \nabla\varphi_1 \\ \nabla v \cdot \nabla\varphi_2 \end{pmatrix} + \mu \begin{pmatrix} \frac{\partial u}{\partial x}\frac{\partial\varphi_1}{\partial x} + \frac{\partial v}{\partial x}\frac{\partial\varphi_1}{\partial y} \\ \frac{\partial u}{\partial y}\frac{\partial\varphi_2}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial\varphi_2}{\partial y} \end{pmatrix} \tag{4.13}$$

Como já fizemos anteriormente, definimos $\Pi : \quad 0 = t_0 < t_1 < \cdots < t_N = T$ uma partição qualquer de $I = [0, T]$ com $I_n = (t_{n-1}, t_n)$ e $k_n = t_n - t_{n-1}$ o passo local no tempo, e utilizamos o método de Euler Regressivo para a discretização na variável temporal e ao mesmo tempo para a linearização do problema.

Como podemos notar, conforme a relação entre μ e ρ, a equação (4.1) pode assumir caráter "predominantemente hiperbólico" - Número de Reynolds grande - sendo necessário aplicar a elas métodos estabilizados de elementos finitos. Por isso, optamos pelo método de Galerkin com funções bolha, que foi apresentado no Capítulo 2, para a discretização espacial, porque tal método surgiu especificamente para equações deste tipo.

Desta forma, juntando (4.13) com (4.12), podemos enunciar o problema variacional discreto:

Encontrar u_h^n, $v_h^n \in V_h$ e $p_h^n \in M_h$, $n = 1, 2, \cdots, N$, satisfazendo

$$\rho \, \langle \frac{u_h^n - u_h^{n-1}}{k_n}, \varphi_h \rangle + \mu \, \langle \nabla u_h^n, \nabla \varphi_h \rangle + \rho \, \langle \vec{u_h}^{\,n-1} \cdot \nabla u_h^n, \varphi_h \rangle - \langle p_h^n, \frac{\partial \varphi_h}{\partial x} \rangle +$$

$$+ \mu \langle \frac{\partial u_h^n}{\partial x}, \frac{\partial \varphi_h}{\partial x} \rangle + \mu \langle \frac{\partial v_h^n}{\partial x}, \frac{\partial \varphi_h}{\partial y} \rangle = \langle f_x, \varphi_h \rangle, \quad \forall \, \varphi_h \in V_h,$$

$$\rho \, \langle \frac{v_h^n - v_h^{n-1}}{k_n}, \varphi_h \rangle + \mu \, \langle \nabla v_h^n, \nabla \varphi_h \rangle + \rho \, \langle \vec{u_h}^{\,n-1} \cdot \nabla v_h^n, \varphi_h \rangle - \langle p_h^n, \frac{\partial \varphi_h}{\partial y} \rangle +$$

$$+ \mu \langle \frac{\partial u_h^n}{\partial y}, \frac{\partial \varphi_h}{\partial x} \rangle + \mu \langle \frac{\partial v_h^n}{\partial y}, \frac{\partial \varphi_h}{\partial y} \rangle = \langle f_y, \varphi_h \rangle, \quad \forall \, \varphi_h \in V_h,$$

$$\langle \frac{\partial u_h^n}{\partial x} + \frac{\partial v_h^n}{\partial y}, w_h \rangle = 0, \quad \forall \, w_h \in M_h,$$

e as condições iniciais

$$\langle u_h^0, \varphi_h \rangle = \langle u_0, \varphi_h \rangle, \quad \forall \, \varphi_h \in V_h,$$

$$\langle v_h^0, \varphi_h \rangle = \langle v_0, \varphi_h \rangle, \quad \forall \, \varphi_h \in V_h.$$

Escolhemos os espaços da seguinte forma:

$$H_0^1 = \{v \in \mathcal{H}^1(\Omega) \ / \ v \mid_{\partial\Omega_1} = 0\},$$

$$V_h = \{ \ v \in H_0^1 \ / \ v \mid_K \in \mathcal{P}_2(K) \oplus \mathcal{B}(K), \ \forall \ K \in T_h \ \},$$

sendo $\partial\Omega$ particionado em $\partial\Omega = \partial\Omega_1 \ \cup \ \partial\Omega_2$, onde $\partial\Omega_1$ é a parte da fronteira onde são impostas condições de Dirichlet e $\partial\Omega_2$ a parte da fronteira onde são impostas condições de *stress* nulo. O subespaço $\mathcal{B}(K)$ é o espaço das funções bolha definidas sobre cada elemento da triangularização T_h e

$$M_h = \{ \ v \in \mathcal{H}^1(\Omega) \ / \ v \mid_K \in \mathcal{P}_0(K), \forall \ K \in T_h \ \},$$

isto é, espaço das funções polinomiais constantes em cada elemento.

Para a equação (4.6), utilizamos o método **PGDT-CM-SUB**, descrito no capítulo anterior.

4.4 Testes com o Simulador Proposto

A seguir, apresentamos algumas simulações numéricas com as equações de Navier-Stokes, resolvidas a partir de uma velocidade inicial $\vec{u}_0(\vec{x})$ e fonte $\vec{f}(\vec{x})$ nula, sendo impostas condições de contorno do tipo *stress* nulo nas saídas do fluxo, exceto para o problema de cavidade, e Dirichlet nos demais bordos do domínio coincidindo com a condição inicial $\vec{u}_0(\vec{x})$ em $\partial\Omega$.

Trata-se de problemas clássicos e com solução conhecida, sendo apresentados com o intuito de verificar a qualidade das aproximações obtidas, dando-nos a confiança necessária em nossa implementação do algoritmo para o cálculo do escoamento.

Em todas as figuras, os resultados são mostrados após o campo de velocidades $\vec{u}$ atingir o regime estacionário.

A primeira simulação se refere a um escoamento laminar bidimensional entre paredes paralelas, com Número de Reynolds $Re = 300$, sendo a velocidade inicial dada por

$$\vec{u}_0(\vec{x}) \ = \ \begin{pmatrix} 1 - y^2 \\ 0 \end{pmatrix}, \quad y \in [-1, 1].$$

Neste caso, a solução exata do problema apresenta a velocidade dada por $\vec{u}(\vec{x}) = \vec{u}_0(\vec{x})$, enquanto a pressão varia linearmente na direção x do escoamento, permanecendo constante na direção y. Na Figura 4.1 este comportamento da pressão pode ser observado claramente.

Para verificar o comportamento do erro, em relação ao refinamento da malha, apresentamos na Tabela 4.1 simulações com diferentes malhas. Esta tabela apresenta o erro absoluto, calculado utilizando-se a norma-infinito, entre as soluções exata e numérica. Lembramos que a definição do parâmetro de discretização da malha - h_{min} - pode ser encontrada na Seção 2.6.1.

Tais resultados mostram o decaimento do erro, com o refinamento da malha, evidenciando a boa qualidade da aproximação numérica, neste caso.

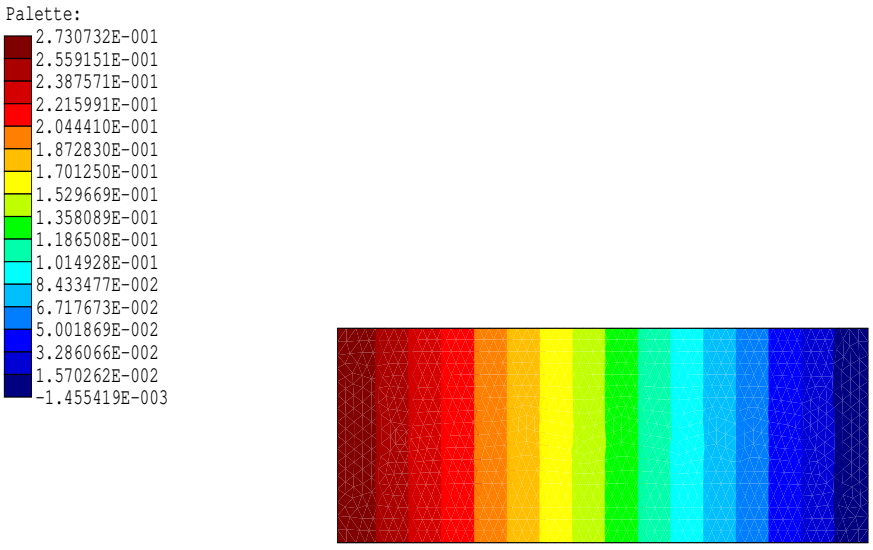

Figura 4.1: Escoamento entre Placas Paralelas (Pressão) - $Re = 300$.

h_{min}	**Erro Absoluto**
0.236	2.17×10^{-3}
0.085	1.25×10^{-4}
0.049	8.39×10^{-6}
0.021	6.53×10^{-8}

Tabela 4.1: Erro Absoluto - Exata X Numérica (norma-infinito).

Na Figura 4.2, apresentamos o campo de velocidades, no regime estacionário, para uma simulação de escoamento num canal contendo um obstáculo circular de raio $r = 0.25$ e com Número de Reynolds $Re = 300$. Assim como na simulação anterior, este também é um problema com solução conhecida dada por

$$\vec{u}(\vec{x}) = \begin{pmatrix} 1 - \frac{2x^2r^2}{(x^2+y^2)^2} + \frac{r^2}{(x^2+y^2)} \\ \frac{-2xyr^2}{(x^2+y^2)^2} \end{pmatrix}.$$

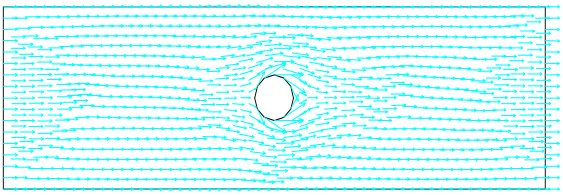

Figura 4.2: Escoamento num Canal com Obstáculo - $Re = 300$.

Na Tabela 4.2, simulações deste problema com diferentes malhas são apresentadas, com o intuito de verificar o comportamento do erro da aproximação numérica, em função do refinamento da malha. Neste problema também podemos observar que o erro absoluto entre as soluções exata e numérica diminui quando a malha é refinada, mostrando a boa qualidade das aproximações.

h_{min}	**Erro Absoluto**
0.297	3.63×10^{-3}
0.115	6.54×10^{-5}
0.061	1.92×10^{-7}
0.029	8.39×10^{-9}

Tabela 4.2: Erro Absoluto - Exata X Numérica (norma-infinito).

A seguir, apresentamos um problema de cavidade quadrada com tampa móvel. Este é um problema bastante importante para a validação de códigos computacionais porque nele estão presentes as principais dificuldades encontradas em soluções numéricas em mecânica dos fluidos, apesar de sua simplicidade geométrica. Neste caso, foram impostas condições de

Dirichlet para a velocidade, nulas nas paredes laterais e inferior, e $\vec{u} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ na tampa da cavidade. Utilizamos uma malha uniforme com 200 elementos triangulares e 441 pontos e $Re = 100$.

Os perfis de velocidade, através do centro geométrico da cavidade, podem ser vistos nas Figuras $4.3 - 4.4$, em comparação com os resultados obtidos por Ghia [30], que servem para validar nosso simulador aqui indicado pela sigla **GFB**, numa alusão à utilização do método de Galerkin com funções bolha. Cabe salientar que os resultados de Ghia foram obtidos com um método que combina diferenças finitas centrais e *upwind*. Neste caso, em que os termos convectivos possuem pouca expressão, os resultados obtidos já são bastante bons, mesmo numa malha grosseira como a que utilizamos.

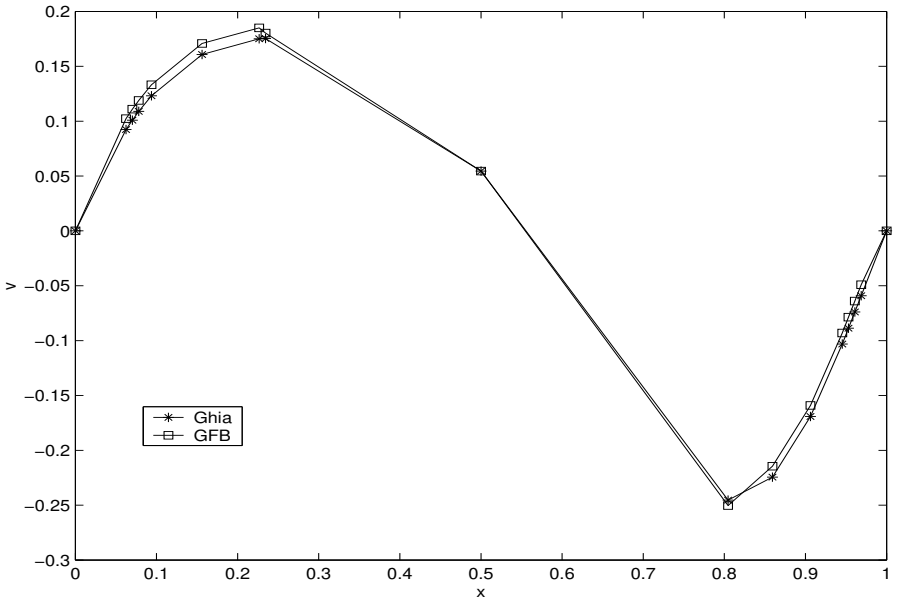

Figura 4.3: Perfil de Velocidade para o Problema de Cavidade - $Re = 100$.

A próxima simulação foi realizada com $Re = 400$. Neste caso, o cálculo do escoamento torna-se mais difícil sendo necessária a utilização de uma malha mais refinada, com 800 elementos triangulares e 1681 pontos.

Aqui, de acordo com o trabalho de Ghia [30], começam a aparecer nos cantos inferiores direito e esquerdo, vórtices mostrando as recirculações que ocorrem nestes locais. Além disso, o campo de velocidades deve girar numa região nas proximidades do centro geométrico do problema.

Para simulações com Re maiores, torna-se necessário um refinamento ainda maior da malha, em virtude das singularidades apresentadas nos cantos superiores direito e esquerdo

da cavidade, devido à discontinuidade nas condições de contorno impostas nestes extremos.

Os perfis de velocidade, neste caso, podem ser vistos nas Figuras $4.5 - 4.6$.

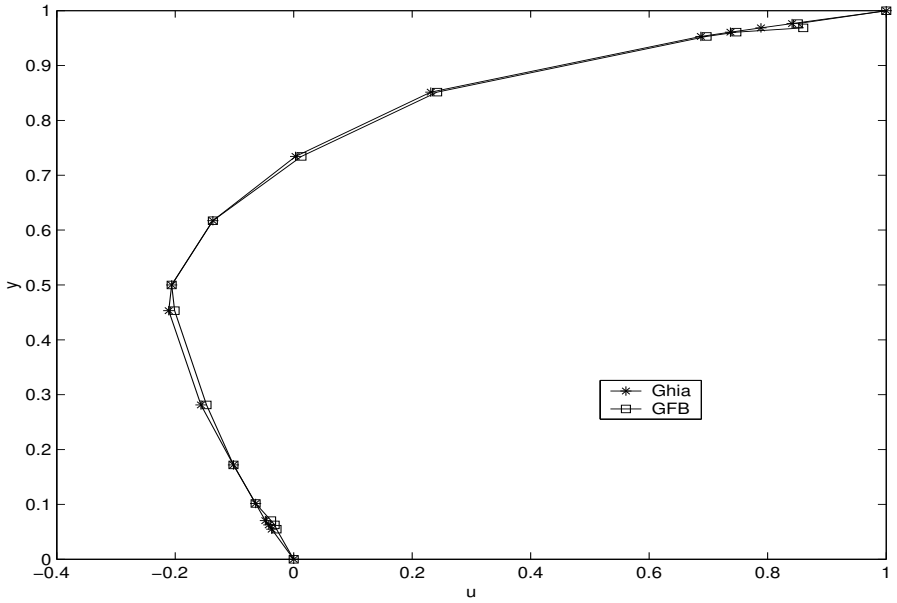

Figura 4.4: Perfil de Velocidade para o Problema de Cavidade - $Re = 100$.

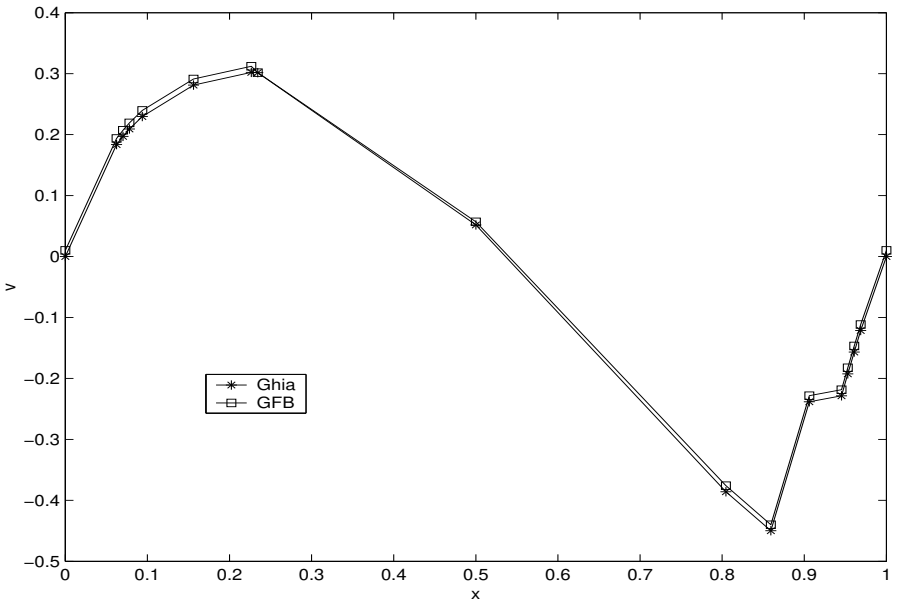

Figura 4.5: Perfil de Velocidade para o Problema de Cavidade - $Re = 400$.

Novamente, ocorreu uma boa concordância de resultados, conseguida pelo refinamento da malha utilizada, neste caso em que os termos convectivos aumentaram sua influência no problema.

Gostaríamos também de salientar que a equação para o divergente nulo (4.2), foi muito bem resolvida em todas as simulações realizadas, apresentando norma-infinito da ordem de 10^{-12}, sendo esta uma outra boa evidência da confiabilidade do algoritmo utilizado.

Desta forma, após examinar os resultados do problema de escoamento entre placas paralelas e da cavidade quadrada, nos sentimos satisfeitos, pelo menos para nossos objetivos atuais, com o simulador de fluxos incompressíveis que temos em mãos, passando a utilizá-lo na próxima seção.

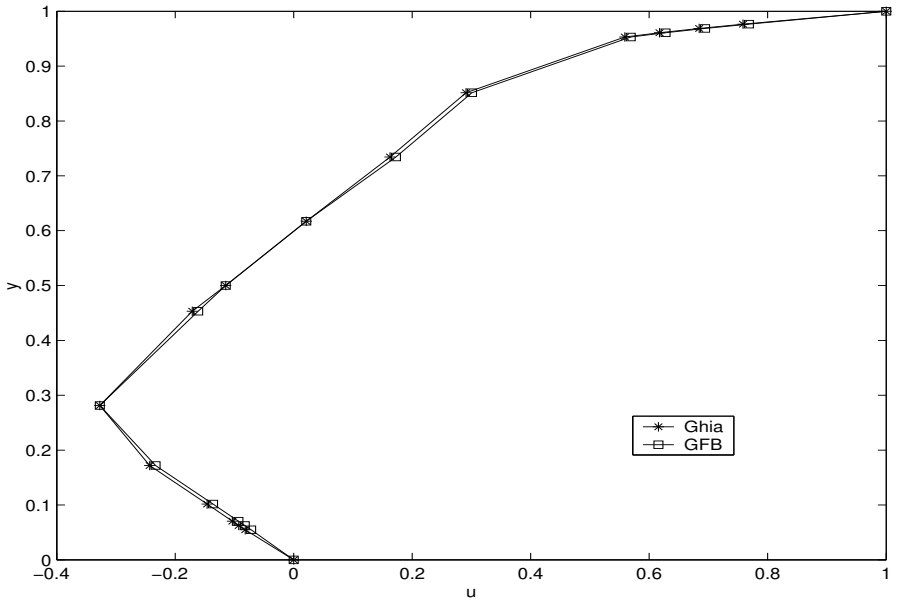

Figura 4.6: Perfil de Velocidade para o Problema de Cavidade - $Re = 400$.

4.5 Simulações de Derrames

Chegamos, finalmente, ao ponto em que realizaremos experiências computacionais com o objetivo de descrever derramamentos de óleo de forma mais completa, pela introdução do cálculo do campo de velocidades - via equações de Navier-Stokes - ao modelo de espalhamento tratado exaustivamente neste trabalho. Desta forma, o escoamento da água deixa de ser dado diretamente, passando a ser uma das incógnitas do problema, que agora transformou-se num sistema contendo as equações de Navier-Stokes $(4.1) - (4.2)$ juntamente com a equação não-linear de advecção-difusão (4.6).

Antes, porém, gostaríamos de ressaltar que, para o cálculo do escoamento em regiões costeiras ou lagos, o modelo mais indicado é representado pelas equações de águas-rasas (*shallow water equations*), que são uma variação das equações de Navier-Stokes e levam em consideração a profundidade destas regiões, podendo ser encontradas em [9], por exemplo. Entretanto, em vista das dificuldades adicionais, além da falta de dados sobre a profundidade da Baía de Guanabara, não faremos uso de tal modelo, deixando sua inclusão entre os possíveis trabalhos futuros.

Como já citamos acima, realizaremos simulações na Baía de Guanabara, onde têm ocorrido inúmeros acidentes deste tipo, principalmente no ano de 2000.

Na Figura 4.7 um mapa de tal região é apresentado. Neste mapa, destacamos a presença da Refinaria Duque de Caxias (Reduc), localizada no município carioca de mesmo nome. Tal refinaria tem capacidade de produção de 36 milhões de litros de derivados de petróleo por dia, os quais são bombeados para o Terminal da Ilha D'Água, pequena ilha que não aparece neste mapa e se localiza próxima à divisa entre as cidades de São João do Meriti e Rio de Janeiro.

Figura 4.7: Baía de Guanabara.

A transmissão do petróleo entre a Reduc e este terminal é feita em 14 dutos que possuem mais de 20 quilômetros de extensão, sendo estes dutos a maior fonte de vazamentos responsáveis pelos acidentes ocorridos ultimamente.

Na Figura 4.8, apresentamos o domínio computacional com sua respectiva malha de elementos finitos para a discretização das equações de Navier-Stokes. Tal região exclui a parte continental do mapa e inclui a Ilha de Paquetá, que também não aparece na Figura 4.7, mas pode ser um importante modificador do escoamento em tal baía.

Partindo de uma velocidade inicial na entrada do canal, podemos estabelecer o escoamento em toda a baía.

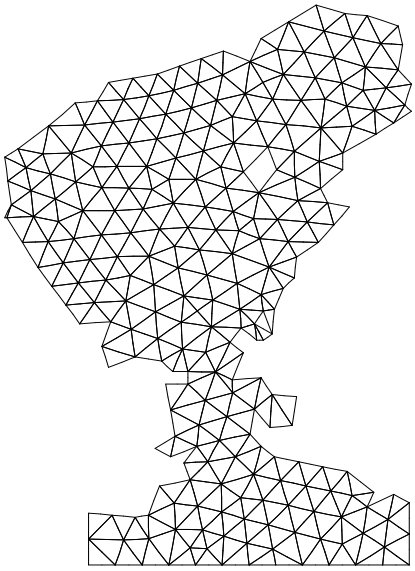

Figura 4.8: Domínio Computacional e Malha de Elementos Finitos - $L_2(T_h)$.

Condições de contorno nulas, para a velocidade, foram impostas nas fronteiras do tipo mar-terra. Para evitar a presença de singularidades na região de entrada da baía, cantos inferiores direito e esquerdo da Figura 4.8, foram impostas velocidades horizontais e com perfil parabólico, de forma a eliminar descontinuidades nas condições de contorno.

Os parâmetros para estas simulações, Cenário 8, podem ser encontrados na Tabela 4.3.

Nesta tabela, podemos observar que as malhas de elementos finitos são do tipo $L_2(T_h)$ para as equações de Navier-Stokes e $L_1(T_h)$ para o problema de advecção-difusão, conforme os espaços de aproximação definidos anteriormente. Assim, o campo de velocidades $\vec{u}$ foi interpolado para servir como dado para a equação de advecção-difusão.

Cenário 8: Baía de Guanabara

Parâmetro	Valor
c	1×10^{-3}
c_2	1×10^{-2}
f_1	0.0
g	0.0
k (Passo no tempo)	5×10^{-2}
Elementos (advecção-difusão)	1735
Nós (advecção-difusão)	978
Elementos (Navier-Stokes)	422
Nós (Navier-Stokes)	962

Tabela 4.3: Dados Relativos ao Cenário 8.

Na Figura 4.9, apresentamos o campo de velocidades $\vec{u}$ para o respectivo domínio, depois de atingir o estado estacionário.

Na primeira simulação, consideramos o maior causador dos acidentes ocorridos nesta baía, o rompimento dos oleodutos que transportam os derivados de petróleo entre a refinaria e o terminal. Em geral, este tipo de ocorrência só é percebida depois que uma grande quantidade de óleo já vazou, formando uma mancha de grandes proporções.

Na Figura 4.10 encontramos as curvas de nível da mancha, no instante de sua descoberta. Consideramos que, neste mesmo momento, o vazamento dos dutos foi localizado e reparado. Nas Figuras $4.11 - 4.13$, acompanhamos a evolução de tal mancha nos instantes seguintes.

De acordo com as características do escoamento, a mancha se desloca na direção de São João do Meriti, causando um grande acúmulo de óleo principalmente nas praias desta cidade além de Duque de Caxias, Magé e em parte da Ilha de Paquetá (Figura 4.7).

Uma conclusão importante é que, num acidente deste tipo, as praias das cidades do Rio de Janeiro e Niterói estariam livres da contaminação por óleo.

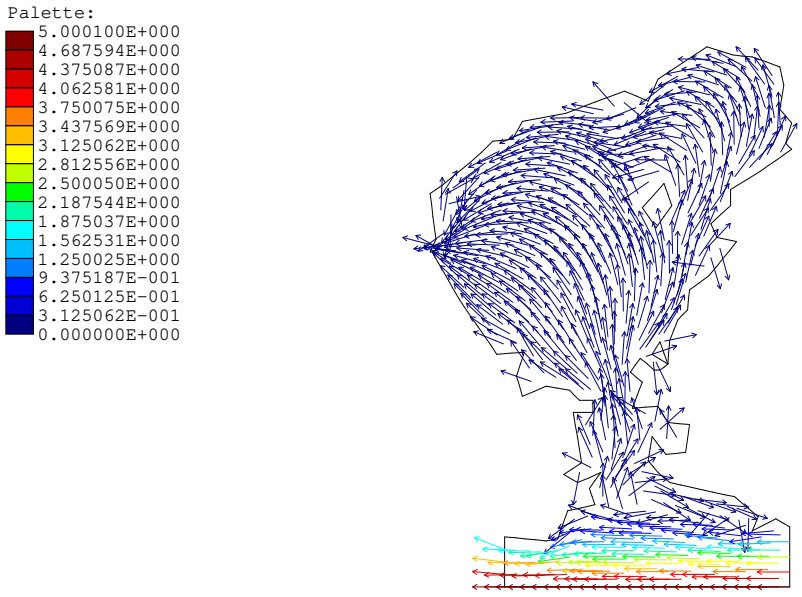

Figura 4.9: Campo de Velocidades na Baía.

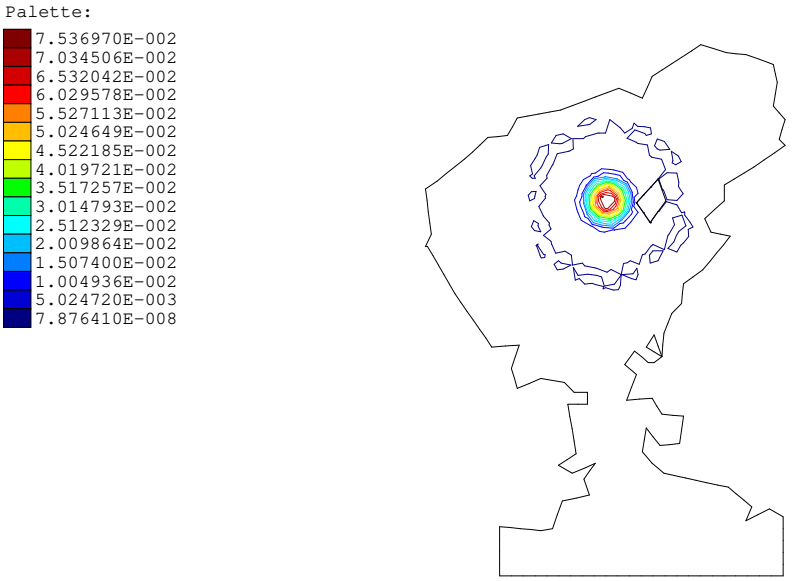

Figura 4.10: Problema de Advecção-Difusão - Condição inicial.

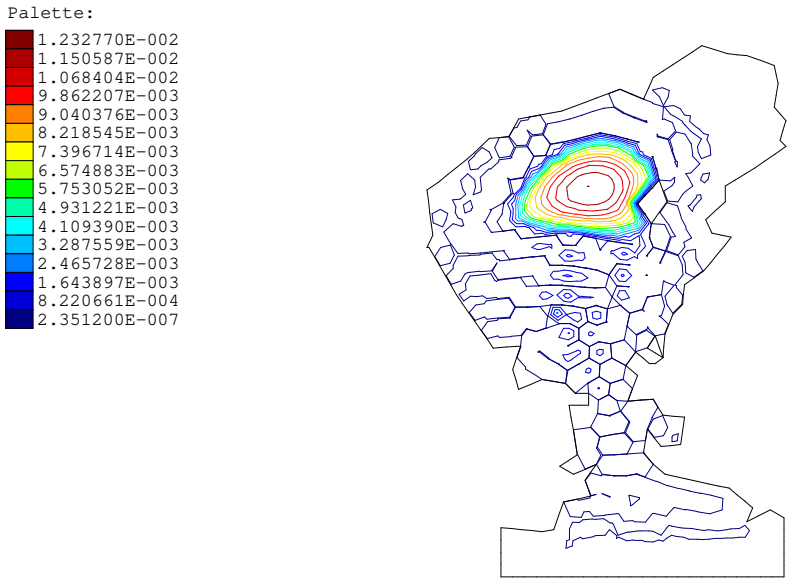

Figura 4.11: Problema de Advecção-Difusão - 100 passos no tempo.

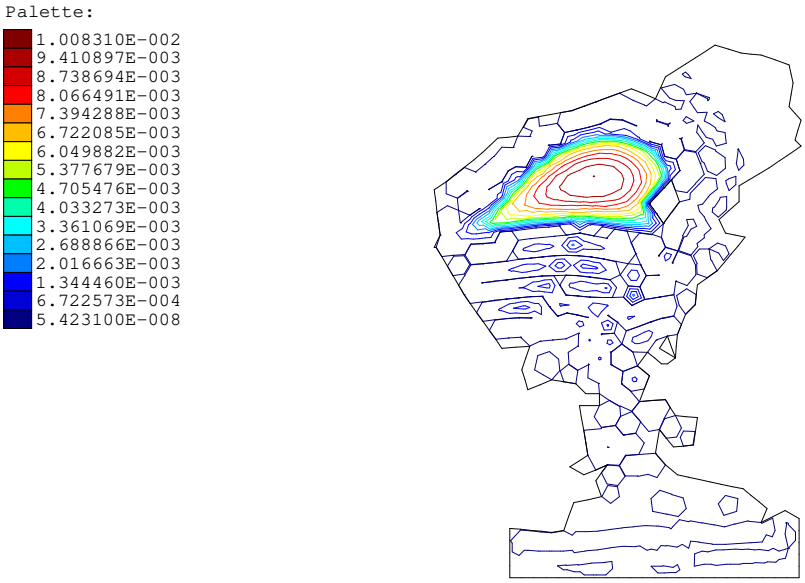

Figura 4.12: Problema de Advecção-Difusão - 200 passos no tempo.

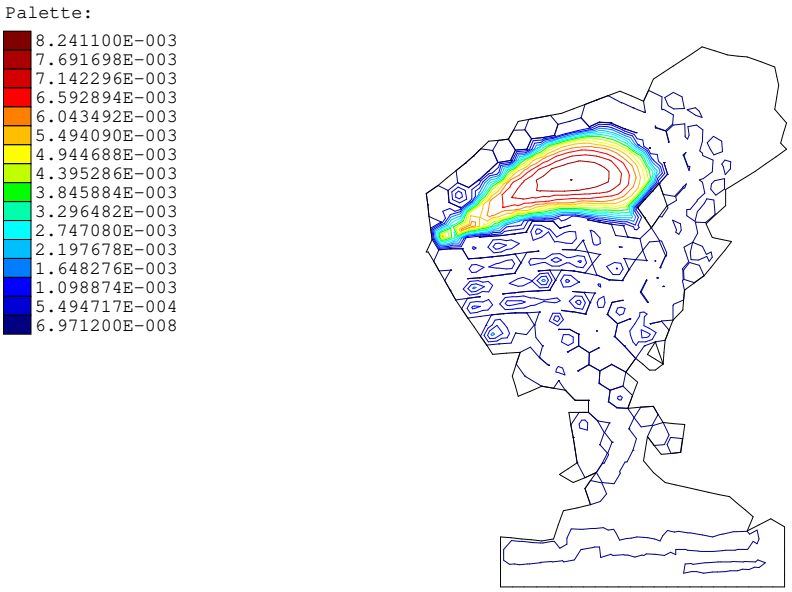

Figura 4.13: Problema de Advecção-Difusão - 250 passos no tempo.

Na simulação seguinte, consideramos o modelo de espalhamento com difusão linear do tipo $-div(c\nabla u)$. A escolha do coeficiente de difusão para este problema apresenta alguma dificuldade. No caso não-linear, tal coeficiente é da forma $3cu^2$ e varia, no primeiro passo de tempo, entre 0 e 3×10^{-5}, para a simulação apresentada nas Figuras 4.10-4.13. Em [6], foram feitos testes com c constante variando entre 10^{-5} e 5×10^{-3}. Para a próxima experiência, tomamos o menor destes valores: $c = 10^{-5}$.

A Figura 4.14 apresenta a mancha, partindo da mesma condição inicial da simulação anterior (Figura 4.10), depois de 250 iterações no tempo.

Comparando este resultado com o respectivo (Figuras 4.13) de difusão não-linear, observamos claramente como o modelo linear apresenta excessiva difusão, causando a contaminação de grande parte da baía. Por outro lado, o aumento exagerado do suporte da solução torna mais rápida a evaporação da mancha, causando a diminuição da gravidade do derrame.

Assim, o modelo com difusão não-linear parece ser o mais adequado porque apresenta difusão proprocional à quantidade de óleo existente em cada região do domínio.

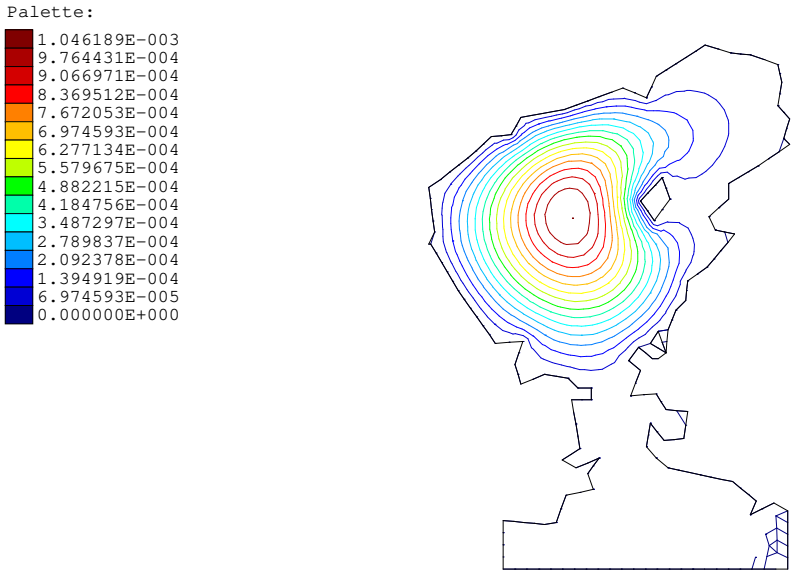

Figura 4.14: Problema Linear de Advecção-Difusão - 250 passos no tempo.

Na próxima simulação, novamente com a equação (4.6), consideramos uma mancha observada na entrada da Baía de Guanabara (Figura 4.15), resultante, por exemplo, de um vazamento em um navio petroleiro na entrada da baía. As Figuras 4.16-4.17, apresentam a evolução de tal mancha.

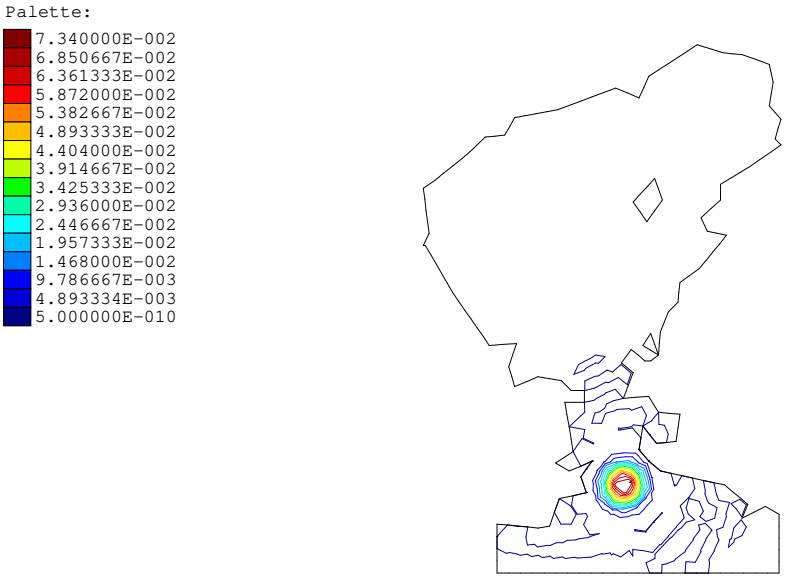

Figura 4.15: Problema de Advecção-Difusão - Condição Inicial.

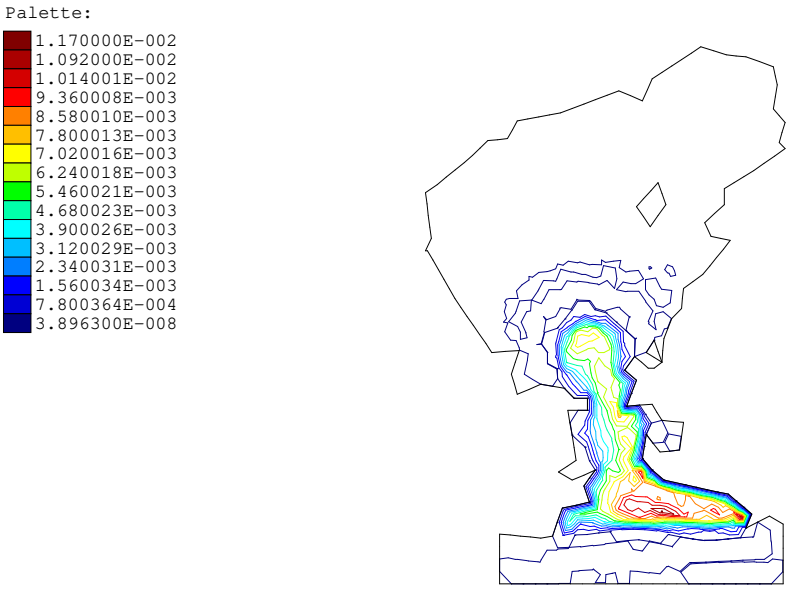

Figura 4.16: Problema de Advecção-Difusão - 150 passos no tempo.

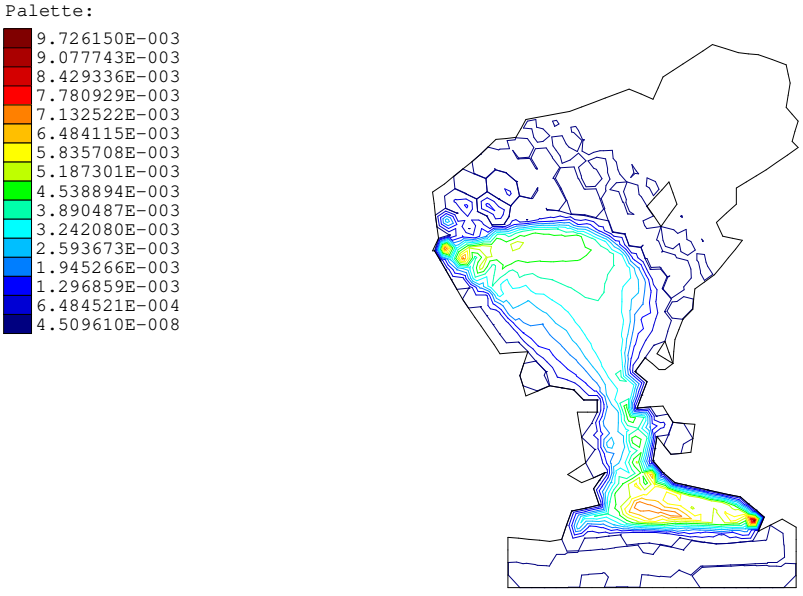

Figura 4.17: Problema de Advecção-Difusão - 250 passos no tempo.

Inicialmente, em virtude das velocidades maiores estarem concentradas na entrada da baía, a mancha se propaga rapidamente para seu interior. A partir das iterações seguintes,

a mancha se movimenta em direção às praias de São João do Meriti, deixando poluição também no Rio de Janeiro e em Niterói.

O campo de velocidades é o parâmetro essencial na determinação do deslocamento do óleo. Nas simulações tendo como cenário a Baía de Guanabara, utilizamos sempre o mesmo campo de velocidades em virtude de que alterações na velocidade de entrada da baía não causaram mudanças significativas em tal campo, sendo desnecessário repetí-las.

Concluímos enfatizando que as simulações realizadas ainda são bastante hipotéticas. Para torná-las mais realistas, salientamos a necessidade de alguns dados complementares, de difícil obtenção e grande importância:

- observações sobre a velocidade dos ventos e correntes na entrada da Baía de Guanabara, assim como no seu interior, em vários períodos do ano, para possibilitar comparações com os resultados obtidos pelo simulador de escoamentos incompressíveis que utilizamos;
- mapas mais detalhados onde apareçam, por exemplo, as Ilhas do Governador e do Fundão, inclusive com a localização e vazão dos rios que desembocam na baía, para incluí-los como fontes no cálculo do escoamento;
- profundidade da Baía de Guanabara, para a utilização das equações de águas-rasas no modelo;
- dados sobre derramamentos verdadeiros, para a comparação dos resultados e calibragem dos parâmetros envolvidos nas equações, além do detalhamento das regiões pesqueiras, de mangezais e turísticas, para a realização da análise do impacto social e ecológico dos acidentes, bem como para o planejamento de ações visando minimizá-los.

Apesar da falta destes dados, acreditamos ter conseguido isolar a obtenção de soluções numéricas para o modelo de espalhamento, que era o objetivo principal deste trabalho, e além disso, apontamos para simulações em situações reais, que em trabalhos futuros poderão se tornar cada vez mais completas.

Conclusões

O objetivo principal deste trabalho foi propor um método numérico eficaz, beseado na técnica de elementos finitos, para a realização de simulações numéricas com um modelo não-linear de advecção-difusão.

Inicialmente, vários métodos recorrentes na literatuta específica foram implementados e testados em problemas com dificuldades controladas, principalmente em relação à geometria dos domínios, com o intuito de tornar claras as vantagens e desvantagens destes métodos.

Neste ponto, concluímos que os métodos que discretizaram o tempo por diferenças finitas e o espaço por elementos finitos, assim como o método Semi-Lagrangiano, controlaram as oscilações da solução numérica às custas de difusão artificial excessiva, produzindo soluções que não podem ser consideradas boas aproximações para a solução do problema. Além disso, este último método apresentou dificuldades adicionais para a sua implementação que foram resolvidas pela utilização ção de um mecanismo, rápido e eficaz, para a localização de pontos em malhas de elementos finitos.

A discretização unificada da variável temporal com as variáveis espaciais, via método *Streamline Diffusion* (**PGDT**), eliminou o excesso de difusão artificial e conseguiu controlar as oscilações espúrias das soluções numéricas, mas aumentou o esforço computacional se comparado ao esforço demandado pelos demais métodos.

Na etapa seguinte, introduzimos modificações no método **PGDT** de forma a garantir a conservação da massa da solução ao longo do tempo, uma importante propriedade presente no problema estudado, dando origem ao método que convencionamos chamar de **PGDT-CM** e que obteve soluções com boa qualidade.

Além disso, uma segunda contribuição foi incorporada ao método **PGDT-CM**, de forma a aumentar sua eficiência em termos de tempo de CPU, tornando-o comparável aos métodos mais rápidos explorados neste trabalho. Tal redução no tempo de execução foi conseguida pela reestruturação de seu algoritmo, que passou a resolver o problema apenas numa região contendo o suporte da solução e não no domínio todo, diminuindo assim a dimensão do sistema linear a cada nível de tempo.

Na parte final desta tese, apontamos para simulações mais realistas, considerando problemas de espalhamento de manchas de petróleo em situações mais gerais. Para isto, incorporamos aos programas computacionais o cálculo do escoamento incompressível utilizando as equações de Navier-Stokes. Para a validação do código de nosso simulador de escoamantos, nos utilizamos de problemas clássicos da literatura - escoamento entre placas paralelas, cuja solução é conhecida de forma fechada, e cavidade quadrada com tampa móvel. Além disso, apresentamos simulações numéricas de derramamentos de petróleo tendo como cenário a Baía de Guanabara, no Estado do Rio de Janeiro. Com tais simulações, pudemos perceber que as previsões obtidas com o modelo não-linear de advecção-difusão foram muito mais precisas em relação ao modelo linear.

Entretanto, as simulações realizadas ainda foram bastante hipotéticas em virtude da falta de alguns dados complementares tais como: a velocidade dos ventos e correntes na Baía de Guanabara, mapas mais detalhados, profundidade da baía, dados sobre derramamentos verdadeiros para a realização de comparações.

Em relação ao cálculo do campo de velocidades na baía, cabe também ressaltar a importância da utilização das equações de águas-rasas. Além disso, a investigação da possibilidade de paralelização dos algoritmos desenvolvidos neste trabalho surgem como tema natural para trabalhos futuros.

Portanto, acreditamos ter conseguido atingir o objetivo principal deste trabalho, ou seja, construir um simulador para o modelo de espalhamento, confiável e preparado para simulações em situações reais.

Apêndice A

Aspectos Computacionais

A.1 Introdução

Nos capítulos anteriores, formulamos o método de Petrov-Galerkin Descontínuo no Tempo com Controle de Massa para a resolução de problemas do tipo $(2.1)-(2.4)$. Mas a geração de um código computacional capaz de reproduzir tal método passa por vários problemas, entre eles a discretização do domínio, montagem dos sistemas lineares, resolução dos mesmos e visualização dos resultados. Neste ponto, queremos explicar cada uma das etapas necessárias para contornar estas dificuldades.

Nas primeiras versões deste programa, havia uma simplificação com respeito aos domínios. Estes deveriam ser retangulares e sua triangularização era gerada pelo próprio programa que dividia-o em retângulos homogêneos que eram redivididos ao meio, por uma de suas diagonais, dando origem a dois triângulos, cada um. A visualização era feita através do Matlab.

Com o andamento das pesquisas, percebemos a necessidade da utilização de softwares específicos de triangularização. Tivemos acesso, então, ao toolbox PDETOOL do Matlab, desenvolvido com o intuito de resolver alguns tipos de equações diferenciais parcias pelo método de elementos finitos, desde a criação do domínio, até a visualização dos resultados. O PDETOOL gerava os domínios com as respectivas triangularizações que eram organizadas em arquivos para utilização por nosso programa. Este por sua vez resolvia o problema e devolvia a solução em arquivo, no formato do PDETOOL, para posterior visualização dos resultados neste mesmo pacote computacional. Mas ainda havia uma limitação, o PDETOOL só trabalhava com malhas com interpolação linear. Para problemas que exigissem testes com malhas para interpolação quadrática, passamos a utilizar o software GEMESIS, responsável pela criação do domínio e geração de malhas de elementos finitos para Lagrange linear ou quadrática. Juntamente com o GEMESIS, utilizamos o software VED, para a análise visual dos resultados.

A.2 Utilização do PGDTCMSUB

Vamos explicar como se executa o programa PGDTCMSUB na simulação numérica de problemas de advecção-difusão do tipo apresentado em (2.1) – (2.4).

Para iniciar, precisaremos de uma malha de elementos finitos para o domínio em questão, gerada por um dos softwares citados na seção anterior, o VED ou o PDETOOL do Matlab, e guardadas em arquivo texto com a seguinte formatação:

```
Numero Maximo de Referencias
08
Numero de Funcoes da Base Local
03
Numero de Pontos da Malha
01840
Matriz de Coordenadas dos Pontos
00001        3.5000000000       -1.0000000000
00002        3.5000000000        1.0000000000
00003       -3.5000000000        1.0000000000
00004       -3.5000000000       -1.0000000000
00005       -0.2500000000        0.0000000000
00006        0.0000000000       -0.2500000000
00007        0.2500000000        0.0000000000

Numero de Elementos da Malha
03520
Matriz de Incidencia dos Nos (Enumeracao Local X Global)
00001   3   00090   01100  01661   00001
00002   3   00097   01208  01633   00001
00003   3   00018   00488  01167   00001
00004   3   00019   00489  00921   00001
00005   3   00039   00562  01322   00001
00006   3   00030   00502  00898   00001
00007   3   00024   00534  00580   00001

Numero de Nos da Malha
01840
Vetor de Referencia dos Nos (Numero do lado ou 0 se interior)

00001     04
00002     01
00003     03
```

```
00004       00
00005       08
00006       05
```

Arquivos com este aspecto podem ser construídos à partir dos programas SDSAFE e SDMLAB, que listamos a seguir. O primeiro, transforma arquivos gerados no formato GEMESIS e o segundo no formato do PDETOOL.

```
C-----------------------------------------------------------------------
C             FILE                                    SDSAFE.FOR
C-----------------------------------------------------------------------
C
C
      PARAMETER( MAX = 10000 )
C
C
      REAL*8 C(2,MAX)
C
      INTEGER*2 M(9,MAX), INB(MAX)
C
C
      CALL CONTROLE(C,M,INB)
C
C
C
      END
C-----------------------------------------------------------------------
C             FILE                                    CONTROLE.FOR
C-----------------------------------------------------------------------
C
C
      SUBROUTINE CONTROLE(C,M,INB)
C     ============================
C
      REAL*8  C(2,*)
C
      INTEGER*2 M(9,*), INB(*)
      INTEGER*2 NFB, NREF, NOE, NE, IEL
C
      CHARACTER res*1
C
C
C
1000  write(*,1100)
1100  format(//,T10,'Deseja Ler a Estrutura  FEM ?  (S)/(N)  :  ',$)
      read(*,1200) res
1200  format(A1)
       if((res.eq.'N').or.(res.eq.'n'))  go to 1300
       if((res.ne.'S').and.(res.ne.'s')) go to 1000
C
C
       CALL ARANHA(C,M,INB,NFB,NREF,NOE,NE)
c
c
c      ----------------------------------------------------
c      Escreve a Estrutura da Malha em um Arquivo de Dados
c      ----------------------------------------------------
c
c
c      write(*,*) '     Chamada da Subroutine  MALHA'
       CALL GERAMALHA(C,M,INB,NFB,NREF,NOE,NE)
c
c      ----------------------------------------------------
c      Leitura da Estrutura da Malha de um Arquivo de Dados
c      ----------------------------------------------------
c
1300   CALL LEEMALHA(C,M,INB,NFB,NREF,NOE,NE)
C
C
       CALL INFOREDE(NFB,NREF,NE,NOE,IEL)
C
C
C
       RETURN
       END
```

```
C-----------------------------------------------------------------------
C            FILE                                  INFOREDE.FOR
C-----------------------------------------------------------------------
C
       SUBROUTINE INFOREDE(NFB,NREF,NTEL,NTNOS,IEL)
C      ============================================
C
       INTEGER*2 NFB, NREF, NTEL, NTNOS, IEL
C
       CHARACTER ELEMENTO*19
C
C      ----------------------
C      INFORMACOE SOBRE A REDE
C      ----------------------
C
     IF(NFB.EQ.3)   THEN
       IEL = 1
       ELEMENTO = '    Lagrange Linear'
     ELSE IF(NFB.EQ.6)  THEN
       IEL = 2
       ELEMENTO = 'Lagrange Quadratico'
     ELSE
       WRITE(*,*) '          Erro na Escolha do Espaco de Aproximacao'
       STOP
     END IF
C
C
     WRITE(*,100)
100  FORMAT(5(/),T10,'Informacoes Sobre a Rede',/)
C
     WRITE(*,200) NFB, NREF, NTEL, NTNOS, ELEMENTO
200  FORMAT(2(/),T10,
    * '  Numero de Funcoes de Base Local   ',I1.1,//,T10,
    *  '            Numero de Referencias   ',I2.2,//,T10,
    *  'Numero Total de Elementos da Rede   ',I6.6,//,T10,
    *  '      Numero Total de Nos da Rede   ',I6.6,//,T24,A19,//)
C
C
C
     RETURN
     END
C-----------------------------------------------------------------------
C          FILE                                 GERAMALHA.FOR
C-----------------------------------------------------------------------
C
     SUBROUTINE GERAMALHA(C,M,INB,NFB,NREF,NOE,NE)
C    =============================================
C
     REAL*8  C(2,*)
C
     INTEGER*2 M(9,*), INB(*), NFB, NREF, NOE, NE, IFLAG
C
     CHARACTER  NAME*15, res*1
C
C
C
C    --------------------
C    Variaveis de Entrada
C    --------------------
C
C       C : Matriz de Coordenadas dos Nos
C
C       M : Matriz de Incidencia dos Nos
C
C     INB : Vetor de Referencia dos Nos
C
C    NREF : Numero Maximo de Referencias
C
C     NFB : Numero de Funcoes da Base Local
C
C      NE : Numero de Elementos da Malha
C
C     NOE : Numero de Nos da Malha
C
C
C    ----------------------------
C    Escreve a Estrutura da Malha
C    ----------------------------
C
C
1800  write(*,1900)
```

```
1900  format(//,T10,
     * 'Deseja Escrever a Estrutura da Malha ? (S)/(N)  :  ',$)
      read(*,1950) res
1950  format(A1)
      if((res.eq.'N').or.(res.eq.'n'))  return
      if((res.ne.'S').and.(res.ne.'s')) go to 1800
c
c
      write(*,2000)
2000  format(//,T10,'Arquivo para Escrever a Estrutura da Malha ',$)
      read(*,2100) NAME
2100  format(A15)
c
c
      OPEN (UNIT=20, FILE=NAME, STATUS='UNKNOWN')
C
C
C
C
      WRITE(20,*) '          Numero Maximo de Referencias'
      WRITE(20,2110) NREF
2110  FORMAT(T5,I2)
      WRITE(20,*) '          Numero de Funcoes da Base Local'
      WRITE(20,2110) NFB
      WRITE(20,*) '          Numero de Pontos da Malha'
      WRITE(20,2120) NOE
2120  FORMAT(T5,I5)
C
C
      WRITE(20,*) '          Matriz de Coordenadas dos Pontos'
C
      DO 1000 K = 1,NOE
C
      WRITE(20,2200) K,C(1,K),C(2,K)
2200  FORMAT(T5,I5,T20,F18.12,T40,F18.12)
C
1000  CONTINUE
C
C
C
      WRITE(20,*) '          Numero de Elementos da Malha'
      WRITE(20,2120) NE
C
C
      WRITE(20,*) '          Matriz de Incidencia dos Nos'
C
      DO 1200 K = 1,NE
C
      WRITE(20,2300) K, NFB, ( M(J,K) , J = 1,NFB  )
2300  FORMAT(T5,I5,T12,I1,T16,6(I5,3X))
C
1200  CONTINUE
C
C
      WRITE(20,*) '          Numero de Nos da Malha'
      WRITE(20,2120) NOE
C
C
      WRITE(20,*) '          Vetor de Referencia dos Nos'
C
      DO 1300 K = 1,NOE
C
      WRITE(20,2400) K,INB(K)
2400  FORMAT(T5,I5,T20,I5)
C
1300  CONTINUE
C
C
C
      IFLAG = 0
      WRITE(20,*) 'Correcao dos KeyPoints  -  (0) Nao    (1)  Sim '
      WRITE(20,2120) IFLAG
C
C
      CLOSE(UNIT=20)
C
C
C
      RETURN
      END
C---------------------------------------------------------------------
```

```
C            FILE                              LEEMALHA.FOR
C-----------------------------------------------------------------------
C
C
      SUBROUTINE LEEMALHA(C,M,INB,NFB,NREF,NOE,NE)
C     ============================================
C
      REAL*8  C(2,*)
C
      INTEGER*2 M(9,*), INB(*)
      INTEGER*2 NE, NOE, NREF, NFB, NNO, IFLAG
C
      CHARACTER  NAME*15, TEXTO*60, res*1
C
C
C
C     ------------------
C     Variaveis de Saida
C     ------------------
C
C      C : Matriz de Coordenadas dos Nos
C
C      M : Matriz de Incidencia dos Nos
C
C    INB : Vetor de Referencia dos Nos
C
C   NREF : Numero Maximo de Referencias
C
C    NFB : Numero de Funcoes da Base Local
C
C     NE : Numero de Elementos da Malha
C
C    NOE : Numero de Nos da Malha
C
C
C
C      ----------------------------
C      Leitura da Estrutura da Malha
C      ----------------------------
C
C
2000  write(*,2100)
2100  format(//,T10,
     * 'Deseja Acertar os KeyPoints da Malha  ?  (S)/(N)  :   ',$)
      read(*,2150) res
2150  format(A1)
      if((res.eq.'N').or.(res.eq.'n'))  return
      if((res.ne.'S').and.(res.ne.'s')) go to 2000
c
c
2190  write(*,2200)
2200  format(//,T10,'Arquivo de Leitura da Estrutura da Malha ',$)
      read(*,2300) NAME
2300  format(A15)
c
c
c
      OPEN (UNIT=20, FILE=NAME, STATUS='OLD', IOSTAT=IERR)
C
      IF(IERR.NE.0) THEN
      WRITE(*,2310)
2310  FORMAT(//,T10,
     * 'Arquivo de Dados Desconhecido. Informe um Novo Arquivo.')
      GO TO 2190
      END IF
C
C
C
C
      READ(20,2350) TEXTO
2350  FORMAT(A60)
      READ(20,*) NREF
C
C
      READ(20,2350) TEXTO
      READ(20,*) NFB
C
C
      READ(20,2350) TEXTO
      READ(20,*) NOE
C
```

```
C
      READ(20,2350) TEXTO
C
      DO K = 1,NOE
C
      READ(20,*) KKK,C(1,K),C(2,K)
C
      END DO
C
C
      READ(20,2350) TEXTO
      READ(20,*) NE
C
C
      READ(20,2350) TEXTO
C
C
      DO K = 1,NE
C
      READ(20,*) KKK, NNO, ( M(J,K) , J = 1,NFB )
C
      END DO
C
C
      READ(20,2350) TEXTO
      READ(20,*) NOE
C
C
      READ(20,2350) TEXTO
C
C
      DO K = 1,NOE
C
      READ(20,*) KKK,INB(K)
C
      END DO
C
C
      READ(20,2350) TEXTO
      READ(20,*) IFLAG
C
C
      CLOSE(UNIT=20)
C
C
      IF(IFLAG.EQ.0) THEN
      CALL  ACERTA_KEYPOINTS(C,INB,NOE)
      IFLAG = 1
C
      CALL  ESCREVE_MALHA(C,M,INB,NFB,NREF,NOE,NE,NAME,IFLAG)
      END IF
C
C
C
C
      RETURN
      END
C---------------------------------------------------------------------
C          FILE                          ACERTA_KEYPOINTS.FOR
C---------------------------------------------------------------------
C
      SUBROUTINE ACERTA_KEYPOINTS(C,INB,NTNOS)
C     ----------------------------------------
C
      REAL*8   C(2,*), X, Y
      real*8  cxykeyp(20,2)
C
      INTEGER*2   INB(*), NTNOS
      integer*2   nkeyp, ladokeyp(20)
C
      common / acertamalha/ cxykeyp, ladokeyp, nkeyp
C
C
        do 600 i = 1,nkeyp
c
               do j = 1,ntnos
c
                  if(inb(j).ne.0) then
c
                  X = C(1,j)
                  Y = C(2,j)
```

```
c
                if(X.eq.cxykeyp(i,1)) then
                if(Y.eq.cxykeyp(i,2)) then
                inb(j) = ladokeyp(i)
                go to 600
                end if
                end if
c
                end if
c
              end do
c
600     continue
C
C
C
        RETURN
        END
C-----------------------------------------------------------------------
C        FILE                                  ESCREVE_MALHA.FOR
C-----------------------------------------------------------------------
C
      SUBROUTINE ESCREVE_MALHA(C,M,INB,NFB,NREF,NOE,NE,NAME,IFLAG)
C     ============================================================
C
      REAL*8  C(2,*)
C
      INTEGER*2  M(9,*), INB(*), NFB, NREF, NOE, NE, IFLAG
C
      CHARACTER  NAME*60
C
C
C
C     --------------------
C     Variaveis de Entrada
C     --------------------
C
C       C : Matriz de Coordenadas dos Nos
C
C       M : Matriz de Incidencia dos Nos
C
C     INB : Vetor de Referencia dos Nos
C
C    NREF : Numero Maximo de Referencias
C
C     NFB : Numero de Funcoes da Base Local
C
C      NE : Numero de Elementos da Malha
C
C     NOE : Numero de Nos da Malha
C
C
C     ----------------------------
C     Escreve a Estrutura da Malha
C     ----------------------------
C
C
      OPEN (UNIT=20, FILE=NAME, STATUS='UNKNOWN')
C
C
C
C
      WRITE(20,*) '          Numero Maximo de Referencias'
      WRITE(20,2110) NREF
2110  FORMAT(T5,I2)
      WRITE(20,*) '          Numero de Funcoes da Base Local'
      WRITE(20,2110) NFB
      WRITE(20,*) '          Numero de Pontos da Malha'
      WRITE(20,2120) NOE
2120  FORMAT(T5,I5)
C
C
      WRITE(20,*) '          Matriz de Coordenadas dos Pontos'
C
      DO 1000 K = 1,NOE
C
      WRITE(20,2200) K,C(1,K),C(2,K)
2200  FORMAT(T5,I5,T20,F18.12,T40,F18.12)
C
1000  CONTINUE
C
```

```
C
C
      WRITE(20,*) '          Numero de Elementos da Malha'
      WRITE(20,2120) NE
C
C
      WRITE(20,*) '          Matriz de Incidencia dos Nos'
C
      DO 1200 K = 1,NE
C
      WRITE(20,2300) K, NFB, ( M(J,K) , J = 1,NFB  )
2300  FORMAT(T5,I5,T12,I1,T16,6(I5,3X))
C
1200  CONTINUE
C
C
      WRITE(20,*) '          Numero de Nos da Malha'
      WRITE(20,2120) NOE
C
C
      WRITE(20,*) '          Vetor de Referencia dos Nos'
C
      DO 1300 K = 1,NOE
C
      WRITE(20,2400) K,INB(K)
2400  FORMAT(T5,I5,T20,I5)
C
1300  CONTINUE
C
C
C
      WRITE(20,*) 'Correcao dos KeyPoints  -  (0) Nao    (1)  Sim '
      WRITE(20,2120) IFLAG
C
C
      CLOSE(UNIT=20)
C
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C             FILE                            ARANHA.FOR
C-----------------------------------------------------------------------
C
      SUBROUTINE ARANHA(C,M,INB,NFB,NREF,NOE,NE)
C     ==========================================

      REAL*8 R1,R2
      REAL*8 C(2,*)
C
      INTEGER*2 M(9,*), INB(*), NOE, NE, NREF, NFB
      INTEGER*2 I, IGRUPO, J, I1, I2, V(6)
C
      CHARACTER*15 TEMP,NOME
C
C
C

C       < ARANHA > FAZ A LEITURA DE UM ARQUIVO DE DADOS GERADO NO FORMATO
C     'PREGRAPH' PELO GERADOR DE MALHAS DO PACOTE << GEMESIS >>
C
C       A PARTIR DA LEITURA, ESTA ROTINA PREENCHE A MATRIZ DE COORDENADAS
C     ( C ), A MATRIZ DE INCIDENCIA ( M ) E O VETOR DE NOS NO BORDO ( INB )
C
C       ALEM DISSO, ELA INFORMA O NUMERO DE NOS POR ELEMENTO ( NFB ) E O
C     NUMERO DE NOS (NOE) E DE ELEMENTOS (NE) DA MALHA

C     CHAMADA DO ARQUIVO
C
C
      WRITE(*,1)
1     FORMAT(///,T10,'Leitura da Estrutura da Malha Gerada pelo SAFE')
2     WRITE(*,3)
3     FORMAT(//,T10,'Entre com o nome do arquivo ( <nome>.fem ) :  ',$)
      READ(*,4) NOME
4     FORMAT(A15)
C
C
C
      OPEN(UNIT=50, FILE=NOME, STATUS='OLD', IOSTAT=IERR)
```

```
C
      IF(IERR.NE.0) THEN
      WRITE(*,5)
5     FORMAT(//,T10,
     * 'Arquivo de Dados Desconhecido. Informe um Novo Arquivo.')
      GO TO 2
      END IF
C
C
C
C     OBTEM O NUMERO DE NOS

      TEMP = '*COORDINATES'
      CALL DESLOQUE(TEMP)
      READ(50,*) NOE

C     PREENCHE A MATRIZ DE COORDENADAS

      DO 10 J = 1,NOE
          READ(50,*) I1,R1,R2
          C(1,J) = R1
          C(2,J) = R2
10    CONTINUE
C
C     OBTEM O NUMERO DE ELEMENTOS E O TIPO DE ELEMENTO (EM TEMP)
C
      TEMP = '*ELEMENT_GROUPS'
      CALL DESLOQUE(TEMP)
C
      READ(50,*) IGRUPO
C
C
      READ(50,11) I1, NE, TEMP
11    FORMAT(I2,I4,A5)
C
C
      IF(TEMP.EQ.'TRI_3') THEN
         NFB = 3
      ELSE
         NFB = 6
      END IF

C     PREENCHE A MATRIZ DE INCIDENCIA

      TEMP = '*INCIDENCES'
      CALL DESLOQUE(TEMP)
      DO 20 J = 1,NE
          READ(50,*) I1,(V(I2), I2=1,NFB)
          DO 30 I = 1,NFB
              M(I,J) = V(I)
30        CONTINUE
20    CONTINUE

C     INICIALIZA O VETOR DE NOS NO BORDO

      DO 40 I = 1,NOE
         INB(I) = 0
40     CONTINUE

C     OBTEM O NUMERO DE FACES DO BORDO ( NREF )

      TEMP = '*LINES'
      CALL DESLOQUE(TEMP)
      READ(50,*) NREF

C     PREENCHE O VETOR DE NOS DO BORDO ( SE O NO i ESTA NA FACE j DO BORDO,
C     ENTAO INB(i) = J )

      DO 60 I = 1,NREF
          CALL AVANCE_LINHA
          READ(50,*) I1,I1
          DO 70 J = 1,I1
              READ(50,*) I2
              INB(I2) = I
70        CONTINUE
60    CONTINUE


      CLOSE(UNIT = 50)
C
C
```

```
C
      RETURN
      END

C-----------------------------------------------------------------------
        FILE                                  DESLOQUE.FOR
C-----------------------------------------------------------------------
C
      SUBROUTINE DESLOQUE(REF)
C     ========================

      CHARACTER*15 REF,TEMP

C        POSICIONA A LEITURA NO ARQUIVO PARA APOS A STRING 'REF'
C        ( SUBROTINA DA ROTINA <ARANHA> )


105   CONTINUE
        READ(50,15) TEMP
15      FORMAT(A15)
      IF(TEMP.NE.REF) GOTO 105
C
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C       FILE                                  AVANCE_LINHA.FOR
C-----------------------------------------------------------------------
C
      SUBROUTINE AVANCE_LINHA
C     ========================

      CHARACTER TEMP

C        PULA UMA LINHA NA LEITURA DO ARQUIVO
C        ( SUBROTINA DA ROTINA <ARANHA> )

      READ(50,25) TEMP
25    FORMAT(A1)
C
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
```

```
C-----------------------------------------------------------------------
C             FILE                                  SDMLAB.FOR
C-----------------------------------------------------------------------
C
C
      PARAMETER( MAX = 20000 )
C
C
      REAL*8 C(2,MAX)
C
      INTEGER*2 M(6,MAX), INB(MAX)
C
C
      CALL CONTROLE(C,M,INB)
C
C
C
      END
C-----------------------------------------------------------------------
C             FILE                                CONTROLE.FOR
C-----------------------------------------------------------------------
C
C
      SUBROUTINE CONTROLE(C,M,INB)
C     ============================
C
      REAL*8  C(2,*)
```

```
C
      INTEGER*2 M(6,*), INB(*)
      INTEGER*2 NFB,NREF,NOE,NE
C
      CHARACTER res*1
C
C
C
1000  write(*,1100)
1100  format(//,T10,
     *  'Deseja Ler a Estrutura  Gerada pelo MatLab ?  (S)/(N)  :  ',$)
      read(*,1200) res
1200  format(A1)
      if((res.eq.'N').or.(res.eq.'n'))  go to 1300
      if((res.ne.'S').and.(res.ne.'s')) go to 1000
C
C
      CALL LEEMESH(C,M,INB,NFB,NREF,NOE,NE)
c
c
c     ------------------------------------------------
c     Escreve a Estrutura da Malha em um Arquivo de Dados
c     ------------------------------------------------
c
c
c     write(*,*) '     Chamada da Subroutine  MALHA'
      CALL GERAMALHA(C,M,INB,NFB,NREF,NOE,NE)
C
C
c     ------------------------------------------------
c     Leitura da Estrutura da Malha de um Arquivo de Dados
c     ------------------------------------------------
C
C
1300  CONTINUE
C
C
C
      RETURN
      END
C---------------------------------------------------------------------
C         FILE                                GERAMALHA.FOR
C---------------------------------------------------------------------
C
C
      SUBROUTINE GERAMALHA(C,M,INB,NFB,NREF,NOE,NE)
C     ============================================
C
      REAL*8  C(2,*)
C
      INTEGER*2 M(6,*), INB(*), NFB, NREF, NOE, NE
C
      CHARACTER  NAME*15, res*1
C
C
C
C     --------------------
C     Variaveis de Entrada
C     --------------------
C
C        C : Matriz de Coordenadas dos Nos
C
C        M : Matriz de Incidencia dos Nos
C
C      INB : Vetor de Referencia dos Nos
C
C     NREF : Numero Maximo de Referencias
C
C      NFB : Numero de Funcoes da Base Local
C
C       NE : Numero de Elementos da Malha
C
C      NOE : Numero de Nos da Malha
C
C
C
C     ----------------------------
C     Escreve a Estrutura da Malha
C     ----------------------------
C
C
```

```
1800  write(*,1900)
1900  format(//,T10,
     *  'Deseja Escrever a Estrutura da Malha ? (S)/(N)  :  ',$)
      read(*,1950) res
1950  format(A1)
      if((res.eq.'N').or.(res.eq.'n'))  return
      if((res.ne.'S').and.(res.ne.'s')) go to 1800
c
c
      write(*,2000)
2000  format(//,T10,'Arquivo para Escrever a Estrutura da Malha ',$)
      read(*,2100) NAME
2100  format(A15)
c
c
      OPEN (UNIT=20, FILE=NAME, STATUS='UNKNOWN')
C
C
C
C
      WRITE(20,*) '          Numero Maximo de Referencias'
      WRITE(20,2110) NREF
2110  FORMAT(T5,I2)
      WRITE(20,*) '          Numero de Funcoes da Base Local'
      WRITE(20,2110) NFB
      WRITE(20,*) '          Numero de Pontos da Malha'
      WRITE(20,2120) NOE
2120  FORMAT(T5,I5)
C
C
      WRITE(20,*) '          Matriz de Coordenadas dos Pontos'
C
      DO 1000 K = 1,NOE
C
      WRITE(20,2200) K,C(1,K),C(2,K)
2200  FORMAT(T5,I5,T20,F18.12,T40,F18.12)
C
1000  CONTINUE
C
C
C
      WRITE(20,*) '          Numero de Elementos da Malha'
      WRITE(20,2120) NE
C
C
      WRITE(20,*) '          Matriz de Incidencia dos Nos'
C
      DO 1200 K = 1,NE
C
      WRITE(20,2300) K, NFB, ( M(J,K) , J = 1,NFB  )
2300  FORMAT(T5,I5,T12,I1,T16,6(I5,3X))
C
1200  CONTINUE
C
C
      WRITE(20,*) '          Numero de Nos da Malha'
      WRITE(20,2120) NOE
C
C
      WRITE(20,*) '          Vetor de Referencia dos Nos'
C
      DO 1300 K = 1,NOE
C
      WRITE(20,2400) K,INB(K)
2400  FORMAT(T5,I5,T20,I5)
C
1300  CONTINUE
C
      CLOSE(UNIT=20)
C
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C            FILE                              LEEMALHA.FOR
C-----------------------------------------------------------------------
C
C
      SUBROUTINE LEEMALHA(C,M,INB,NFB,NREF,NOE,NE)
C     ==========================================
```

```
C
      REAL*8  C(2,*)
C
      INTEGER*2 M(6,*), INB(*)
      INTEGER*2 NE, NOE, NREF, NFB, NNO
C
      CHARACTER  NAME*15, TEXTO*60, res*1
C
C
C
C     ------------------
C     Variaveis de Saida
C     ------------------
C
C        C : Matriz de Coordenadas dos Nos
C
C        M : Matriz de Incidencia dos Nos
C
C      INB : Vetor de Referencia dos Nos
C
C     NREF : Numero Maximo de Referencias
C
C      NFB : Numero de Funcoes da Base Local
C
C       NE : Numero de Elementos da Malha
C
C      NOE : Numero de Nos da Malha
C
C
C
C     ----------------------------
C     Leitura da Estrutura da Malha
C     ----------------------------
C
C
2000  write(*,2100)
2100  format(//,T10,
     *   'Deseja Ler a Estrutura da Malha ? (S)/(N) :   ',$)
      read(*,2150) res
2150  format(A1)
      if((res.eq.'N').or.(res.eq.'n'))  return
      if((res.ne.'S').and.(res.ne.'s')) go to 2000
c
c
2190  write(*,2200)
2200  format(//,T10,'Arquivo de Leitura da Estrutura da Malha ',$)
      read(*,2300) NAME
2300  format(A15)
c
c
c
      OPEN (UNIT=20, FILE=NAME, STATUS='OLD', IOSTAT=IERR)
C
      IF(IERR.NE.0) THEN
      WRITE(*,2310)
2310  FORMAT(//,T10,
     * 'Arquivo de Dados Desconhecido. Informe um Novo Arquivo.')
      GO TO 2190
      END IF
C
C
C
C
      READ(20,2350) TEXTO
2350  FORMAT(A60)
      READ(20,*) NREF
C
C
      READ(20,2350) TEXTO
      READ(20,*) NFB
C
C
      READ(20,2350) TEXTO
      READ(20,*) NOE
C
C
      READ(20,2350) TEXTO
C
      DO K = 1,NOE
C
      READ(20,*) KKK,C(1,K),C(2,K)
```

```
C
      END DO
C
C
      READ(20,2350) TEXTO
      READ(20,*) NE
C
C
      READ(20,2350) TEXTO
C
C
      DO K = 1,NE
C
      READ(20,*) KKK, NNO, ( M(J,K) , J = 1,NFB )
C
      END DO
C
C
      READ(20,2350) TEXTO
      READ(20,*) NOE
C
C
      READ(20,2350) TEXTO
C
C
      DO K = 1,NOE
C
      READ(20,*) KKK,INB(K)
C
      END DO
C
      CLOSE(UNIT=20)
C
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C         FILE                                  LEEMESH.FOR
C-----------------------------------------------------------------------
C
      SUBROUTINE LEEMESH(C,M,INB,NFB,NREF,NOE,NE)
C     ===========================================
C
      REAL*8  C(2,*)
C
      INTEGER*2 M(6,*), INB(*)
      INTEGER*2 NE, NOE, NREF, NFB
C
      CHARACTER  NAME*15, TEXTO*60
C
C
C
C     ------------------
C     Variaveis de Saida
C     ------------------
C
C          C : Matriz de Coordenadas dos Nos
C
C          M : Matriz de Incidencia dos Nos
C
C        INB : Vetor de Referencia dos Nos
C
C       NREF : Numero Maximo de Referencias
C
C        NFB : Numero de Funcoes da Base Local
C
C         NE : Numero de Elementos da Malha
C
C        NOE : Numero de Nos da Malha
C
C
C
C     ----------------------------
C     Leitura da Estrutura da Malha
C     ----------------------------
C
C
c
c
2190  write(*,2200)
```

```
2200  format(//,T10,'Arquivo de Leitura da Estrutura da Malha ',$)
      read(*,2300) NAME
2300  format(A15)
c
c
c
      OPEN (UNIT=20, FILE=NAME, STATUS='OLD', IOSTAT=IERR)
C
      IF(IERR.NE.0) THEN
      WRITE(*,2310)
2310  FORMAT(//,T10,
     *  'Arquivo de Dados Desconhecido. Informe um Novo Arquivo.')
      GO TO 2190
      END IF
C
C
C
C
      READ(20,2350) TEXTO
2350  FORMAT(A60)
      READ(20,*) NREF
C
C
      READ(20,2350) TEXTO
      READ(20,*) NFB
C
C
      READ(20,2350) TEXTO
      READ(20,*) NOE
C
C
      READ(20,2350) TEXTO
C
      DO K = 1,NOE
C
      READ(20,*)  C(1,K),C(2,K)
C
      END DO
C
C
      READ(20,2350) TEXTO
      READ(20,*) NE
C
C
      READ(20,2350) TEXTO
C
C
      DO K = 1,NE
C
      READ(20,*) ( M(J,K) , J = 1,NFB )
C
      END DO
C
C
      READ(20,2350) TEXTO
      READ(20,*) NOE
C
C
      READ(20,2350) TEXTO
C
C
      DO K = 1,NOE
C
      READ(20,*) KKK,INB(K)
C
      END DO
C
      CLOSE(UNIT=20)
C
C
C
      RETURN
      END
C---------------------------------------------------------------------
C---------------------------------------------------------------------
```

Os coeficientes do problema podem ser descritos no arquivo **"funcoes.for"** que contém as seguintes funções: FONTE - expressão matemática da função que representa a condição inicial do problema, SIGMA - constante de evaporação, ALFA - constante que acompanha o termo de difusão não-linear, CONVECCAO - expressão do campo de velocidades e CONTORNO - condição de contorno de Dirichlet.

```
C-----------------------------------------------------------------------
C         FILE                                FUNCOES.FOR
C-----------------------------------------------------------------------
C
      REAL*8 FUNCTION FONTE(X,Y)
C     ==========================
C
      REAL*8 X,Y,XX,YY, ALTURA
C
C
C     ----------------------------
C     Condicao Inicial do Problema
C     ----------------------------
C
C
      ALTURA = 1.0D-1
      XX = (X+2.5d0)*(X+2.5d0)
      YY = Y*Y
      FONTE =  ALTURA*DEXP( - 16.0D0*( XX + YY ) )
c
C
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
C
      REAL*8 FUNCTION SIGMA(X,Y)
C     ==========================
C
      REAL*8 X, Y
C
C
C
C     -------------------
C     Termo de Evaporacao
C     -------------------
C
      SIGMA  =  0.0d0
C
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
C
      REAL*8 FUNCTION ALFA(X,Y)
C     =========================
C
      REAL*8 X,Y,ALPHA
C
      COMMON / DIFUSION / ALPHA
c
C
C     ---------------------------------------------
C     ALPHA - Lida no arquivo de entrada Malha.dat
C     ---------------------------------------------
C
C     ---------------------------------------------
C     Constante Multiplicativa do Termo de Difusao
C     ---------------------------------------------
C
      ALFA  =  ALPHA
C
C
C
      RETURN
      END
C-----------------------------------------------------------------------
```

```
C-----------------------------------------------------------------------
C
      SUBROUTINE CONVECCAO(X,Y,BETA)
C     ==============================
C
      REAL*8 X, Y, BETA(*)
C
C
C
C     ----------------------------------------------------
C     Termo de Conveccao     BETA(X,Y) - Campo de Velocidades
C     ----------------------------------------------------
C
C
C
      BETA(1) =   (1.0D0 - Y*Y)
      BETA(2) =   0.0D0
C
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
C
      REAL*8 FUNCTION CONTORNO(X,Y,ILD)
C     =================================
C
      REAL*8 X,Y
C
      INTEGER ILD
C
C
C     ILD  :   Numero de Referencia do Bordo
C
C
C     ----------------------------------
C     Condicoes de Contorno de Dirichlet
C     ----------------------------------
C
      IF(ILD.EQ.1)  CONTORNO =  0.0D0
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
```

Os demais parâmetros para a simulação numérica do problema encontram-se no arquivo arquivo **“Malha.dat”**:

```
C---------------------------------------------------------------
                      Arquivo Malha.dat
C---------------------------------------------------------------

     Numero de Vertices do Elemento (elemento tridimensional)
     6
     Interpolacao Polinomial =  (1) Lagrange Linear  (2) Quadratico
     1
     Integracao Numerica  =  (1)  Quadratic     (2)  Cubic     (3)  Quintic
     3
     Passo no Tempo     Niveis de Tempo
     0.05               120
     Coeficiente de Difusao - ALPHA
     1.0d-3
     Constante para a Escolha do Parametro Delta em Petrov-Galerkin
     5.0d-1
     Arquivo Contendo as Informacoes sobre a Malha
     canal3.dat
C---------------------------------------------------------------
C---------------------------------------------------------------
```

O arquivo **"Saigraf.dat"** contém a informação referente aos arquivos de saída gráfica que devem ser gerados.

```
C-----------------------------------------------------------------
            Arquivo Saigraf.dat
C-----------------------------------------------------------------

    Numero de Graficos, Numero de Iteracoes no Tempo, Nome do Arquivo
     8      20 40 60 80 90 100 110 120
     muc01.m
     muc02.m
     muc03.m
     muc04.m
     muc05.m
     muc06.m
     muc07.m
     muc08.m
C-----------------------------------------------------------------
C-----------------------------------------------------------------
```

A.3 O Programa Principal e Suas Subrotinas

Nesta seção, pretendemos expor todas as subrotinas que integram a implementação do método de Petrov-Galerkin Descontínuo no Tempo com Controle de Massa, de forma a propiciar que um leitor interessado neste tipo de método de elementos finitos possa adaptar este programa às suas necessidades.

```
c-----------------------------------------------------------------
c-----------------------------------------------------------------
c
c
c
c
c              ================================
c              PROGRAMA  PRINCIPAL   PGDTCMSUB
c              ================================
c
c
c
c
      INTEGER  NEMAX, MAXICN, MAXIRN, NNMAX
c
      PARAMETER  (   NEMAX =  25000  )
      PARAMETER  (   NNMAX =  36000  )
      PARAMETER  (  MAXICN =  900000  )
      PARAMETER  (  MAXIRN =  900000  )
c
c
c
      REAL*8  F(NNMAX), U(NNMAX), C(2,NNMAX), CNEW(2,NNMAX)
      REAL*8  A(MAXICN), X , Y, PIVO, W(NNMAX), USD(NNMAX)
      REAL*8  EPS, TETHA, STEP, SOLIDO(2000), INTSURF
      REAL*8  K1SUB(2,2), M1SUB(2,2), V1SUB(2,2), MAXIMO
      REAL*8  MAXX, MAXY, MINX, MINY, Mn, Mntil, MO
      REAL*8  XMAX(NEMAX), YMAX(NEMAX), XMIN(NEMAX), YMIN(NEMAX)
c
c
      INTEGER  M(6,NEMAX), INB(NNMAX), NELEM, VAR(NNMAX)
      INTEGER  ILIN(MAXIRN), JCOL(MAXICN), LISTA(NEMAX), LISTANEW(NEMAX)
      INTEGER  IROWST(NNMAX), LENROW(NNMAX), LISTAUX(NEMAX)
      INTEGER  IKEEP(5*NNMAX), IW(8*NNMAX)
      INTEGER  N, NZ, IFLAG, MTYPE, LICN, LIRN
c
      INTEGER  INTNUM, LIMAX, TIPO(20), NGRAFS, VETEMP(20)
      INTEGER  IEL,NFB,NBRD,NVER,NTEL,NTNOS,NDIM,NTEMPO
c
      CHARACTER  MUC(20)*15
```

```
c
      COMMON / INFOREDE / NTNOS,NDIM,NTEL,NVER,NFB
      COMMON / LAGRANGE / IEL
      COMMON /CONVERGENCIA/ EPS,LIMAX
      COMMON / FRONTEIRA / TIPO
      COMMON /ANGULO/ TETHA
      COMMON /PASSOTEMPO/ STEP
      COMMON / GRAFICOS / NGRAFS, VETEMP, MUC
      COMMON / MATRIX / K1SUB,M1SUB,V1SUB
      COMMON / SURFACE / INTSURF
      COMMON / MASSAS / Mn, Mntil
c
c
190   WRITE(*,1)
1     FORMAT(5(/),T15,'METODO  DOS  ELEMENTOS  FINITOS',//,T15,
     *                  '  PROBLEMAS DE DIFUSAO-CONVECCAO',/,T15,
     *                  '  ------------------------------',/,T15,
     *                  '     -------------------------',/,T15,
     *                  '        ------------------',/,T15,
     *                  '           ------------',//)
c
c
c
c
c     -------------------------------     ------------------
c     Leitura dos Parametros da Malha  -  Arquivo  MALHA.DAT
c     -------------------------------     ------------------
c
      CALL LEITURA(NVER,NTEMPO,INTNUM)
c
c     ------------------------------------------------------
c     Leitura do Arquivo Contendo as Informacoes sobre a Rede
c     ------------------------------------------------------
c
      CALL LEEMALHA(C,M,INB,NFB,NBRD,NTNOS,NTEL,NDIM)
c
c     ---------------------------------------------
c     Alteracao da Rede para Discretizacao no Tempo
c              Elementos Tridimensionais
c     ---------------------------------------------
c
      NDIM = 2*NTNOS
c
      DO I = 1,NTEL
         LISTA(I) = 0
         DO J = 4,NVER
            M(J,I) = M(J-3,I) + NTNOS
         END DO
      END DO
c
      DO I = 1,NTNOS
         INB(I+NTNOS) = INB(I)
      END DO
c
c     ------------------------------
c     Tipo das Condicoes de Contorno
c     ------------------------------
c
      CALL TIPOCONTOR(INB,NBRD,NDIM)
c
c     ---------------------------------------------
c     Integracao Numerica Sobre o Triangulo Padrao
c     ---------------------------------------------
c     intnum  =  (1) Quadratic   (2) Cubic   (3) Quintic
c
      CALL PONTOS(INTNUM)
c
c     --------------------------------------
c     Pontos de Integracao de Gauss-Legendre
c     --------------------------------------
c
      CALL GAUSSQ
c
c     --------------------------------------
c     Submatrizes da Discretizacao no Tempo
c     --------------------------------------
c
      CALL SUBSTIFF(K1SUB,M1SUB,V1SUB,0.0D0,STEP)
c
c     ----------------
c     Condicao Inicial
```

```
c    ----------------
c
     MAXIMO = 0.0D0
c
     DO 5 I = 1,NTNOS
          X = C(1,I)
          Y = C(2,I)
          U(I) = 0.0D0
          U(I+NTNOS) = FONTE(X,Y)
          MAXIMO = DMAX1(MAXIMO,U(I+NTNOS))
5    CONTINUE
c
     CALL MASSA(M,C,U,INB,Mn)
c
     CALL SURF(M,C,U,INTSURF)
c
     NDIM = 0
     DO I=1,NTEL
        MAXX = -1.0D10
        MAXY = -1.0D10
        MINX = 1.0D10
        MINY = 1.0D10
        DO J=4,NVER
           X = C(1,M(J,I)-NTNOS)
           Y = C(2,M(J,I)-NTNOS)
           MAXX = DMAX1(X,MAXX)
           MAXY = DMAX1(Y,MAXY)
           MINX = DMIN1(X,MINX)
           MINY = DMIN1(Y,MINY)
           IF (U(M(J,I)) .GE. MAXIMO*1.0D-3)   LISTA(I) = 1
        END DO
        XMAX(I) = MAXX
        YMAX(I) = MAXY
        XMIN(I) = MINX
        YMIN(I) = MINY
     END DO
c
     CALL SUBDOM(M,C,CNEW,LISTA,LISTANEW,LISTAUX,VAR,
     &              XMAX,YMAX,XMIN,YMIN)
c
c    ----------------------------
c    Construcao do Sistema Linear
c    ----------------------------
c
     CALL FORMACAO(M,C,INB,F,A,IROWST,LENROW,JCOL,NELEM,U,
     &                VAR,LISTA,LISTANEW)
c
     CALL POSPROC(IROWST,LENROW,ILIN,JCOL,A,NDIM)
c
c
     WRITE(*,1000) NTNOS, NDIM, NELEM, NTEL
1000 FORMAT(//,t10,
     *  '          Numero Total de Nos da Rede    =  ',I6.6,/,T10,
     *  'Numero Total de Variaveis do Subdominio  =  ',I6.6,/,T10,
     *  'Numero Total de Elementos No Nulos      =  ',I6.6,/,T10,
     *  '          Numero Total de Elementos      =  ',I6.6,//)
c
c
c
c    ====================
c    Problema de Evolucao
c    ====================
c
     WRITE(*,390)
390  FORMAT(/,T10,'Problema de Difusao-Conveccao de Evolucao',/)
c
     KK = 1
     DO 400 K = 1,NTEMPO
c
           WRITE(*,410) K
410        FORMAT(/,T10,'Nivel de Tempo  =   ',I3.3)
c
c
           N = NDIM
           NZ = NELEM
           LICN = MAXICN
           LIRN = MAXIRN
           MTYPE = 1
           PIVO = 0.5D0
           CALL MA28AD(N,NZ,A,LICN,ILIN,LIRN,JCOL,PIVO,IKEEP,IW,W,IFLAG)
c
```

```
c
            MAXIMO = 0.0D0
            CALL MA28CD(N,A,LICN,JCOL,IKEEP,F,W,MTYPE)
            DO I = 1,NDIM
               USD(I) = F(I)
               MAXIMO = DMAX1(MAXIMO,F(I))
            END DO
c
            DO I=1,2*NTNOS
               U(I) = 0.0d0
               IF (VAR(I) .NE. 0) U(I) = USD(VAR(I))
            END DO
c
            CALL VOLUME(M,C,U,SOLIDO(K))
c
            CALL MASSA(M,C,U,INB,MO)
            Mntil = SOLIDO(K)
            Mn = Mn + MO
c
            CALl SURF(M,C,U,INTSURF)
c
c
c    ------------------------------------------
c    Gera um Arquivo de Dados para a Saida Grafica
c    ------------------------------------------
c
            IF(VETEMP(KK).EQ.K)  THEN
             CALL GRAFICO(U,M,C,MUC(KK))
             KK = KK + 1
            END IF
c
            DO I=1,NTEL
               LISTA(I) = 0
               DO J=4,NVER
                  IF (U(M(J,I)) .GE. 1.0D-3*MAXIMO) THEN
                   LISTA(I) = 1
                   EXIT
                   END IF
               END DO
            END DO
c
            CALL SUBDOM(M,C,CNEW,LISTA,LISTANEW,LISTAUX,VAR,
            &              XMAX,YMAX,XMIN,YMIN)
c
            CALL FORMACAO(M,C,INB,F,A,IROWST,LENROW,JCOL,NELEM,U
            &               VAR,LISTA,LISTANEW)
c
            CALL POSPROC(IROWST,LENROW,ILIN,JCOL,A,NDIM)
c
400  CONTINUE
c
c
     OPEN(UNIT=200, FILE='SOLIDO.M', STATUS='UNKNOWN')
     DO I = 1,NTEMPO
        WRITE(200,210) I, SOLIDO(I)
210     FORMAT(T10,I4.4,T20,F15.9)
     END DO
     CLOSE(UNIT=200)
c
c
c
    END
c---------------------------------------------------------------------
c---------------------------------------------------------------------
```

Subrotina: Leitura
Descrição: Leitura dos dados de entrada do problema.

```
c-----------------------------------------------------------------
c              FILE                        LEITURA.FOR
c-----------------------------------------------------------------
c
      SUBROUTINE LEITURA(NVER,NTEMPO,INTNUM)
c     ====================================
c
      REAL*8 STEP, ALPHA, CONST
c
      INTEGER NVER, NTEMPO, IEL, LIMAX, INTNUM
      INTEGER VETEMP(20), NGRAFS
c
      CHARACTER TEXTO*72, NAME*30, MUC(20)*15
c
      COMMON /PASSOTEMPO/ STEP
      COMMON / LAGRANGE / IEL
      COMMON / DIFUSION / ALPHA
      COMMON / ARQUIVO / NAME
      COMMON / PETROV /  CONST
      COMMON / GRAFICOS / NGRAFS, VETEMP, MUC
c
c
c     -------------------------------
c     Leitura dos Parametros da Malha
c     -------------------------------
c
      OPEN(UNIT = 100, FILE = 'MALHA.DAT', STATUS = 'OLD')
c
      READ(100,2) TEXTO
2     FORMAT(a72)
      READ(100,*) NVER
      READ(100,2) TEXTO
      READ(100,*) IEL
      READ(100,2) TEXTO
      READ(100,*) INTNUM
      READ(100,2) TEXTO
      READ(100,*) STEP, NTEMPO
      READ(100,2) TEXTO
      READ(100,*) ALPHA
      READ(100,2) TEXTO
      READ(100,*) CONST
      READ(100,2) TEXTO
      READ(100,3) NAME
3     FORMAT(8X,a30)
c
      CLOSE(UNIT=100)
c
c
c
      OPEN(UNIT = 200, FILE = 'SAIGRAF.DAT', STATUS = 'OLD')
c
      READ(200,2) TEXTO
      READ(200,*) NGRAFS, ( VETEMP(I) , I = 1,NGRAFS )
      DO I = 1,NGRAFS
         READ(200,4) MUC(I)
4        FORMAT(8X,A15)
      END DO
c
      CLOSE(UNIT=200)
c
c
c
      RETURN
      END
c-----------------------------------------------------------------
c-----------------------------------------------------------------
```

Subrotina: Substiff

Descrição: Montagem das submatrizes relativas aos elementos unidimensionais da variável temporal.

```
C-----------------------------------------------------------------
C             FILE                          SUBSTIFF.FOR
C-----------------------------------------------------------------
C
      SUBROUTINE SUBSTIFF(KSUB,MSUB,VSUB,XMIN,XMAX)
C     ============================================
C
      REAL*8 PHI(2),DPHI(2),MSUB(2,2),VSUB(2,2),KSUB(2,2)
      REAL*8 T(4),W(4),JACOB,XX,XMIN,XMAX
C
      INTEGER I,J,K,NPI
C
      COMMON / GAUSSPOINTS / W,T,NPI
C
C
C
C
      DO I = 1,2
      DO J = 1,2
C
      KSUB(I,J) = 0.0D0
      MSUB(I,J) = 0.0D0
      VSUB(I,J) = 0.0D0
C
      END DO
      END DO
C
      JACOB = ( XMAX - XMIN )*5.0D-1
C
      DO 30 K = 1,NPI
C
      CALL TRANSF1(T(K),XMIN,XMAX,XX,1)
      CALL BASES1(T(K),PHI,DPHI)
C
      DO 20 I = 1,2
      DO 10 J = 1,2
C
      KSUB(I,J) = KSUB(I,J) + W(K)*PHI(I)*PHI(J)*JACOB
      MSUB(I,J) = MSUB(I,J) + W(K)*DPHI(I)*DPHI(J)/JACOB
      VSUB(I,J) = VSUB(I,J) + W(K)*PHI(I)*DPHI(J)
C
10    CONTINUE
20    CONTINUE
C
30    CONTINUE
C
C
      RETURN
      END
C
C-----------------------------------------------------------------
C-----------------------------------------------------------------
```

Subrotina: Formacao

Descrição: Formação do sistema linear resultante das discretizações no tempo e no espaço, a cada faixa de tempo.

```
C-----------------------------------------------------------------
C           FILE                            FORMACAO.FOR
C-----------------------------------------------------------------
C
      SUBROUTINE FORMACAO(M,C,INB,F,A,IROWST,LENROW,JCOL,NELEM,U,
     &                    VAR,LISTA,LISTANEW)
C     ===========================================================
C
      REAL*8 C(2,*),F(*),SUBMAT(6,6),A(*),SOMA,U(*),OSUB(6,6)
      REAL*8 MSUB(6,6),KSUB(6,6),VSUB(6,6),PSUB(6,6),LSUB(6,6)
      REAL*8 NSUB(6,6),QSUB(6,6),K1SUB(2,2),M1SUB(2,2),V1SUB(2,2)
      REAL*8 RSUB(6,6),KKSUB(6,6),OOSUB(6,6),RRSUB(6,6),INTSURF
C
      INTEGER    M(6,*), INB(*), NELEM
      INTEGER    IROWST(*), JCOL(*), LENROW(*)
      INTEGER    IM, JM, LINHA, VAR(*), LISTA(*), LISTANEW(*)
      INTEGER    NTEL,NVER,NTNOS,NFB,NDIM,K, KIJ
C
      COMMON / INFOREDE / NTNOS,NDIM,NTEL,NVER,NFB
      COMMON / MATRIX / K1SUB,M1SUB,V1SUB
      COMMON /SURFACE/ INTSURF
C
C
C
C     ------------------------------
C     Variaveis de Entrada  e  Saida
C     ------------------------------
C
C          M  :  Matriz de Incidencia
C
C          C  :  Matriz de Coordenadas dos Nos
C
C          F  :  Vetor de Carga
C
C     SUBMAT  :  SubMatrizes
C
C
C
C     ------------------------------
C     CONSTRUCAO  DO VETOR DE CARGA
C     ------------------------------
C
C
      CALL ZEROS(A,F,IROWST,LENROW,JCOL,NDIM,LINHA)
C
C
      CALL EVOLUCAO(M,C,U,F,VAR,LISTA,LISTANEW)
C
c     ------------------------------------------------
c     Condicao de Contorno Essencial no Vetor de Carga
c     ------------------------------------------------
      CALL BOUNDVETOR(M,C,INB,F,VAR,LISTA,LISTANEW)
c
c
      NELEM = 0
      DO 200 K = 1,NTEL
C
      IF ((LISTANEW(K) .EQ. 1) .OR. (LISTA(K) .EQ. 1)) THEN
C
      CALL ZERASUB(MSUB,KSUB,VSUB,PSUB,LSUB,NSUB,QSUB,OSUB,RSUB,
     &                KKSUB,OOSUB,RRSUB,NFB)
C
      CALL SUBMATRIX(M,C,KSUB,MSUB,VSUB,PSUB,LSUB,NSUB,QSUB,OSUB,
     &                 RSUB,KKSUB,OOSUB,RRSUB,K,U)
C
      DO  I = 1,NVER
C
      CALL PROCURA(I,I1,I2)
C
      DO  J = 1,NVER
C
      CALL PROCURA(J,J1,J2)
C
```

```
      SOMA = KSUB(I1,J1)*V1SUB(I2,J2) + VSUB(I1,J1)*K1SUB(I2,J2)
      SOMA = SOMA + OSUB(I1,J1)*M1SUB(I2,J2) + RSUB(I1,J1)*V1SUB(I2,J2)
      SOMA = SOMA + QSUB(I1,J1)*V1SUB(J2,I2) + NSUB(I1,J1)*K1SUB(I2,J2)
      SOMA = SOMA + LSUB(J1,I1)*V1SUB(J2,I2) + PSUB(I1,J1)*K1SUB(I2,J2)
      SOMA = SOMA + MSUB(I1,J1)*K1SUB(I2,J2)
      SOMA = SOMA + INTSURF*KKSUB(I1,J1)*K1SUB(I2,J2)
      SOMA = SOMA + INTSURF*RRSUB(I1,J1)*K1SUB(I2,J2)
      SOMA = SOMA + INTSURF*OOSUB(I1,J1)*V1SUB(J2,I2)
      SUBMAT(I,J) = SOMA
C
      END DO
      END DO
C
      DO I = 1,NFB
      DO J = 1,NFB
C
      SUBMAT(I,J) = SUBMAT(I,J) + KSUB(I,J)
C
      END DO
      END DO
c
c
c     ----------------------------------------
c     Condicao de Contorno Essencial na SubMatriz
c     ----------------------------------------
      CALL ESSENCIAL(M,SUBMAT,INB,K)
c
      DO 120 I = 1,NVER
      IM = VAR(M(I,K))
      DO 110 J = 1,NVER
      JM = VAR(M(J,K))
C
      IF(SUBMAT(I,J).EQ.0.0D0) GO TO 110
      CALL FASTFIND(IROWST,LENROW,JCOL,NELEM,IM,JM,KIJ,LINHA)
C
      A(KIJ) = A(KIJ) + SUBMAT(I,J)
C
110   CONTINUE
120   CONTINUE
C
C
      END IF
C
200   CONTINUE
C
C
      RETURN
      END
C---------------------------------------------------------------------
C---------------------------------------------------------------------
```

Subrotina: Submatrix

Descrição: Montagem das submatrizes relativas aos elementos bidimensionais da variável espacial.

```
C-----------------------------------------------------------------
C            FILE                          SUBMATRIX.FOR
C-----------------------------------------------------------------
C
      SUBROUTINE  SUBMATRIX(M,C,KSUB,MSUB,VSUB,PSUB,LSUB,NSUB,QSUB,
     &                          OSUB,RSUB,KKSUB,OOSUB,RRSUB,K,U)
C     ===============================================================
C
      REAL*8 KSUB(6,*), MSUB(6,*), NSUB(6,*), VSUB(6,*), PSUB(6,*)
      REAL*8 W(7),T(2,7), C(2,*), LSUB(6,*), QSUB(6,*), U(*),FUNCAO
      REAL*8 PHI(6), GRAD(2,6), B(2,2), BTINV(2,2), SCALAR(6,6)
      REAL*8 XI,ETA,JACOB,X,Y,ALFA,BETA(2),BETAGRAD(6),OSUB(6,6)
      REAL*8 GRADREAL(2,6), AXY, DELTA, DIFUS, MODBETA, CONST
      REAL*8 GRADALFA(2),GALFA(6),RSUB(6,*),BETA2(2),BETAGRAD2(6)
      REAL*8 KKSUB(6,*), OOSUB(6,*), RRSUB(6,*), SIGMA
C
      INTEGER   M(6,*)
      INTEGER   IEL,NTNOS,NTEL,NVER,NPI,K,NFB,NDIM
C
      COMMON / INTEGRACAO / T, W, NPI
      COMMON / LAGRANGE / IEL
      COMMON / INFOREDE / NTNOS,NDIM,NTEL,NVER,NFB
      COMMON / PETROV /  CONST
C
C
C
C     --------------------------
C     Construcao das SubMatrizes
C     --------------------------
C
C
      DO 200 L = 1,NPI
C
       XI = T(1,L)
      ETA = T(2,L)
C
      CALL BASES(PHI,IEL,XI,ETA)
      CALL GRADIENTE(GRAD,IEL,XI,ETA)
C
C     ------------------------------
C     Calculo da Transformacao Afim
C     ------------------------------
C
      CALL TRANSF(M,C,K,XI,ETA,B,BTINV,JACOB,X,Y,1)
C
C     ----------------------------------
C     Produto Escalar entre os Gradiente
C     ----------------------------------
C
      CALL ESCALAR(GRAD,GRADREAL,BTINV,NFB,SCALAR)
C
C
      CALL CONVECCAO(X,Y,BETA)
      CALL TRANSPORTE(GRAD,BETA,BTINV,NFB,BETAGRAD)
C
      GRADALFA(1) = 0.0D0
      GRADALFA(2) = 0.0D0
      FUNCAO = 0.0D0
      DO 50 J = 1,NFB
      FUNCAO  =  FUNCAO  +  U(M(J+NFB,K))*PHI(J)
      GRADALFA(1) = GRADALFA(1) + U(M(J+NFB,K))*GRADREAL(1,J)
      GRADALFA(2) = GRADALFA(2) + U(M(J+NFB,K))*GRADREAL(2,J)
50    CONTINUE
C
      GRADALFA(1) = 6.0D0*FUNCAO*GRADALFA(1)
      GRADALFA(2) = 6.0D0*FUNCAO*GRADALFA(2)
      FUNCAO = 3.0D0*FUNCAO*FUNCAO
C
      DO J = 1,NFB
      GALFA(J) = 0.0D0
      DO I = 1,2
      GALFA(J) = GALFA(J) + GRADALFA(I)*GRADREAL(I,J)
      END DO
```

```
      END DO
c
      AXY =  ALFA(X,Y)
      BETA2(1) = BETA(1) - AXY*GRADALFA(1)
      BETA2(2) = BETA(2) - AXY*GRADALFA(2)
C
      DO I = 1,NFB
      BETAGRAD2(I) = GRADREAL(1,I)*BETA2(1) + GRADREAL(2,I)*BETA2(2)
      END DO
C
      difus  =  AXY*FUNCAO
      MODBETA  =  DSQRT( BETA(1)*BETA(1)  + BETA(2)*BETA(2) + 1.0D0 )
      DELTA   =  DSQRT(0.5d0*JACOB) - DIFUS / MODBETA
      DELTA   =  ( DMAX1( DELTA, 0.0d0 ) / MODBETA )*CONST
C
C     ----------------------------
C     Construcao  das  Submatrizes
C     ----------------------------
C
      DO 25 I = 1,NFB
      DO 15 J = 1,NFB
C
      KSUB(I,J) = KSUB(I,J) + PHI(I)*PHI(J)*JACOB*W(L)
C
      OSUB(I,J) = OSUB(I,J) + PHI(I)*PHI(J)*JACOB*W(L)*DELTA
C
      MSUB(I,J) = MSUB(I,J) + SCALAR(I,J)*difus*JACOB*W(L)
C
      VSUB(I,J) = VSUB(I,J) + BETAGRAD(J)*PHI(I)*JACOB*W(L)
C
      PSUB(I,J) = PSUB(I,J) + BETAGRAD(J)*BETAGRAD2(I)*JACOB*W(L)*DELTA
C
      NSUB(I,J) = NSUB(I,J)-GALFA(J)*AXY*BETAGRAD2(I)*JACOB*W(L)*DELTA
C
      LSUB(I,J) = LSUB(I,J) + BETAGRAD(I)*PHI(J)*JACOB*W(L)*DELTA
C
      RSUB(I,J) = RSUB(I,J) + BETAGRAD2(I)*PHI(J)*JACOB*W(L)*DELTA
C
      QSUB(I,J) = QSUB(I,J) - GALFA(J)*AXY*PHI(I)*JACOB*W(L)*DELTA
C
      KKSUB(I,J) = KKSUB(I,J) + SIGMA(X,Y)*PHI(I)*PHI(J)*JACOB*W(L)
C
      RRSUB(I,J) = RRSUB(I,J)
     &             + SIGMA(X,Y)*BETAGRAD2(I)*PHI(J)*JACOB*W(L)*DELTA
C
      OOSUB(I,J) = OOSUB(I,J)
     &             + SIGMA(X,Y)*PHI(I)*PHI(J)*JACOB*W(L)*DELTA
C
15    CONTINUE
25    CONTINUE
C
C
200   CONTINUE
C
C
C
      RETURN
      END
C---------------------------------------------------------------
C---------------------------------------------------------------
```

Subrotina: Evolucao

Descrição: Montagem do vetor do lado direito do sistema linear resultante, a cada faixa de tempo.

```
C-----------------------------------------------------------------
C            FILE                            EVOLUCAO.FOR
C-----------------------------------------------------------------
C
      SUBROUTINE  EVOLUCAO(M,C,U,F,VAR,LISTA,LISTANEW)
C     ================================================
C
      REAL*8 U(*), C(2,*), F(*)
      REAL*8 PHI(6), FSUB(6), Mn, Mntil
      REAL*8 W(7),T(2,7),B(2,2),BTINV(2,2)
      REAL*8 XI,ETA,JACOB,FUNCAO,X,Y
C
      INTEGER   M(6,*), VAR(*), LISTA(*), LISTANEW(*)
      INTEGER IEL,NDIM,NTNOS,NTEL,NVER,NPI,K,NFB
C
      COMMON / INTEGRACAO / T,W,NPI
      COMMON / LAGRANGE / IEL
      COMMON / INFOREDE / NTNOS,NDIM,NTEL,NVER,NFB
      COMMON / MASSAS / Mn, Mntil
C
C
C
      DO J = 1,NDIM
      F(J) = 0.0D0
      END DO
C
      DO 600 K = 1,NTEL
C
      IF ((LISTANEW(K) .EQ. 1) .OR. (LISTA(K) .EQ. 1)) THEN
C
      DO I = 1,NVER
      FSUB(I) = 0.0D0
      END DO
C
      DO 200 I = 1,NPI
C
      XI = T(1,I)
      ETA = T(2,I)
C
      CALL BASES(PHI,IEL,XI,ETA)
C
C
C     ----------------------------
C     Calculo da Transformacao Afim
C     ----------------------------
C
      CALL TRANSF(M,C,K,XI,ETA,B,BTINV,JACOB,X,Y,1)
C
C
      FUNCAO = 0.0D0
      DO 55 J = 1,NFB
      FUNCAO  =  FUNCAO  +  U(M(J+NFB,K))*PHI(J)
55    CONTINUE
C
C     -----------------
C     Controle da Massa
C     -----------------
C
      FUNCAO = FUNCAO - 5.0D-3*(Mntil - Mn)*DMAX1(FUNCAO,0.0D0)
C
      DO J = 1,NFB
      FSUB(J) = FSUB(J)  +  FUNCAO*PHI(J)*W(I)*JACOB
      END DO
C
C
200   CONTINUE
C
C
      DO J = 1,NVER
      F(VAR(M(J,K))) = F(VAR(M(J,K))) + FSUB(J)
      END DO
C
C
```

```
     END IF
C
600  CONTINUE
C
C
C
     RETURN
    END
C-----------------------------------------------------------------
C-----------------------------------------------------------------
```

Subrotina: Subdom

Descrição: Busca do subdomínio que contém a mancha num determinado nível de tempo e previsão do novo subdomínio para o próximo nível de tempo.

```
c-----------------------------------------------------------------
c             FILE                        SUBDOM.FOR
c-----------------------------------------------------------------
c
      SUBROUTINE SUBDOM(M,C,CNEW,LISTA,LISTANEW,LISTAUX,VAR,
     &                  XMAX,YMAX,XMIN,YMIN)
c
c
      REAL*8 C(2,*), CNEW(2,*), BETA(2), STEP
      REAL*8 X, Y, XI, ETA, B(2,2), BTINV(2,2), JACOB
      REAL*8 XMAX(*), YMAX(*), XMIN(*), YMIN(*)
c
      INTEGER LISTA(*), LISTANEW(*), LISTAUX(*), NTEL, NFB
      INTEGER M(6,*), NDIM, NTNOS, NVER, VAR(*)
c
      COMMON / INFOREDE / NTNOS,NDIM,NTEL,NVER,NFB
      COMMON /PASSOTEMPO/ STEP
c
      DO I=1,2*NTNOS
       VAR(I) = 0
      END DO
c
c       Busca p/ tras
c
      DO I=1,NTEL
        LISTANEW(I) = 0
        LISTAUX(I) = 0
        DO J=1,NFB
         X = C(1,M(J,I))
         Y = C(2,M(J,I))
         CALL CONVECCAO(X,Y,BETA)
         CNEW(1,M(J,I)) = X - BETA(1)*STEP
         CNEW(2,M(J,I)) = Y - BETA(2)*STEP
        END DO
      END DO
c
      DO I=1,NTEL
        DO J=1,NFB
         X = CNEW(1,M(J,I))
         Y = CNEW(2,M(J,I))
         DO K=1,NTEL
          IF (LISTA(K) .EQ. 1) THEN
           IF (X .LE. XMAX(K)) THEN
            IF (X .GE. XMIN(K)) THEN
             IF (Y .LE. YMAX(K)) THEN
              IF (Y .GE. YMIN(K)) THEN
               CALL TRANSF(M,C,K,XI,ETA,B,BTINV,JACOB,X,Y,-1)
               IF (((XI + ETA) .LE. 1.0D0) .AND. (XI .GE. 0.0D0) .AND.
     &          (ETA .GE. 0.0D0)) THEN
                LISTANEW(I) = 1
                EXIT
               END IF
              END IF
             END IF
            END IF
           END IF
          END IF
         END DO
        END DO
      END DO
```

```
c
      DO I=1,NTEL
        IF (LISTANEW(I) .EQ. 1) THEN
         DO J=1,NFB
          DO K=1,NTEL
           IF (LISTANEW(K) .EQ. 0) THEN
            DO L=1,NFB
             IF (M(J,I) .EQ. M(L,K)) THEN
              LISTAUX(K) = 1
              EXIT
             END IF
            END DO
           END IF
          END DO
         END DO
        END IF
       END DO
c
      DO I=1,NTEL
c
        IF (LISTAUX(I) .EQ. 1) LISTANEW(I) = 1
c
        IF (LISTA(I) .EQ. 1) THEN
         DO J=1,NVER
          VAR(M(J,I)) = 1
         END DO
        END IF
c
        IF (LISTANEW(I) .EQ. 1) THEN
         DO J=1,NVER
          VAR(M(J,I)) = 1
         END DO
        END IF
c
      END DO
c
      NDIM = 0
      DO I=1,2*NTNOS
        IF (VAR(I) .EQ. 1) THEN
         NDIM = NDIM + 1
         VAR(I) = NDIM
        END IF
      END DO
c
      RETURN
      END
c
c---------------------------------------------------------------
c---------------------------------------------------------------
```

Subrotina: Pontos

Descrição: Pontos e pesos para a regra de integração numérica de Gauss, para um triângulo.

```
c---------------------------------------------------------------
c           FILE                        PONTOS.FOR
c---------------------------------------------------------------
c
      subroutine  pontos(iflag)
c     ========================
c
      real*8 w(7), t(2,7), alfa(2), beta(2)
c
      integer iflag, npi
c
      common /integracao/ t, w, npi
c
c
c
c     ---------------------------------
c     Integral sobre o Triangulo Padrao
c     ---------------------------------
c
c     -----------------------------------------------------
c     Pontos de Integracao  e  Pesos da Regra de Quadratura
```

```
c     ------------------------------------------------------
c
c
c     -------------------
c     Variavel de Entrada
c     -------------------
c
c
c     iflag =     (1)  Quadratic     (2)  Cubic     (3)  Quintic
c
c
c     --------------------------------------
c     Variaveis de Saida ( comando common )
c     --------------------------------------
c
c     t : Pontos de Integracao
c
c     w : Pesos da Regra de Quadratura
c
c    npi : Numero de Pontos de Integracao
c
c
c
c
      if( iflag.eq.1 ) then
c
c
      w(1) = 1.0d0/6.0d0
      w(2) = w(1)
      w(3) = w(1)
c
c
      t(1,1) =  1.0d0/6.0d0
      t(2,1) =  1.0d0/6.0d0
c
      t(1,2) =  2.0d0/3.0d0
      t(2,2) =  1.0d0/6.0d0
c
      t(1,3) =  1.0d0/6.0d0
      t(2,3) =  2.0d0/3.0d0
c
c
      npi = 3
c
      else if( iflag.eq.2 ) then
c
      w(1) = -27.0d0/96.0d0
      w(2) =  25.0d0/96.0d0
      w(3) =  w(2)
      w(4) =  w(2)
c
c
      t(1,1) =  1.0d0/3.0d0
      t(2,1) =  1.0d0/3.0d0
c
      t(1,2) =  0.2d0
      t(2,2) =  0.6d0
c
      t(1,3) =  0.2d0
      t(2,3) =  0.2d0
c
      t(1,4) =  0.6d0
      t(2,4) =  0.2d0
c
c
      npi = 4
c
      else if( iflag.eq.3 )  then
c
      alfa(1) = ( 9.0d0 - 2.0d0*dsqrt(15.0d0) ) / 21.0d0
      alfa(2) = ( 9.0d0 + 2.0d0*dsqrt(15.0d0) ) / 21.0d0
      beta(1) = ( 6.0d0 + dsqrt(15.0d0) ) / 21.0d0
      beta(2) = ( 6.0d0 - dsqrt(15.0d0) ) / 21.0d0
c
c
      w(1) =  9.0d0/80.0d0
      w(2) =  ( 155.0d0 + dsqrt(15.0d0) ) / 2400.0d0
      w(3) =  w(2)
      w(4) =  w(2)
      w(5) =  ( 155.0d0 - dsqrt(15.0d0) ) / 2400.0d0
      w(6) =  w(5)
```

```
      w(7) =  w(5)
c
c
      t(1,1) = 1.0d0/3.0d0
      t(2,1) = 1.0d0/3.0d0
c
c
      t(1,2) = beta(1)
      t(2,2) = beta(1)
c
      t(1,3) = alfa(1)
      t(2,3) = beta(1)
c
      t(1,4) = beta(1)
      t(2,4) = alfa(1)
c
      t(1,5) = beta(2)
      t(2,5) = beta(2)
c
      t(1,6) = alfa(2)
      t(2,6) = beta(2)
c
      t(1,7) = beta(2)
      t(2,7) = alfa(2)
c
c
      npi = 7
c
c
      end if
c
c
c
      return
      end
c---------------------------------------------------------------------
c---------------------------------------------------------------------
```

Subrotina: Gaussq

Descrição: Pontos e pesos para a regra de integração numérica de Gauss, para um intervalo real.

```
c---------------------------------------------------------------------
c                FILE                       GAUSSQ.FOR
c---------------------------------------------------------------------

      SUBROUTINE  GAUSSQ
C     ==================
C
      REAL*8 W(4),T(4)
C
      INTEGER  NPI
C
      COMMON / GAUSSPOINTS / W, T, NPI
C
C
C
C
C     -------------------------
C     NUMERO DE PONTOS DE GAUSS
C     -------------------------
C
      NPI = 4
C
C     ------------------------------
C     ZEROS DO POLINOMIO DE LEGENDRE
C     ------------------------------
C
      T(1) =  0.861136311594052575 22D0
      T(2) =  0.339981043584856264 80D0
      T(3) = -T(2)
      T(4) = -T(1)
C
C     ------------------------------------------------
C     PESOS DA FORMULA DA QUADRATURA  GAUSS-LEGENDRE
```

```
C     ---------------------------------------------
C
      W(1) = 0.347854845137453857370
      W(2) = 0.652145154862546142620
      W(3) = W(2)
      W(4) = W(1)
C
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
```

Subrotina: Tipocontor

Descrição: Tradução das condições de contorno impostas em cada lado da fronteira do domínio.

```
c-----------------------------------------------------------------------
c               FILE                        TIPOCONTOR.FOR
c-----------------------------------------------------------------------
c
      subroutine tipocontor(INB,nbrd,ndim)
c     ====================================
c
      integer tipo(20), INB(*), nbrd, ndim
c
      common / fronteira / tipo
c
c     ------------------------------
c     Tipo das Condicoes de Contorno
c     ------------------------------
c
      write(*,210)
210   format(2(/),T10,'Tipo das Condicoes de Contorno',/)
      write(*,220)
220   format(/,T10,'(0)  Neumann Nula             (1)  Dirichlet',/)
c
c
      do i = 1,nbrd
c
      write(*,230) i
230   format(/,T10,'Curva com Referencia =  ',i2.2,T40,
     *  'Condicao de Contorno ===> ',$)
      read(*,*) tipo(i)
c
      if (tipo(i) .eq. 1)  tipo(i) = i
c
      end do
c
c
c     ------------------------------------------------
c     Condicoes de Contorno - Informacoes no Vetor  INB
c     ------------------------------------------------
c
      do  i = 1,ndim
      if(INB(i).ne.0)  INB(i) = tipo(INB(i))
      end do
c
c
c
      return
      end
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
```

Subrotina: Grafico
Descrição: Saída em formato gráfico do Matlab.

```
C-----------------------------------------------------------------------
C            FILE                                GRAFICO.FOR
C-----------------------------------------------------------------------
C
      SUBROUTINE GRAFICO(U,M,C,NAME)
C     ==============================
C
      REAL*8 C(2,*),U(*)
C
      INTEGER   M(6,*)
      INTEGER   NTEL,NVER,NTNOS,NFB,NSUBD,NDIM,IEL
C
      CHARACTER NAME*15
C
      COMMON / INFOREDE / NTNOS,NDIM,NTEL,NVER,NFB
      COMMON / LAGRANGE / IEL
C
C
C     ------------------------------------------------
C     GERA UM ARQUIVO PARA TRACAR O GRAFICO DA FUNCAO  U
C     ------------------------------------------------
C
C
      OPEN(UNIT = 150, FILE = NAME, STATUS = 'UNKNOWN')
      GO TO 285
C
      NSUBD = 1
      WRITE(150,90)
90    FORMAT('M=[')
      DO 10 J = 1,NTEL
C
      WRITE(150,100) ( M(I,J) , I = 1,NVER ), NSUBD
100   FORMAT(T5,6(I5,3X),3X,I1)
C
10    CONTINUE
      WRITE(150,110)
110   FORMAT('];',//)
C
C
      WRITE(150,190)
190   FORMAT('C=[')
      DO 20 I = 1,NTNOS
C
      WRITE(150,200) C(1,I),C(2,I)
200   FORMAT(T5,D17.9,T35,D17.9)
C
20    CONTINUE
      WRITE(150,210)
210   FORMAT('];',//)
C
C
285   WRITE(150,290)
290   FORMAT('U=[')
      DO 30 I = 1,NTNOS
C
      WRITE(150,300) U(I+NTNOS)
300   FORMAT(T5,D17.9)
C
30    CONTINUE
      WRITE(150,310)
310   FORMAT('];',//)
C
C
      CLOSE(UNIT=150)
C
      WRITE(*,600) NAME
600   FORMAT(//,T15,'Saida Grafica no Arquivo        ', A15,/,T15,
     *               '---------------------------------------',/)
C
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
```

Subrotina: Volume
Descrição: Cálculo do volume da solução, a cada nível de tempo.

```
C-----------------------------------------------------------------------
C                FILE                          VOLUME.FOR
C-----------------------------------------------------------------------
C
      SUBROUTINE  VOLUME(M,C,U,SOLIDO)
C     ================================
C
      REAL*8 U(*), C(2,*), SOLIDO
      REAL*8 PHI(6), B(2,2), BTINV(2,2)
      REAL*8 W(7), T(2,7)
      REAL*8 XI, ETA, JACOB, FUNCAO, SOMA
C
      INTEGER M(6,*)
      INTEGER IEL,NTNOS,NDIM,NTEL,NVER,NPI,NFB
C
      COMMON / INTEGRACAO / T,W,NPI
      COMMON / LAGRANGE / IEL
      COMMON / INFOREDE / NTNOS,NDIM,NTEL,NVER,NFB
C
C
C
C     ---------------------------------------
C     CALCULO DO VOLUME DEFINIDO PELA SOLUCAO
C     ---------------------------------------
C
C
      SOLIDO = 0.0D0
      DO 600 K = 1,NTEL
C
      SOMA = 0.0D0
      DO 200 I = 1,NPI
C
       XI = T(1,I)
      ETA = T(2,I)
C
      CALL BASES(PHI,IEL,XI,ETA)
C
      CALL TRANSF(M,C,K,XI,ETA,B,BTINV,JACOB,X,Y,1)
C
      FUNCAO = 0.0D0
      DO 50 J = 1,NFB
C
      FUNCAO  =  FUNCAO  +  U(M(J+NFB,K))*PHI(J)
C
50    CONTINUE
C
      SOMA = SOMA  +  FUNCAO*W(I)*JACOB
C
200   CONTINUE
C
      SOLIDO  =   SOLIDO  +  SOMA
C
C
600   CONTINUE
C
C
      WRITE(*,700) SOLIDO
700   FORMAT(/,T10,'Volume Definido pela Solucao  U  =  ',f15.9)
C
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
```

Subrotina: Surf
Descrição: Cálculo da área definida pela superfície da solução, a cada nível de tempo.

```
C-----------------------------------------------------------------------
C               FILE                                SURF.FOR
C-----------------------------------------------------------------------
C
      SUBROUTINE  SURF(M,C,U,INTSURF)
C     ===============================
C
      REAL*8 U(*),C(2,*), INTSURF
      REAL*8 UX, UY, DELTA, GRDX, GRDY
      REAL*8 PHI(6),GRAD(2,6),B(2,2),BTINV(2,2)
      REAL*8 W(7),T(2,7)
      REAL*8 XI,ETA,JACOB,FUNCAO,SUM,MAXIMO
C
      INTEGER M(6,*)
      INTEGER IEL,NTNOS,NTEL,NVER,NPI,NEL,NFB
C
      COMMON / INTEGRACAO / T,W,NPI
      COMMON / LAGRANGE / IEL
      COMMON / INFOREDE / NTNOS,NDIM,NTEL,NVER,NFB
C
C
C
C
C     ------------------------------------
C     CALCULO DA AREA DA SUPERFICIE U(x,y)
C     ------------------------------------
C
C
C
      MAXIMO = 0.0D0
      DO I = 1,NTNOS
      MAXIMO = DMAX1(MAXIMO,U(I))
      END DO
      MAXIMO = 1.0D-2*MAXIMO
C
      INTSURF = 0.0D0
      DO 600 NEL = 1,NTEL
C
      SUM = 0.0D0
      DO 200 I = 1,NPI
C
       XI = T(1,I)
      ETA = T(2,I)
C
      CALL BASES(PHI,IEL,XI,ETA)
      CALL GRADIENTE(GRAD,IEL,XI,ETA)
C
      CALL TRANSF(M,C,NEL,XI,ETA,B,BTINV,JACOB,X,Y,1)
C
      FUNCAO = 0.0D0
          UX = 0.0D0
          UY = 0.0D0
C
      DO 50 K = 1,NFB
C
        GRDX  =  BTINV(1,1)*GRAD(1,K)  +  BTINV(1,2)*GRAD(2,K)
        GRDY  =  BTINV(2,1)*GRAD(1,K)  +  BTINV(2,2)*GRAD(2,K)
C
      FUNCAO  =  FUNCAO  +  U(M(K,NEL))*PHI(K)
          UX  =      UX  +  U(M(K,NEL))*GRDX
          UY  =      UY  +  U(M(K,NEL))*GRDY
C
50    CONTINUE
C
C
      IF (FUNCAO .GT. MAXIMO) THEN
      DELTA  =  DSQRT( UX*UX + UY*UY + 1.0D0 )
      ELSE
      DELTA = 0.0D0
      ENDIF
C
      SUM =  SUM  +   DELTA*W(I)*JACOB
C
C
200   CONTINUE
```

```
C
C
      INTSURF  =  INTSURF  +  SUM
C
C
600   CONTINUE
C
C
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
```

Subrotina: Massa
Descrição: Cálculo da integral no bordo do domínio.

```
C-----------------------------------------------------------------------
C             FILE                          MASSA.FOR
C-----------------------------------------------------------------------
C
      SUBROUTINE  MASSA(M,C,U,INB,SOLIDO)
C     ===================================
C
      REAL*8 U(*), C(2,*), XII, YII, XJ, YJ
      REAL*8 PHI(6), B(2,2), BTINV(2,2), GRAD(2,6),SOLIDO
      REAL*8 W(4), T(4), BETA(2), BETANORMAL, DNORMAL, DS
      REAL*8 XI, ETA, X, Y, JACOB, FUNCAO, SOMA
      REAL*8 GRD(2), GRADREAL(2,6), NORMAL(2), STEP
C
      INTEGER M(6,*),IM,JM,ILD(4),NUMVER(4),INB(*)
      INTEGER IEL,NTNOS,NDIM,NTEL,NVER,NPI,NFB,IFLAG
C
      COMMON / GAUSSPOINTS / W,T,NPI
      COMMON / LAGRANGE / IEL
      COMMON / INFOREDE / NTNOS,NDIM,NTEL,NVER,NFB
      COMMON /PASSOTEMPO/ STEP
C
C
C
C     ------------------------------------------------
C     CALCULO DA VARICAO DO VOLUME DEFINIDO PELA SOLUCAO
C     ------------------------------------------------
C
C
      SOLIDO = 0.0D0
C
      DO 600 K = 1,NTEL
C
      SOMA = 0.0D0
C
      DO I = 1,NVER
      ILD(I) = INB(M(I,K))
      NUMVER(I) = I
      END DO
      ILD(NVER+1) = INB(M(1,K))
      NUMVER(NVER+1) = 1
C
C     --------------------------------------
C     Verifica se o Elemento esta' no bordo
C     --------------------------------------
C
      DO I = 1,NVER
C
      IF ( (ILD(I)*ILD(I+1) ) .NE. 0 ) THEN
      IM = NUMVER(I)
      JM = NUMVER(I+1)
      GOTO 800
      END IF
C
      END DO
      GOTO 600
C
C
800   XII = C(1,M(IM,K))
```

```
      YII = C(2,M(IM,K))
       XJ = C(1,M(JM,K))
       YJ = C(2,M(JM,K))
C
      DO 200 L = 1,NPI
C
      X = 0.5D0*((XJ - XII)*T(L) + XJ + XII)
      Y = 0.5D0*((YJ - YII)*T(L) + YJ + YII)
C
      IFLAG = -1
      CALL TRANSF(M,C,K,XI,ETA,B,BTINV,JACOB,X,Y,IFLAG)

      CALL BASES(PHI,IEL,XI,ETA)
      CALL GRADIENTE(GRAD,IEL,XI,ETA)
      CALL CONVECCAO(X,Y,BETA)
C
C     ---------------------------------------
C     Calculo dos Gradientes no Elemento Real
C     ---------------------------------------
      DO  I = 1,NFB
C
      GRADREAL(1,I)  =  BTINV(1,1)*GRAD(1,I)  +  BTINV(1,2)*GRAD(2,I)
      GRADREAL(2,I)  =  BTINV(2,1)*GRAD(1,I)  +  BTINV(2,2)*GRAD(2,I)
C
      END DO
C
      FUNCAO = 0.0D0
      DO 50 J = 1,NFB
      FUNCAO  =  FUNCAO  +  U(M(J,K))*PHI(J)
50    CONTINUE
C
      NORMAL(1) = 0.5D0*(YJ - YII)
      NORMAL(2) = 0.5D0*(XII - XJ)
C
      DS = DSQRT(NORMAL(1)*NORMAL(1) + NORMAL(2)*NORMAL(2))
C
      BETANORMAL = BETA(1)*NORMAL(1)/DS + BETA(2)*NORMAL(2)/DS
C
      GRD(1) = 0.0D0
      GRD(2) = 0.0D0
      DO J = 1,NFB
            DO I = 1,2
            GRD(I) = GRD(I) + U(M(J,K))*GRADREAL(I,J)
            END DO
      END DO
C
      DNORMAL = GRD(1)*NORMAL(1)/DS + GRD(2)*NORMAL(2)/DS
C
      SOMA = SOMA  +  FUNCAO*BETANORMAL*DS*W(L)*step
C
200   CONTINUE
C
      SOLIDO  =   SOLIDO  +  SOMA
C
C
600   CONTINUE
C
C
      WRITE(*,700) SOLIDO
700   FORMAT(/,T10,'MASSA =  ',f15.9)
C
C
C
      RETURN
      END
C---------------------------------------------------------------------
C---------------------------------------------------------------------
```

Subrotina: Bases
Descrição: Expressão das funções de base para elementos triangulares.

```
C-----------------------------------------------------------------------
C           FILE                                        BASES.FOR
C-----------------------------------------------------------------------
C
      SUBROUTINE BASES(PHI,IEL,XI,ETA)
C     ================================
C
      REAL*8 PHI(*),XI,ETA
C
      INTEGER IEL
C
C
C
C     --------------------
C     Variaveis de Entrada
C     --------------------
C
C              IEL : Tipo do Elemento de Lagrange
C
C     ( xi , eta ) : Ponto do Elemento Padrao
C
C
C     -----------------
C     Variavel de Saida
C     -----------------
C
C     PHI :  Vetor com os Valores das Funcoes de Base em (xi , eta)
C
C
C
C     -----------------------------------
C     Elementos de Lagrange do  Tipo  1
C     -----------------------------------
C     Funcoes de Base no Triangulo Padrao
C     -----------------------------------
C
      PHI(1)  =  1.0D0 - XI - ETA
      PHI(2)  =  XI
      PHI(3)  =  ETA
C
C
      IF(IEL.EQ.2)  THEN
C
C
C           -----------------------------------
C           Elementos de Lagrange do  Tipo  2
C           -----------------------------------
C           Funcoes de Base no Triangulo Padrao
C           -----------------------------------
C
            PHI(4)  =  4.0D0*PHI(1)*PHI(2)
            PHI(5)  =  4.0D0*PHI(2)*PHI(3)
            PHI(6)  =  4.0D0*PHI(1)*PHI(3)
C
            DO  I  =  1,3
            PHI(I) =  PHI(I)*(2.0D0*PHI(I) - 1.0D0)
            END DO
C
C
                  ELSE IF(IEL.EQ.3) THEN
C
C                 -----------------------------------------------
C                 Elementos de Lagrange Tipo 1  +  Funcao Bolha
C                 -----------------------------------------------
C                 Funcoes de Base no Triangulo Padrao
C                 -----------------------------------
C
                  PHI(4)  =  27.0D0*PHI(1)*PHI(2)*PHI(3)
C
                  END  IF
C
      RETURN
      END
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
```

Subrotina: Gradiente
Descrição: Expressão dos gradientes das funções de base para elementos triangulares.

```
C-----------------------------------------------------------------------
C             FILE                        GRADIENTE.FOR
C-----------------------------------------------------------------------
C
      SUBROUTINE GRADIENTE(GRAD,IEL,XI,ETA)
C     ====================================
C
      REAL*8 GRAD(2,*),XI,ETA
C
      INTEGER IEL
C
C
C
C
C     --------------------
C     Variaveis de Entrada
C     --------------------
C
C        IEL : Tipo do Elemento de Lagrange
C
C     ( xi , eta ) : Ponto do Elemento Padrao
C
C
C     -----------------
C     Variavel de Saida
C     -----------------
C
C     PHI  :  Vetor com os Valores dos Gradientes  em  (xi , eta)
C
C
C
C
      IF(IEL.EQ.1)  THEN
C
C
C     -------------------------------
C     Elemento de Lagrange do Tipo  1
C     -------------------------------
C     Gradientes das Funcoes de Base no Triangulo Padrao
C     --------------------------------------------------
C
C
      GRAD(1,1)  =  -1.0D0
      GRAD(2,1)  =  -1.0D0
C
      GRAD(1,2)  =   1.0D0
      GRAD(2,2)  =   0.0D0
C
      GRAD(1,3)  =   0.0D0
      GRAD(2,3)  =   1.0D0
C
C
      ELSE IF(IEL.EQ.2)  THEN
C
C
C     -------------------------------
C     Elementos de Lagrange do Tipo  2
C     -------------------------------
C     Gradientes das Funcoes de Base no Triangulo
C     -------------------------------------------
C
C
      GRAD(1,1)  =   4.D0*( XI + ETA ) - 3.0D0
      GRAD(2,1)  =   4.D0*( XI + ETA ) - 3.0D0
C
      GRAD(1,2)  =   4.D0*XI - 1.0D0
      GRAD(2,2)  =   0.0D0
C
      GRAD(1,3)  =   0.0D0
      GRAD(2,3)  =   4.0D0*ETA - 1.0D0
C
      GRAD(1,4)  =   4.0D0 - 8.0D0*XI - 4.0D0*ETA
      GRAD(2,4)  =  -4.0D0*XI
C
      GRAD(1,5)  =   4.0D0*ETA
```

```
      GRAD(2,5)  =   4.0D0*XI
C
      GRAD(1,6)  =  -4.0D0*ETA
      GRAD(2,6)  =   4.0D0 - 4.0D0*XI - 8.0D0*ETA
C
C
      ELSE IF(IEL.EQ.3) THEN
C
C
C     ------------------------------------------
C     Elementos de Lagrange Tipo 1 + Funcao Bolha
C     ------------------------------------------
C     Gradientes das Funcoes de Base no Triangulo
C     ------------------------------------------
C
      GRAD(1,1)  =  -1.0D0
      GRAD(2,1)  =  -1.0D0
C
      GRAD(1,2)  =   1.0D0
      GRAD(2,2)  =   0.0D0
C
      GRAD(1,3)  =   0.0D0
      GRAD(2,3)  =   1.0D0
C
      GRAD(1,4)  =   27.0D0*ETA*( 1.0D0 - ETA - 2.0D0*XI )
      GRAD(2,4)  =   27.0D0*XI*(1.0D0 - 2.0D0*ETA - XI )
C
C
      END IF
C
C
C
      RETURN
      END
C---------------------------------------------------------------------
C---------------------------------------------------------------------
```

Subrotina: Bases1
Descrição: Expressão das funções de base para elementos unidimensionais.

```
C---------------------------------------------------------------------
C               FILE                      BASES1.FOR
C---------------------------------------------------------------------
C
      SUBROUTINE BASES1(T,PHI,DPHI)
C     ============================
C
      REAL*8 T,PHI(*),DPHI(*)
C
C
C     FUNCOES DE LAGRANGE LINEARES 1D
C
C     T  pertence  [ -1 , 1 ]
C
C
      PHI(1)  =  ( 1 - T )*5.D-1
C
      PHI(2)  =  ( 1 + T )*5.D-1
C
C
C     DERIVADA DAS FUNCOES DE LAGRANGE LINEARES
C
C     T  pertence  [ -1 , 1 ]
C
C
      DPHI(1)  =  -5.D-1
C
      DPHI(2)  =   5.D-1
C
C
      RETURN
      END
C---------------------------------------------------------------------
C---------------------------------------------------------------------
```

Subrotina: Transf
Descrição: Transformação linear entre elemento triangular real e padrão.

```
c-----------------------------------------------------------------------
c              FILE                              TRANSF.FOR
c-----------------------------------------------------------------------
c
      subroutine transf(M,C,K,xi,eta,B,BTINV,jacob,x,y,iflag)
c     ======================================================
c
      real*8 C(2,*), xi, eta, jacob, x, y
      real*8 B(2,*), BTINV(2,*), eps
c
      integer   M(6,*),iflag
      integer   K, ip, jp, kp
c
      data eps / 1.0d-5 /
c
c
c
c     --------------------
c     Variaveis de Entrada
c     --------------------
c
c               M  :  Matriz de Incidencia
c
c               C  :  Matriz de Coordenadas dos Nos
c
c               K  :  Numero do Elemento onde Estamos Integrando
c
c    ( xi , eta )  :  Ponto do Elemento Padrao
c
c
c     ------------------
c     Variaveis de Saida
c     ------------------
c
c        B  :  Matriz da Transformacao Afim
c
c    BTINV  :  Inversa da Transposta da Matriz da Transformacao
c
c    jacob  :  Jacobiano da Transformacao - det(B)
c
c ( x, y )  :  Ponto do Elemento Real, Imagem de ( xi , eta )
c
c
c
c     ------------------------------------------------------------
c     Mudanca de Variavel : Triangulo Padrao <---> Triangulo Real
c
c                             ( xi , eta )   <--->   ( x , y )
c     ------------------------------------------------------------
c
c
c
c     Enumeracao dos Vertices do Triangulo Real
c     -----------------------------------------
c
      ip = M(1,K)
      jp = M(2,K)
      kp = M(3,K)
c
c
c     Calculo da Matriz Jacobiana
c     ---------------------------
c
      B(1,1) =  C(1,jp) - C(1,ip)
      B(1,2) =  C(1,kp) - C(1,ip)
      B(2,1) =  C(2,jp) - C(2,ip)
      B(2,2) =  C(2,kp) - C(2,ip)
c
c
c     Calculo do Determinante da Matriz Jacobiana
c     -------------------------------------------
c
      jacob = B(1,1)*B(2,2) - B(1,2)*B(2,1)
c
      if(jacob.le.eps) then
```

```
      write(*,1000) K, jacob
1000  format(5(/),T10,'Area do Elemento  ',I4.4,'   =   ',D10.3,//)
c
      write(*,*) '          Matriz de Incidencia'
      write(*,1100) (  M(i,K) , i = 1,4 )
1100  format(/,T10,4(I4.4,5X),/)
      write(*,*) '          Matriz de Coordenadas'
      do i = 1,3
      write(*,1200) C(1,M(i,K)), C(2,M(i,K))
1200  format(T10,2(D17.9,5X))
      end do
c
      stop
      end if
c
c
c     ----------------------------------------------------
c     Calculo da Inversa da Transposta da Matriz Jacobiana
c     ----------------------------------------------------
c
      BTINV(1,1) =  B(2,2)/jacob
      BTINV(1,2) = -B(2,1)/jacob
      BTINV(2,1) = -B(1,2)/jacob
      BTINV(2,2) =  B(1,1)/jacob
c
c
c     ------------------------------------------------------------
c     Calculo da Transformcao de    ( xi , eta )  --->  ( x , y )
c     ------------------------------------------------------------
c
      if (iflag .eq. 1) then
       x = B(1,1)*xi + B(1,2)*eta + C(1,ip)
       y = B(2,1)*xi + B(2,2)*eta + C(2,ip)
       return
      endif
c
c
c
c     ------------------------------------------------------------
c     Calculo da Transformcao de    ( x , y )  --->  ( xi , eta )
c     ------------------------------------------------------------
c
      if (iflag .eq. -1) then
       xi = BTINV(1,1)*(x - C(1,ip)) + BTINV(2,1)*(y - C(2,ip))
       eta = BTINV(1,2)*(x - C(1,ip)) + BTINV(2,2)*(y - C(2,ip))
      endif
c
c
c
      return
      end
c
c---------------------------------------------------------------------
c---------------------------------------------------------------------
```

Subrotina: Transf1
Descrição: Transformação linear entre elemento unidimensional real e padrão.

```
c---------------------------------------------------------------------
c             FILE                        TRANSF1.FOR
c---------------------------------------------------------------------
C
      SUBROUTINE TRANSF1(T,XMIN,XMAX,XX,IFLAG)
C     ========================================
C
      REAL*8 T,XX,XMIN,XMAX
C
      INTEGER*2 IFLAG
C
C
C
C     Transformacao Afim  :
C     ---------------------
C
C     IFLAG =  1 [ -1 , 1 ] ------> [ xmin , xmax ]
C
```

```
C                    T -----------------> XX = F(T)
C
C     IFLAG = -1 [ xmin , xmax ] ------> [ -1 , 1 ]
C
C                    T -----------------> XX = F(T)
C
C
C
C
      IF(IFLAG.EQ.1) THEN
      XX = ( ( XMAX - XMIN )*T + ( XMAX + XMIN ) )*5.0D-1
      ELSE
      XX = ( 2.D0*T - ( XMIN + XMAX ) ) / ( XMAX - XMIN )
      END IF
C
C
C
      RETURN
      END
C---------------------------------------------------------------------
C---------------------------------------------------------------------
```

Subrotina: Escalar
Descrição: Produto escalar entre os gradientes das funções de base.

```
C---------------------------------------------------------------------
C             FILE                                ESCALAR.FOR
C---------------------------------------------------------------------
C
      SUBROUTINE ESCALAR(GRAD,GRADREAL,BTINV,NFB,SCALAR)
C     ==================================================
C
      REAL*8 GRAD(2,*),BTINV(2,*),GRADREAL(2,6),SCALAR(6,*)
      REAL*8 PROD
C
      INTEGER NFB
C
C
C
c     --------------------
c     Variaveis de Entrada
c     --------------------
c
c     GRAD  :  Gradientes das Funcoes de Base
c
c    BTINV  :  Inversa da Transposta da Matriz da Transformacao
c
c      NFB  :  Numero de Funcoes de Base
c
c
c     -----------------
c     Variavel de Saida
c     -----------------
c
c   SCALAR  :  Produto Escalar entre os Gradientes
c
C
C
C     ---------------------------------------
C     Calculo dos Gradientes no Elemento Real
C     ---------------------------------------
      DO  K = 1,NFB
C
      GRADREAL(1,K)  =  BTINV(1,1)*GRAD(1,K)  +  BTINV(1,2)*GRAD(2,K)
      GRADREAL(2,K)  =  BTINV(2,1)*GRAD(1,K)  +  BTINV(2,2)*GRAD(2,K)
C
      END DO
C
C
C     ------------------------------------------------
C     Calculo do Produto Escalar entre os Gradientes
C     ------------------------------------------------
      DO  I = 1,NFB
      DO  J = I,NFB
C
      PROD = GRADREAL(1,I)*GRADREAL(1,J) + GRADREAL(2,I)*GRADREAL(2,J)
```

```
      SCALAR(I,J) = PROD
C
      SCALAR(J,I) = SCALAR(I,J)
C
      END DO
      END DO
C
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
```

Subrotina: Transporte

Descrição: Produto escalar entre os gradientes das funções de base e o campo de velocidades.

```
C-----------------------------------------------------------------------
C                FILE                      TRANSPORTE.FOR
C-----------------------------------------------------------------------
C
      SUBROUTINE TRANSPORTE(GRAD,BETA,BTINV,NFB,BETAGRAD)
C     ===================================================
C
      REAL*8 GRAD(2,*),BTINV(2,*),GRADREAL(2,6)
      REAL*8 BETAGRAD(*), BETA(*)
C
      INTEGER NFB
C
C
C
c     --------------------
c     Variaveis de Entrada
c     --------------------
c
c     GRAD  :  Gradientes das Funcoes de Base
c
c    BTINV  :  Inversa da Transposta da Matriz da Transformacao
c
c      NFB  :  Numero de Funcoes de Base
c
c
c
c     -----------------
c     Variavel de Saida
c     -----------------
c
c   BETAGRAD  :  Produto Escalar entre o termo de transporte
C                      e os Gradientes
c
C
C
C     ---------------------------------------
C     Calculo dos Gradientes no Elemento Real
C     ---------------------------------------
C
      DO  K = 1,NFB
C
      GRADREAL(1,K)  =  BTINV(1,1)*GRAD(1,K)  +  BTINV(1,2)*GRAD(2,K)
      GRADREAL(2,K)  =  BTINV(2,1)*GRAD(1,K)  +  BTINV(2,2)*GRAD(2,K)
C
      END DO
C
C
C     -----------------------------------------
C     Calculo do Produto Escalar
c     entre O Termo de Transporte e os Gradientes
c     -----------------------------------------
c
      DO I = 1,NFB
c
      BETAGRAD(I) = GRADREAL(1,I)*BETA(1) + GRADREAL(2,I)*BETA(2)
c
      END DO
```

```
c
c
c
      RETURN
END
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
```

Subrotina: Boundvetor

Descrição: Colocação das condições de contorno essenciais no vetor do lado direito do sistema linear resultante da discretização.

```
C-----------------------------------------------------------------------
C          FILE                         BOUNDVETOR.FOR
C-----------------------------------------------------------------------
C
      SUBROUTINE BOUNDVETOR(M,C,INB,F,var,lista,listanew)
C     ==================================================
C
      REAL*8  C(2,*),F(*)
      REAL*8  X, Y, CONTORNO
C
      INTEGER    M(6,*),INB(*)
      INTEGER  NTNOS,NTEL,NVER,NFB,NDIM
      INTEGER  ILD, var(*), lista(*), listanew(*)
C
      COMMON/ INFOREDE / NTNOS,NDIM,NTEL,NVER,NFB
C
C
C
C     ------------------------------
C     Variaveis de Entrada  e  Saida
C     ------------------------------
C
C      M  :  Matriz de Incidencia
C
C      C  :  Matriz de Coordenadas dos Nos
C
C      F  :  Vetor de Carga
C
C  NTNOS  :  Numero Total de Nos da Rede
C
C   NTEL  :  Numero Total de Elementos da Rede
C
C   NVER  :  Numero de Vertices do Elemento
C
C    NFB  :  Numero de Funcoes de Base Local
C
C    INB  :  Vetor que Contem a Localizacao dos Nos
C
C
C
C
C
C*****************************************************
C*    ---------------------------------------        *
C*    Coloca a Condicao de Contorno Essencial        *
C*                                                   *
C*     no Vetor de Carga                             *
C*    ---------------------------------------        *
C*****************************************************
C
C
C
      DO  I = 1,2*NTNOS
      IF (( INB(I) .NE. 0 )  .AND. ( VAR(I) .NE. 0 )) THEN
        F(VAR(I)) = 0.0D0
      END IF
      END DO
C
C
      DO 200 K = 1,NTEL
C
      DO 100 I = 1,NVER
C
```

```
      IF ((LISTANEW(K) .EQ. 1) .OR. (LISTA(K) .EQ. 1)) THEN
C
       IM = M(I,K)
        X = C(1,IM)
        Y = C(2,IM)
      ILD = INB(IM)
      IF ( INB(IM) .NE. 0 )  F(VAR(IM)) = F(VAR(IM)) + CONTORNO(X,Y,ILD)
C
      END IF
C
100   CONTINUE
C
200   CONTINUE
C
C
C
      RETURN
      END
C---------------------------------------------------------------------
C---------------------------------------------------------------------
```

Subrotina: Essencial

Descrição: Colocação das condições de contorno essenciais nas submatrizes do sistema linear resultante da discretização.

```
C---------------------------------------------------------------------
C             FILE                              ESSENCIAL.FOR
C---------------------------------------------------------------------
C
C
       SUBROUTINE ESSENCIAL(M,SUBMAT,INB,K)
C      ====================================
C
       REAL*8  SUBMAT(6,*)
C
       INTEGER    M(6,*), INB(*)
       INTEGER  NTNOS, NTEL, NVER, NFB, NDIM
C
       COMMON/ INFOREDE / NTNOS,NDIM,NTEL,NVER,NFB
C
C
C
C
C
C     ------------------------------
C     Variaveis de Entrada  e  Saida
C     ------------------------------
C
C      M  :  Matriz de Incidencia
C
C      C  :  Matriz de Coordenadas dos Nos
C
C SUBMAT  :  SubMatrizes
C
C  NTNOS  :  Numero Total de Nos da Rede
C
C   NTEL  :  Numero Total de Elementos da Rede
C
C   NVER  :  Numero de Vertices do Elemento
C
C    NFB  :  Numero de Funcoes de Base Local
C
C    INB  :  Vetor que Contem a Localizacao dos Nos
C
C
C
C
C
C**************************************************
C*    ---------------------------------------     *
C*    Coloca a Condicao de Contorno Essencial     *
C*                                                *
C*     no Vetor de Carga e nas SubMatrizes        *
C*    ---------------------------------------     *
C**************************************************
```

```
C
C
C
C
      DO 100 I = 1,NVER
C
       IM = M(I,K)
C
      IF ( INB(IM) .NE. 0 )  THEN
C
            DO J = 1,NVER
            SUBMAT(I,J) = 0.0D0
            END DO
C
            SUBMAT(I,I) = 1.0D0
C
C
      END IF
C
C
100   CONTINUE
C
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
```

Subrotina: Zeros

Descrição: Zera o vetor do lado direito do sistema linear resultante e reserva espaço para a alocação vetorizada da matriz deste sistema linear.

```
C-----------------------------------------------------------------------
C             FILE                                    ZEROS.FOR
C-----------------------------------------------------------------------
C
      SUBROUTINE ZEROS(A,F,IROWST,LENROW,JCOL,NDIM,LINHA)
C     ==================================================
C
      REAL*8 A(*), F(*)
C
      INTEGER IROWST(*), LENROW(*), JCOL(*), NDIM, LINHA
C
C
      DO I = 1,NDIM
      F(I) = 0.0D0
      END DO
C
      LINHA = 20
      IROWST(1) = 1
      LENROW(1) = 0
      DO I = 2,NDIM
      IROWST(I) = IROWST(I-1) + LINHA
      LENROW(I) = 0
      END DO
C
      DO I = 1,(LINHA*NDIM)
      A(I) = 0.0D0
      JCOL(I) = 0
      END DO
C
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
```

Subrotina: Zerasub
Descrição: Zera as submatrizes relativas aos elementos bidimensionais.

```
C-----------------------------------------------------------------------
C          FILE                                ZERASUB.FOR
C-----------------------------------------------------------------------
C
     SUBROUTINE ZERASUB(MSUB,KSUB,VSUB,PSUB,LSUB,NSUB,QSUB,OSUB,
    &                   RSUB,KKSUB,OOSUB,RRSUB,NFB)
C    ==========================================================
C
      REAL*8 MSUB(6,*),KSUB(6,*),VSUB(6,*),PSUB(6,*),LSUB(6,*)
      REAL*8 NSUB(6,*),QSUB(6,*),OSUB(6,*),RSUB(6,*)
      REAL*8 KKSUB(6,*),OOSUB(6,*), RRSUB(6,*)
C
      INTEGER NFB
C
C
      DO  I = 1,NFB
      DO  J = 1,NFB
C
      MSUB(I,J) = 0.0D0
      KSUB(I,J) = 0.0D0
      VSUB(I,J) = 0.0D0
      PSUB(I,J) = 0.0D0
      LSUB(I,J) = 0.0D0
      NSUB(I,J) = 0.0D0
      QSUB(I,J) = 0.0D0
      OSUB(I,J) = 0.0D0
      RSUB(I,J) = 0.0D0
      KKSUB(I,J) = 0.0D0
      OOSUB(I,J) = 0.0D0
      RRSUB(I,J) = 0.0D0
C
      END DO
      END DO
C
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
```

Subrotina: Procura
Descrição: Relação entre a enumeração das funções de base nos elementos uni e bidimensionais com a enumeração das funções de base nos elementos tridimensionais.

```
C-----------------------------------------------------------------------
C           FILE                                  PROCURA.FOR
C-----------------------------------------------------------------------
C
      SUBROUTINE PROCURA(K,I,J)
C     =========================
C
      INTEGER I,J,K
C
      IF (K .LE. 3) THEN
      I = K
      J = 1
C
      ELSE
      I = K - 3
      J = 2
C
      END IF
C
      RETURN
      END
C
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
```

Subrotina: Fastfind

Descrição: Verifica se uma determinada posição da matriz do sistema linear já foi alocada no esquema de vetorização.

```
C-----------------------------------------------------------------------
C         FILE                            FASTFIND.FOR
C-----------------------------------------------------------------------
C
      SUBROUTINE FASTFIND(IROWST,LENROW,JCOL,NELEM,INEW,JNEW,KIJ,L)
C     ============================================================
C
      INTEGER    IROWST(*), JCOL(*), LENROW(*)
      INTEGER    NELEM, KIJ, INEW, JNEW
C
C
C     ------------------------------------------------
C     VERIFICA SE O ELEMENTO  ( INEW, JNEW )  JA FOI ALOCADO
C     ------------------------------------------------
C
      DO I = IROWST(INEW),(IROWST(INEW) + L - 1)
C
      IF(JNEW.EQ.JCOL(I)) THEN
      KIJ = I
      RETURN
      END IF
C
      IF(JCOL(I).EQ.0)  THEN
      KIJ = I
      EXIT
      END IF
C
      END DO
C
C
      NELEM = NELEM + 1
      LENROW(INEW) = LENROW(INEW) + 1
      JCOL(KIJ) = JNEW
C
C
      RETURN
      END
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
```

Subrotina: Posproc

Descrição: Transformação do esquema de vetores esparsos para o esquema de vetorização por coordenadas.

```
C-----------------------------------------------------------------------
C             FILE                          POSPROC.FOR
C-----------------------------------------------------------------------
C
      SUBROUTINE  POSPROC(IROWST,LENROW,ILIN,JCOL,A,NDIM)
C     ===================================================
C
      REAL*8   A(*)
C
      INTEGER  IROWST(*), LENROW(*), ILIN(*), JCOL(*)
      INTEGER  NDIM, KONT, KINICIAL, K
C
C
C     Transformacao no Armazenamento Esparso
C     --------------------------------------
C
C     Esquema de Vetores Esparsos  para o  Esquema de Coordenadas
C     -----------------------------------------------------------
C
C     Construcao do Vetor ILIN
C     ------------------------
C
```

```
      KONT = 0
      DO I = 1,NDIM
C
           DO J = 1,LENROW(I)
            KONT = KONT + 1
            ILIN(KONT) = I
            END DO
C
      END DO
C
C
C     Reconstrucao dos Vetores  JCOL   e   A
C     ---------------------------------------
C
      KONT = 0
      DO I = 1,NDIM
C
            KINICIAL = IROWST(I)
            DO J = 1,LENROW(I)
            K = KINICIAL + J - 1
            KONT = KONT + 1
            JCOL(KONT) = JCOL(K)
            A(KONT) =    A(K)
            END DO
C
      END DO
C
C
C     Reconstrucao do Vetor IROWST
C     ----------------------------
C
      IROWST(1) = 1
      DO I = 2,NDIM
      IROWST(I) = IROWST(I-1) + LENROW(I-1)
      END DO
C
C
C
      RETURN
      END
C---------------------------------------------------------------------
C---------------------------------------------------------------------
```

Bibliografia

[1] Axelsson, O.; Barker, V. A.; *Finite element solution of boundary value problems*, Academic Press, 1984.

[2] Baiocchi C.; Franca, L.P.; Brezzi, F.; *"Virtual bubbles and the Galerkin-least-squares method"*, Computer Methods in Applied Mechanics and Engineering, No. 105, pp. 125-141, 1993.

[3] Barenblatt, G. I.; *"On Some Unsteady Motions in a Liquid or a Gas in a Porous Media*, Prikladnaja Matematika i Mechanika, No. 16, pp. 67-78, 1952.

[4] Benqué, J.P.; Hauguel, A.; Viollet, P.L.; *"Engineering Application of Computational Hydraulics"*, Pitman Advanced Publishing Program, London, 1982.

[5] Bertsch, M.; *"Nonlinear Diffusion Problems: The Large Time Behavior*, Tese de Doutorado, Universidade de Leiden, Holanda, 1983.

[6] Cantão, R.; *"Modelagem e Simulação Numérica de Derrames de Óleo no Canal de São Sebastião, SP "*, Tese de Mestrado, DMA/IMECC/UNICAMP, 1998.

[7] Cantelano, M.; *"Métodos Numéricos para Problemas de Convecção-Difusão"*, Tese de Mestrado, DMA/IMECC/UNICAMP, 1997.

[8] Christie, I.; Griffiths, D.F.; Mitchell A.R.; *"Finite element method for second order differential equations with significant first derivates"*, Int. Journal Num. Meth. in Engrg., No. 10, pp. 1389-1396, 1976.

[9] Connor, J.J.; Brebbia, C.A.; *Finite Element Techniques for Fluid Flow*, Newnes-Butterworths, 1976.

[10] Cuesta, I.; Grau, F.; Giralt F.; *"Numerical Simulation of Oil Spills in a Generalized Domain"*, Oil & Chemical Pollution, No. 7, pp. 143-159, 1990.

[11] Dawson, C.; *"Analysis of an Upwind-Mixed Finite Element Method for Nonlinear Contaminant Transport Equations'*, SIAM J. Numer. Anal., Vol. 35, No. 5, pp. 1709-1724, 1998.

[12] Donea, J.; Quartapelle, L.; *"An introduction to finite element methods for transient advection problems"*, Computer Methods in Applied Mechanics and Engineering, No. 95, pp. 169-203, 1992.

[13] Fay, J.A.; *"The Spread of Oil Slicks on a Calm Sea"*, Oil on the Sea, ed. D. P. Hoult Plenun Press, pp. 53-63, 1969.

[14] Fernandes, M. R.; Pulino, P.; *"Métodos estabilizados de elementos finitos"*, Relatório de Pesquisa 39/98, IMECC/UNICAMP, 1998.

[15] Franca, L.P.; Hughes, T.J.R.; Hulbert, G.M.; *"A new finite element formulation for computational fluid dynamics: VIII. The Galerkin/Least-squares method for advective-diffusive equations"*, Computer Methods in Applied Mechanics and Engineering, No. 73, pp. 173-189, 1989.

[16] Franca, L.P.; Frey, S.L.; Hughes, T.J.R.; *"Stabilized finite element method: I. Application to the for advective-diffusive model"*, Computer Methods in Applied Mechanics and Engineering, No. 95, pp. 253-276, 1992.

[17] Franca, L.P.; Frey, S.L.; *"Stabilized finite element method: II. The incompressible Navier-Stokes equations"*, Computer Methods in Applied Mechanics and Engineering, No. 99, pp. 209-233, 1992.

[18] Franca, L.P.; Brezzi, F.; Bristeau, M.O.; Mallet, M.; Rogé, G.; *"A relationship between stabilized finite element methods and the Galerkin method with bubble functions"*, Computer Methods in Applied Mechanics and Engineering, No. 96, pp. 117-129, 1992.

[19] Franca, L.P.; Madureira, A.L.; *"Element diameter free stability parameters for stabilized methods applied to fluids"*, Computer Methods in Applied Mechanics and Engineering, No. 105, pp. 395-403, 1993.

[20] Franca, L.P.; Farhat, C.; *"On the limitations of bubble functions"*, Computer Methods in Applied Mechanics and Engineering, No. 117, pp. 225-230, 1994.

[21] Franca, L.P.; Farhat, C.; *"Bubble functions prompt unusual stabilized finite element methods"*, Computer Methods in Applied Mechanics and Engineering, No. 123, pp. 299-308, 1995.

[22] Franca, L.P.; Lesoinne, M.; Farhat, C.; *"Unusual stabilized finite element methods for second order linear differential equations"*, Proceedings of the Ninth International Conference on Finite Elements in Fluids - New Trends and Applications, pp. 377-386, Venice, Italy, 1995.

[23] Franca, L.P.; Valentin, F.; *"Combining stabilized finite element methods"*, Matemática Aplicada e Computacional, Vol. 14, No. 3, pp. 67-82, 1995.

[24] Franca, L.P.; Brezzi, F.; Hughes, T.J.R.; Russo, A.; *"Stabilization techniques and subgrid scales capturing"*, Proceedings of the Conference on the State of the Art in Numerical Analysis, York, England, 1996.

[25] Franca, L.P.; Russo, A.; *"Approximation of the Stokes problem by residual-free macro bubbles"*, East-West Journal of Numerical Analysis, Vol. 4, pp. 265-278, 1996.

[26] Franca, L.P.; Russo, A.; *"Deriving upwinding, mass lumping and selective reduced integration by residual-free bubbles"*, Applied Mathematics Letters, Vol. 9, pp. 83-88, 1996.

[27] Franca, L.P.; Russo, A.; *"Mass lumping emanating from residual-free bubbles"*, Computer Methods in Applied Mechanics and Engineering, No. 142, pp. 353-360, 1997.

[28] Franca, L.P.; Russo, A.; *"Unlocking with residual-free bubbles"*, Computer Methods in Applied Mechanics and Engineering, No. 142, pp. 361-364, 1997.

[29] Franca, L.P.; Brezzi, F.; Hughes, T.J.R.; Russo, A.; *"b = integral g"*, Computer Methods in Applied Mechanics and Engineering, No. 145, pp. 329-339, 1997.

[30] Ghia, U.; Ghia, K.N.; Shin, C.T.; *"High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method"*, Journal of Computational Physics, No. 48, pp. 387-411, 1982.

[31] Griebel, M.; Dornseifer, T.; Neunhoeffer, T.; *Numerical Simulation in Fluid Dynamics - A Practical Introduction*, SIAM, 1998.

[32] Hughes, T.J.R.; Brooks, A.N.; *" A multidimensional upwind scheme with no crosswind diffusion"*, Finite Element Methods for Convection Dominated Flows, AMD, Vol. 34, pp. 19-35, 1979.

[33] Johnson, C.; Nävert, U.; *" An analysis of some finite element methods for advection-diffusion problems"*, Analytical and Numerical Approaches to Asymptotic Problems in Analysis, North-Holland, pp. 99-116, 1981.

[34] Johnson, C.; Nävert, U.; Pitkäranta, J.; *" Finite element methods for linear hyperbolic problems"*, Comput. Methods Appl. Mech. Engrg., No. 45, pp. 285-312, 1984.

[35] Johnson, C.; Saranen, J.; *" Streamline Diffusion Methods for the Incompressible Euler and Navier-Stokes Equations"*, Mathematics of Computation, Vol. 47, pp. 1-18, 1986.

[36] Johnson, C.; Schatz, A.; Wahlbin, L.; *" Crosswind Smear and Pointwise Errors in Streamline Diffusion Finite Element Methods"*, Mathematics of Computation, Vol. 49, No. 179, pp. 25-38, 1987.

[37] Johnson, C.; Larsson, S.; Thomée V.; Wahlbin, L.; *" Error Estimates for Spatially Discrete Approximations of Semilinear Parabolic Equations with Nonsmooth Initial Data"*, Mathematics of Computation, Vol. 49, No. 180, pp. 331-357, 1987.

[38] Johnson, C.; Szepessy A.; *" On the Convergence of a Finite Element Method for a Nonlinear Hyperbolic Conservation Law"*, Mathematics of Computation, Vol. 49, No. 180, pp. 427-444, 1987.

[39] Johnson, C.; *Numerical solution of partial differential equations by the finite element method*, Cambridge University Press, 1987.

[40] Johnson, C.; Eriksson, K.; *" An Adaptative Finite Element Method for Linear Elliptic Problems"*, Mathematics of Computation, Vol. 50, No. 182, pp. 361-383, 1988.

[41] Johnson, C.; *" Adaptative Finite Element Methods for Diffusion and Convection Problems"*, Computer Methods in Applied Mechanics and Engineering, No. 82, pp. 301-322, 1990.

[42] Johnson, C.; Eriksson, K.; *" Adaptative Streamline Diffusion Finite Element Methods for Stationary Convection-Diffusion Problems"*, Mathematics of Computation, Vol. 60, No. 201, pp. 167-188, 1993.

[43] Maliska, C. R.; *Transferência de Calor e Mecânica dos Fluidos Computacional*, Livros Técnicos e Científicos Editora, 1995.

[44] Nävert, U.; *" A Finite Element Method for Convection-Diffusion Problems"*, Tese de Doutorado, Chalmers University of Technology and University of Gothenburg, 1982.

[45] Pattle, R. E.; *"Diffusion from an Instantaneous Point Source with Concentration Dependent Coefficient"*, Quart. J. Mech. Appl. Math., No. 12, pp. 407-409, 1959.

[46] Peaceman; *"Fundamentals of Numerical Reservoir Simulation"*, Elsevier Scientific Publishing Company, 1977.

[47] Pierre, R.; *"Simple C^0 Approximations for the Computation of Incompressible Flows"*, Computer Methods in Applied Mechanics and Engineering, No. 68, pp. 205-227, 1988.

[48] Pudykiewics, J.; Staniforth, A.; *"Some Properties and Comparative Performance of the Semi-Lagrangian Method of Robert in the Solution of the Advection-Diffusion Equation"*, Atmosphere-Ocean, No. 22, pp. 283-308, 1984.

[49] Reddy, B.D.; *Funtional Analysis and Boundary-value Problems: an Introductory Treatment*, Longman Scientific & Techinical, 1986.

[50] Robert, A.; Henderson, J.; Turnbull, C.; *"An implicit time integration scheme for baroclinic models of the atmosphere"*, Mon. Weather Rev., No. 100, pp. 329-335, 1972.

[51] Robert, A.; *"A Stable Numerical Integration Scheme for the Primitive Meteorological Equations"*, Atmosphere-Ocean, No. 19, pp. 35-46, 1981.

[52] Russo, A.; Brezzi, F.; *"Choosing bubbles for advection-diffusion problems"*, Math. Models Methods Appl. Sci., No. 4, pp. 571-587, 1994.

[53] Russo, A.; Franca, L.P.; Brezzi, F.; *"Further considerations on residual-free bubbles for advective-diffusive equations"*, Computer Methods in Applied Mechanics and Engineering, No. 166, pp. 25-33, 1998.

[54] Russo, A.; Marini, D.; Brezzi, F.; *"Applications of pseudo residual-free bubbles to the stabilization of convection-diffusion problems"*, Computer Methods in Applied Mechanics and Engineering, No. 166, pp. 51-63, 1998.

[55] Russo, A.; Pietra, P.; Brezzi, F.; Marini, D.; Manzini, M.; *"Discontinuous finite elements for diffusion problems"*, IAN-CNR Report No. 1112, pp. 1-18, 1998.

[56] Smith G.D.; *"Numerical solution of partial differential equations"*, Clarendon Press-Oxford, 1982.

[57] Staniforth, A.; Côté, J.; *"Semi-Lagrangian integration schemes and their application to environmental flows"*, Lecture Notes in Physics, No. 371, pp. 63-79, 1990.

[58] Thomasset, F.; *Implementation of Finite Element Methods for Navier-Stokes Equations*, Springer-Verlag, 1981.