

Universidade Federal de Santa Catarina – UFSC
Departamento de Informática e Estatística – INE
Curso de Pós-Graduação em Ciência da Computação - CPGCC

Análise de Desempenho do Esquema de Linearização de Newton na
Discretização das Equações do Movimento com Diagramas de Voronoi

por

Ewerton Eyre de Moraes Alonso

Prof. Sérgio Peters, Dr. Eng. - Orientador

Dissertação submetida ao Curso de Pós-graduação em Ciência da Computação da Universidade Federal de Santa Catarina – UFSC como parte integrante dos requisitos exigidos para a obtenção do grau de Mestre.

Florianópolis (SC), dezembro de 1999.

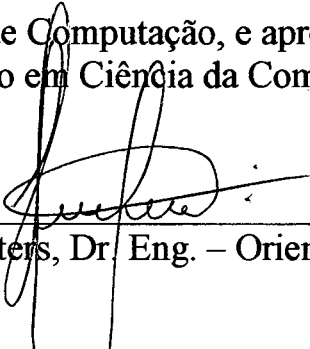
Análise de Desempenho do Esquema de Linearização de Newton na Discretização das Equações do Movimento com Diagramas de Voronoi

Ewerton Eyre de Moraes Alonso

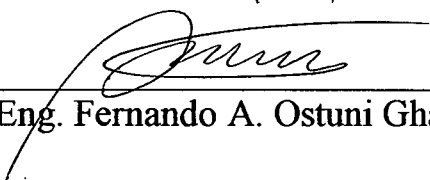
Esta dissertação foi julgada adequada para a obtenção do Título de

Mestre em Ciência da Computação

Área de Concentração : Sistemas de Computação, e aprovada em sua forma final pelo Curso de Pós-Graduação em Ciência da Computação – CPGCC

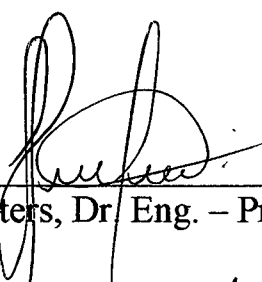


Prof. Sérgio Peters, Dr. Eng. – Orientador




Prof., Dr. Eng. Fernando A. Ostuni Ghauthier – Coordenador

Banca Examinadora



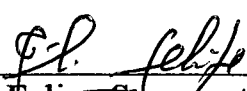
Prof. Sérgio Peters, Dr. Eng. – Presidente



Prof. Daniel Santana de Freitas, Dr. Eng.



Prof. José Mazzusco Jr., Dr. Eng.



Prof. Júlio Felipe Szeremeta, M. Sc.

Ao meu pai Daniel e minha mãe
Adenir.

A minha querida irmã Geysa.

Aos meus queridos avós Ilidia,
Maximinio e Francisca.

Agradecimentos

Ao meu amigo e orientador Sérgio Peters pela oportunidade de realizar este trabalho e pela valiosa orientação, esclarecimento e contribuições a este trabalho.

Em especial a Adriana e Artur pelo carinho, apoio paciência que tornaram possível a realização esta caminhada.

Aos amigos Viviana, Mário, Gilmário e Cleiton aos quais consolidei laços de amizade.

Aos professores Júlio, Daniel e José Mazzucco, membros da banca examinadora, pelas contribuições valiosas ao aprimoramento desta dissertação.

A Verinha e Valdete pelo apoio prestado na coordenação.

A todos os colegas, professores e funcionários da UFSC que de alguma forma contribuíram para a realização deste trabalho.

Resumo

Muitos problemas relacionados à Mecânica dos Fluidos Computacional podem ser modelados matematicamente por equações diferenciais parciais, por exemplo as equações de Navier-Stokes. As metodologias numéricas disponíveis permitem a utilização de malhas adaptáveis a geometria do problema. Dentre tais malhas destacam-se as geradas por Diagrama de Voronoi que, por sua vez, permitem a geração de malhas não estruturadas para a discretização numérica das equações diferenciais parciais.

A discretização numérica pode ser feita através do Método dos Volumes Finitos que gera um sistema de equações diferenciais não lineares de segunda ordem, que necessita de uma forma de linearização para que possa ser resolvido como um sistema linear de equações. Tal sistema de equações geralmente é esparso e de alta ordem, reforçando a necessidade de um método iterativo para a sua solução, envolvendo um grande custo computacional.

O presente trabalho aborda a descrição, análise e implementação de uma linearização alternativa na discretização das equações de Navier-Stokes usando a expansão em Série de Taylor, conforme aplicado pelo método de Newton, utilizando interpolação UpWind de 1ª ordem, no método dos Volumes Finitos com malhas não-estruturadas geradas por Diagramas de Voronoi.

Na solução do sistema linear de equações resultantes da discretização das equações de Navier-Stokes utilizou-se um método iterativo pertencente à família do Gradiente Conjugado (GC).

Na validação dos resultados numéricos, utilizou-se o escoamento laminar incompressível em uma cavidade quadrada de profundidade infinita sujeita a uma parede superior deslizante com velocidade constante, semelhante a uma esteira.

Ainda é apresentada uma descrição sucinta da regra de Armijo que foi embutida no método de Newton com o objetivo de estabilizar o sistema de equações e acelerar a sua convergência.

Abstract

Many problems related to the Computational Fluid Dynamics can be modelled mathematically by partial differential equations, such as the Navier-Stokes equations. The numerical methodologies available allow the use of meshes adaptable to the geometry of the problem. Among such networks are mainly the ones produced by the Diagram of Voronoi which, by its turn, allows the generation of non-structured networks for the numerical discretization of partial differential equations.

The numerical discretization can be done through the Method of Finite Volumes which produces a system of non-linear equations of second order, which needs a way of linearization in order to be solved as linear system of equations. Such system of equations is usually sparse and of high order, reinforcing the need of iterative method for its solution, involving a great computational cost.

This paper approaches the description, analysis and implementation of an alternative linearization in the discretization of the Navier-Stokes equations using the expansion in Taylor Series using UpWind interpolation of first order, in the method of Finite Volumes with non-structured grids produced by Diagrams of Voronoi.

In the solution of the linear system of equations resulting from the discretization of the Navier-Stokes equations was used an iterative method belonging to the Conjugated Gradient (CG) family.

In the validation of numerical results, an incompressible laminar flow was used in a square cavity of infinite depth subject to a sliding superior wall with constant speed, similar to a treadmill.

A succinct description of the Armijo rule is also presented and it was built-in in the method of Newton in order to stabilize the system of equations and accelerate their convergence.

Sumário

Agradecimentos	vii
Resumo	viii
Abstract	ix
Sumário	x
Lista de Figuras	xii
Lista de Tabelas	xiii
Simbologia	xiv
Capítulo 1 – Introdução	17
1.1 Considerações Iniciais	17
1.2 Linearização Através do Método de Newton	18
1.3 Objetivo	19
1.4 Organização do Trabalho	19
Capítulo 2 – Discretização das Equações de Navier-Stokes	21
2.1 Introdução	21
2.2 Discretização das Equações de Navier-Stokes	21
Capítulo 3 – Linearização Conforme o Método de Newton.	29
3.1 Introdução	29
3.2 Discretização das Equações de Navier-Stokes usando a Linearização conforme o Método Newton	29
3.3 Interpolação Genérica Para u_{ik} e v_{ik} , entre os Pontos Nodais i e seus vizinhos $r(k)$	33
3.4 Obtenção das Equações do Movimento Discretizadas com Interpolação UDS para u_{ik}	35
3.5 Obtenção das Equações do Movimento Discretizadas c/ Interpolação UDS para v_{ik}	39
3.6 Equação para a Pressão	43

Capítulo 4 – Resultados e Discussões	46
4.1 Introdução	46
4.2 Formulação do Problema	46
4.3 Análise de Resultados Obtidos	50
4.3.1 Resultados Obtidos com Reynolds 100	51
4.3.2 Resultados Obtidos com Reynolds 1000	59
4.3.3. Reynolds 10000	68
4.4 Conclusões	68
Capítulo 5 – Perspectivas para Trabalhos Futuros	71
Referências Bibliográficas	72
Apêndice A – Linearização na Solução Equações não Lineares	75
Apêndice B - Regra de Armijo (Extrapolação Recursiva)	78
Apêndice C – Avaliação de dw_{ij}	80
Apêndice D - Programa Computacional Implementado em Linguagem C	83

Lista de Figuras

Figura 1 : Diagrama de Voronoi com seis polígonos convexos (volumes de controle)	23
Figura 2 : Componentes normais à face ij sobre os pontos nodais i e j	26
Figura 3 : Vizinhos nodais i e j e suas respectivas vizinhanças $r(k)$ e $s(k)$	34
Figura 4 : Parede superior deslizante sobre uma cavidade quadrada.	46
Figura 5 : Malha hexagonal com 6400 pontos nodais	49
Figura 6 : Velocidade u ao longo de uma linha vertical central, $Re = 100$ – Modelo 1	51
Figura 7 : Velocidade u ao longo de uma linha vertical central, $Re = 100$ – Modelo 2	52
Figura 8 : Velocidade v ao longo de uma linha horizontal central, $Re = 100$ – Modelo 1	53
Figura 9 : Velocidade v ao longo de uma linha horizontal central, $Re = 100$ – Modelo 2	54
Figura 10 : Velocidade u ao longo de uma linha vertical central, $Re = 100$ – Modelo 1	55
Figura 11 : Velocidade u ao longo de uma linha vertical central, $Re = 100$ – Modelo 2	56
Figura 12 : Velocidade v ao longo de uma linha horizontal central, $Re = 100$ – Modelo 1	57
Figura 13 : Velocidade v ao longo de uma linha horizontal central, $Re = 100$ – Modelo 2	58
Figura 14 : Velocidade u ao longo de uma linha vertical central, $Re = 1000$ – Modelo 1	60
Figura 15 : Velocidade u ao longo de uma linha vertical central, $Re = 1000$ – Modelo 2	61
Figura 16 : Velocidade v ao longo de uma linha horizontal central, $Re=1000$ – Modelo 1	62
Figura 17 : Velocidade v ao longo de uma linha horizontal central, $Re=1000$ – Modelo 2	63
Figura 18 : Velocidade u ao longo de uma linha vertical central, $Re = 1000$ – Modelo 1	64
Figura 19 : Velocidade u ao longo de uma linha vertical central, $Re = 1000$ – Modelo 2	65
Figura 20 : Velocidade v ao longo de uma linha horizontal central, $Re = 1000$ – Modelo 1	66
Figura 21 : Velocidade v ao longo de uma linha horizontal central, $Re=1000$ – Modelo 2	67
Figura 22 : Gráfico de s versus $\ f(y(s))\ $	78

Lista de Tabelas

Tabela 1: Expressões para φ , S^φ e Γ^φ , conforme a equação representativa	23
Tabela 2 : Legenda das avaliações das velocidades u e v na linearização de Newton	51
Tabela 3 : Linearização de Newton x linearização por separação, $Re = 100$	54
Tabela 4 : Linearização de Newton com Armijo x linearização por separação, $Re = 100$	58
Tabela 5 : Tipo de Linearização versus n° de iterações – $Re = 100$	59
Tabela 6 : Linearização de Newton x linearização por separação, $Re = 1000$	61
Tabela 7 : Linearização de Newton com Armijo x linearização por separação, $Re = 1000$	64
Tabela 8 : Tipo de Linearização versus n° de iterações – $Re = 1000$	68
Tabela 9 : Processo de separação a partir de um valor estimado, com atualização sucessiva	76
Tabela 10 : Processo de expansão em Série de Taylor em torno de x^*	77

Simbologia

Símbolo

Símbolo	Significado
$\ \ $	Norma Euclidiana.
a	Matriz de coeficientes.
A_{ij}	Coefficientes dos vizinhos do ponto nodal central i .
A_{p_i}	Coefficiente do ponto nodal i ;
b	Vetor de termos independentes.
c_p	Calor específico.
D	
d_n	Superfície de integração normal.
erro	Crítério de convergência.
$e_{x_{ij}}$	Componente do vetor normal unitário à interface ij , na direção x .
$e_{y_{ij}}$	Componente do vetor normal unitário à interface ij , na direção y .
\hat{e}_{ij}	Vetor normal a interface ij .
F	Fluxo de massa que atravessa a interface ik .
\hat{i}	Vetor base na direção x .
\hat{j}	Vetor base na direção y .
\bar{J}_{ij}	Fluxo advectivo-difusivo de ϕ no meio fluido na interface ij .
L_{ij}	Distância entre o ponto nodal central i e o ponto nodal vizinho j .
$\max(a,b)$	Maior valor entre a e b .
\hat{n}	Vetor normal a cada face do Volume de Controle (VC)
P	Pressão local.
$r(i)$	Índice de vizinhança do ponto nodal central i .
Re	Número de Reynolds (equação 67).
$s(j)$	Índice de vizinhança do ponto nodal vizinho j .
S_{ij}	Superfície de passagem entre os VC i e j .

S^u	Termo fonte da componente de velocidade na direção x.
S^v	Termo fonte da componente de velocidade na direção y.
t	Tempo.
u	Componente de velocidade na direção x.
U	Velocidade de movimento da parede superior deslizante da cavidade quadrada.
\vec{V}	Vetor velocidade.
v	Componente de velocidade na direção y.
x	Coordenada cartesiana na direção horizontal.
y	Coordenada cartesiana na direção vertical
w_i e w_j	Componente de velocidade no ponto i e em j, respectivamente.
W_{ij}	Componente normal de velocidade avaliada sobre a face ij.
z	Coordenada auxiliar na direção normal.
ΔS	Área das faces do volume de controle.
$\Delta \mathcal{V}_i$	Volume do Volume de Controle i.
Δt	Variação do tempo.

Gregos

Símbolo	Significado
Γ^ϕ	Coeficiente de difusão.
ϕ	Variável genérica.
μ	Viscosidade dinâmica do fluido.
ρ	Massa específica do fluido.
∂	Indica derivada parcial (operador del).
$\vec{\nabla}$	Operador vetorial Nabla.
∇P	Gradiente de pressão.
Σ	Indica somatório de elementos.

ξ	Precisão desejada no critério de parada.
\mathcal{V}	Volume.

Sub e Superscritos

i	Índice que indica o ponto nodal central.
ij	Conexão entre os pontos nodais i e seu vizinho j .
j	Índice que indica o ponto nodal vizinho.
k	Índice que indica os vizinhos do ponto nodal i ou j .
NV	Número de vizinhos.
*	Iteração anterior.

Abreviaturas

BI-CGSTAB	Biconjugate Gradient Stabilized.
CFD	Computacional Fluids Dynamics.
CG	Conjugate Gradient.
CGS	Conjugate Gradiente Square.
CPU	Unidade Central de Processamento.
DV	Diagrama de Voronoi.
EDP	Equação diferencial parcial.
GMRES	Generalized Minimal Residual.
MVF	Método dos Volumes Finitos.
NV	Número de vizinhos.
TDMA	Tridiagonal Matrix Algorithm.
UDS	UpWind Difference Scheme.
VC	Volume de Controle.

Capítulo 1 – Introdução

1.1 Considerações Iniciais

Os métodos numéricos usados na solução de problemas envolvendo Transferência de Calor e Mecânica dos Fluidos, denominada de “Computational Fluid Dynamics” (CFD), passaram a receber maior atenção por parte de analista numéricos com o advento de computadores que aliam alto desempenho com grande capacidade de armazenamento, pois o sistema de equações gerado após a discretização das equações consome parte significativa do tempo de CPU total, envolvendo, conseqüentemente, um grande custo computacional. Um método numérico eficiente pode conduzir a um baixo custo computacional e financeiro e pode dispensar a dispendiosas experimentações em laboratórios (caso o problema em questão já seja matematicamente conhecido e validado).

Em CFD, a maioria das aplicações envolvem métodos numéricos com escoamentos de fluidos em geometrias complexas, Ronzani e Nieckele (1995), permitindo o uso de malhas adaptáveis a geometria do problema. Dentre tais malhas, destacam-se as não-estruturadas geradas por Diagrama de Voronoi (DV), que são independentes de qualquer coordenada global, Taniguchi *et al.*, (1991) e Marcondes (1996). Uma das aplicações dos DV's é a geração de malhas não-estruturadas para a discretização numérica de equações diferenciais parciais (EDP's).

A discretização numérica das EDP's pode ser feita através do Método do Volumes Finitos (MVF), Patankar (1980). O Método dos Volumes Finitos consiste na divisão do domínio físico em um número de volumes de controle (VC) finitos não sobrepostos, onde promovem-se balanços locais das grandezas físicas envolvidas. Este método destaca-se em problemas envolvendo Transferência de Calor e Mecânica dos Fluidos regidos pelas equações de conservação da massa e da quantidade de movimento ou equações de Navier-Stokes.

As equações de Navier-Stokes são utilizadas para se resolver problemas de escoamentos de fluidos e, após realizada a discretização conforme o MVF, gera um sistema de equações não-lineares de segunda ordem que necessita de uma forma de linearização para ser resolvida como um sistema linear de equações.

A estrutura da matriz de coeficientes obtida na aproximação numérica é um fator de fundamental importância na escolha do método de solução do sistema linear de equações resultante. Tal matriz é, geralmente, diagonalmente dominante, o que garante a convergência por processos iterativos. Uma outra característica importante é o alto índice de esparsidade, o que exige grandes cuidados no armazenamento dos coeficientes utilizados.

O sistema linear de equações resultante sendo esparso e de alta ordem, reforça a necessidade de um método iterativo para sua solução. Uma tendência atual é investir em métodos de solução baseados em minimização de um funcional, cujo mínimo coincide com a própria solução do sistema, como por exemplo métodos baseados nos gradientes conjugados : CG (Conjugate Gradient), CGS (Conjugate Gradient Squared), BI-CGSTAB (Biconjugate Gradient Stabilized), dentre outros.

1.2 Linearização Através do Método de Newton

O sistema de equações gerados da discretização das EDP's de Navier-Stokes através do MVF é de segunda ordem e necessita de uma forma de linearização para que possa ser resolvida como um sistema linear de equações. Normalmente, o sistema de equações gerado da discretização é linearizado, seja na discretização de termos não-lineares, seja na linearização de termos fonte, possibilitando que a solução destas não linearidades possa ser obtida de forma iterativa. Neste caso, é recomendável o uso de fatores de sub-relaxação para amortecer variações bruscas causadas pelas não linearidades envolvidas.

No Método de Newton, a solução de um sistema não linear é obtida através da solução de uma seqüência de sistemas lineares que, por sua vez, é obtida a partir do sistema não linear, utilizando a expansão em Série de Taylor de 1ª ordem. Normalmente, a convergência do Método de Newton é quadrática, Faires e Burden (1993), embora nos contornos não seja possível manter essa mesma taxa de convergência.

Os tipos de linearização e relaxação usadas podem afetar significativamente a convergência do sistema de equações lineares.

A linearização via Método de Newton, bem como detalhes de implementação e aplicações podem ser encontradas em : Raithby (1986), D'Amico (1994), Johan (1990) e Putti (1995).

Discussões sobre linearização e solução do sistema de equações linear resultante podem ser encontradas em : Santos (1996), Beam e Warming (1976), Beam e Warming (1977), Briley e McDonald (1977), Briley e McDonald (1980), MacCormic (1985).

1.3 Objetivo

O presente trabalho visa, essencialmente, implementar nova linearização na discretização das equações de Navier-Stokes usando a expansão via Série de Taylor, conforme o método de Newton, com interpolação UpWind de 1ª ordem. O sistema de equações lineares gerado da discretização é resolvido através do CG (Conjugate Gradiente), conforme Mariani (1997).

O objetivo é otimizar o custo computacional envolvido na solução do sistema de equações lineares gerado da discretização.

1.4 Organização do Trabalho

Este capítulo apresenta uma breve descrição do conteúdo encontrado neste trabalho.

O capítulo dois contém uma descrição das etapas de discretização das equações de Navier-Stokes, segundo Cardoso (1997).

O capítulo 3 apresenta a implementação da discretização das equações de Navier-Stokes usando a expansão via Série de Taylor com interpolação UDS, conforme o método de Newton.

No capítulo 4 encontram-se os resultados obtidos com a implementação do presente trabalho. Tais resultados são analisados e comparados com os resultados obtidos por Ghia *et al.* (1982), Cardoso (1997) e por Mariani (1997).

O capítulo 5 contém as perspectivas para trabalhos futuros.

O Apêndice A apresenta um exemplo de linearização de uma equação à uma variável, comparando três métodos distintos.

O Apêndice B apresenta uma descrição sucinta da Regra de Armijo que foi introduzida no Método de Newton com o objetivo de obter estabilidade numérica do sistema de equações e, por consequência, aumentar o campo de convergência no processo iterativo.

O Apêndice C apresenta a forma como foi obtido o coeficiente d_{ij}^w utilizado no presente trabalho.

O Apêndice D apresenta o programa computacional implementado em linguagem C, utilizado para testar a linearização de Newton.

Capítulo 2 – Discretização das Equações de Navier-Stokes

2.1 Introdução

O Método dos Volumes Finitos (MVF) surgiu no estudo de métodos numéricos de problemas envolvendo transferência de calor e Mecânica dos Fluidos, Patankar (1980) e permitiu a simplificação das modelagens matemáticas por EDP's, no que se refere a implementação das condições de contorno. Também incorporou na sua metodologia uma série de informações originadas na física do problema.

O MVF consiste, basicamente, na integração volumétrica das equações diferenciais em pequenos volumes de controle (VC) finitos, gerando equações algébricas discretas e promovendo o balanço das grandezas físicas envolvidas nos VC's. Este método é mais consistente quando usamos as equações diferenciais na sua forma conservativa, o que permite estender a conservação de volumes elementares a níveis de volumes finitos.

A seguir é apresentada a discretização das equações de Navier-Stokes, pelo Método dos Volumes Finitos, segundo Cardoso (1997).

2.2 Discretização das Equações de Navier-Stokes

As equações de Navier-Stokes em coordenadas cartesianas na forma conservativa e que regem escoamentos de fluidos Newtonianos, Bejan (1984), são escritas da seguinte forma :

Conservação da Massa :

$$\frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} = 0 \quad (1a)$$

Conservação da quantidade de movimento do fluido na direção x :

$$\frac{\partial(\rho u)}{\partial t} + \frac{\partial(\rho uu)}{\partial x} + \frac{\partial(\rho vu)}{\partial y} = \frac{-\partial P}{\partial x} + \frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) + S^u \quad (1b)$$

Conservação da quantidade de movimento do fluido na direção y :

$$\frac{\partial(\rho v)}{\partial t} + \frac{\partial(\rho uv)}{\partial x} + \frac{\partial(\rho vv)}{\partial y} = \frac{-\partial P}{\partial y} + \frac{\partial}{\partial x} \left(\mu \frac{\partial v}{\partial x} \right) + \frac{\partial}{\partial y} \left(\mu \frac{\partial v}{\partial y} \right) + S^v \quad (1c)$$

onde :

u e v → são as componentes de velocidades nas direções x e y, respectivamente.

ρ → é a massa específica do fluido.

μ → é a viscosidade dinâmica.

S^u → é o termo fonte de geração de u nos VC's.

S^v → é o termo fonte de geração de v nos VC's.

t → é o tempo.

P → é a pressão local.

Estas equações podem ser escritas na forma vetorial, conforme equação (2), para facilitar o trabalho de integração nos Volumes de Controle do Método dos Volumes Finitos. Considerando, genericamente, a equação geral de transporte envolvendo advecção/difusão de uma grandeza ϕ genérica em regime transiente (fenômeno dependente do tempo) tem-se :

$$\frac{\partial(\rho\phi)}{\partial t} + \nabla \cdot \vec{J} = S^\phi \quad (2)$$

$$\vec{J} = \rho \vec{V}\phi - \Gamma^\phi \nabla\phi \quad (3)$$

onde :

ρ → é a massa específica do fluido;

$\varphi \rightarrow$ variável genérica;

$\vec{J} \rightarrow$ é o fluxo advectivo-difusivo de φ no meio fluido;

$\vec{V} \rightarrow$ é o vetor velocidade;

$\vec{V} = u\hat{i} + v\hat{j} \rightarrow$ vetor velocidade no sistema de coordenadas cartesianas com versores \hat{i} e \hat{j} para as direções x e y, respectivamente;

$\Gamma^\varphi \rightarrow$ é o coeficiente de difusão de φ no meio;

$S^\varphi \rightarrow$ é o termo fonte de geração de φ nos VC's;

$\nabla = \frac{\partial}{\partial x}\hat{i} + \frac{\partial}{\partial y}\hat{j} \rightarrow$ é o operador gradiente do sistema de coordenadas correspondente.

No caso específico da equação de conservação da massa, equação (1a), da quantidade de movimento em fluidos, equações (1b) e (1c), deve-se ter um valor específico de φ , S^φ e Γ^φ , conforme a tabela a seguir :

Equações	φ	S^φ	Γ^φ
Conservação da massa	1	0	0
Conservação da quantidade de movimento na direção x	U	$-(\partial p / \partial x)$	μ
Conservação da quantidade de movimento na direção y	V	$-(\partial p / \partial y)$	μ

Tabela 1: Expressões para φ , S^φ e Γ^φ , conforme a equação representativa

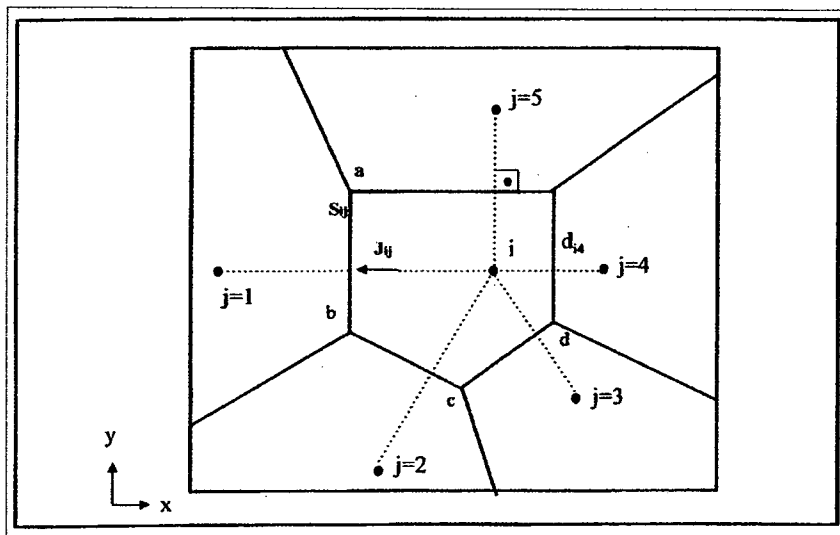


Figura 1: Diagrama de Voronoi com seis polígonos convexos (volumes de controle)

No Método dos Volumes Finitos as equações diferenciais parciais são integradas em cada um dos VC gerados no domínio computacional discretizado, conforme o exemplo dado pela figura 1, acima.

Integra-se a equação geral de transporte, equação (2), envolvendo advecção/difusão de uma grandeza φ genérica em cada VC i :

$$\iiint \frac{\partial(\rho\varphi)}{\partial t} d\mathcal{V} + \iiint (\vec{\nabla} \cdot \vec{J}) d\mathcal{V} = \iiint S^{\circ} d\mathcal{V} \quad (4)$$

Integrando termo a termo a equação (4) em cada VC i , tem-se para o 1° termo a integração do termo transiente que é tratado com aproximação de 1ª ordem a frente. Avaliando φ em t e $t+\Delta t$ tem-se :

$$\iiint \frac{\partial(\rho\varphi)}{\partial t} d\mathcal{V} = \text{Ap}_i^{\circ} (\varphi_i - \varphi_i^{\circ}) \quad (5)$$

onde :

$$\text{Ap}_i^{\circ} = \frac{\rho_i \Delta \mathcal{V}_i}{\Delta t} ;$$

$\Delta \mathcal{V}_i \rightarrow$ volume do VC de um prisma poligonal convexo (profundidade unitária) gerado por DV.

$\varphi_i^{\circ} \rightarrow$ grandeza φ genérica avaliada no instante anterior (t);

$\varphi_i \rightarrow$ valor de φ avaliada no instante $t + \Delta t$.

Para o 2° termo tem-se a avaliação do fluxo líquido de J que atravessa o VC. Utilizando o Teorema da Divergência tem-se :

$$\iiint_{\mathcal{V}} (\vec{\nabla} \cdot \vec{J}) d\mathcal{V} = \iint_{S_n} (\vec{J} \cdot \hat{n}) dS = \sum_{j=1}^{NV} J_{ij} S_{ij} \quad (6)$$

onde :

$S_n \rightarrow$ é a superfície de integração.

$\hat{n} \rightarrow$ é o vetor normal (externo) a cada face do VC i .

$J_{ij} \rightarrow$ é a componente normal de \vec{J} em cada face ij , figura 1.

$S_{ij} \rightarrow$ superfície (aresta) entre os pontos i e j , figura 1.

$j \rightarrow$ índice de vizinho de i ;

$NV \rightarrow$ número de vizinhos de i .

$J_{ij}S_{ij}$ representa o fluxo líquido que passa na face ij . Considerando W_{ij} como a componente normal de velocidade avaliado sobre a face ij , que é responsável pela advecção de ϕ através da face ij , e z como sendo a direção normal a cada face ij (coordenada auxiliar), tem-se J_{ij} , conforme Cardoso (1997), dado por :

$$J_{ij} = \left[\rho W \phi - \Gamma^\phi \frac{\partial \phi}{\partial z} \right]_{ij} = \rho_{ij} W_{ij} \phi_{ij} - \Gamma_{ij}^\phi \left(\frac{\partial \phi}{\partial z} \right)_{ij} \quad (7)$$

onde as variáveis de índice ij são avaliadas sobre a face ij através de média adequadas a cada caso (Patankar, 1980), conforme segue :

$$\rho_{ij} = \frac{\rho_i + \rho_j}{2} \text{ (Média Aritmética);}$$

$$\Gamma_{ij}^\phi = \frac{2\Gamma_i\Gamma_j}{\Gamma_i + \Gamma_j} \text{ (Média Harmônica);}$$

$$\left. \frac{\partial \phi}{\partial z} \right|_{ij} = \frac{\phi_i - \phi_j}{L_{ij}} \rightarrow \text{aproximação por diferença central de 1ª ordem (} E_T = O^2 \text{);}$$

No presente trabalho, adota-se a avaliação de W_{ij} através da média aritmética simples entre as componentes normais projetadas na direção ij , equação (8). Esta aproximação tem sido tradicionalmente utilizada e gera aproximações numericamente mais estáveis, Patankar (1980) e Maliska (1995).

$$W_{ij} = \frac{(w_i)_{ij} + (w_j)_{ij}}{2} \quad (8)$$

$(w_i)_{ij}$ e $(w_j)_{ij}$ são as componentes normais à face ij avaliados sobre os pontos nodais i e j , conforme figura abaixo :

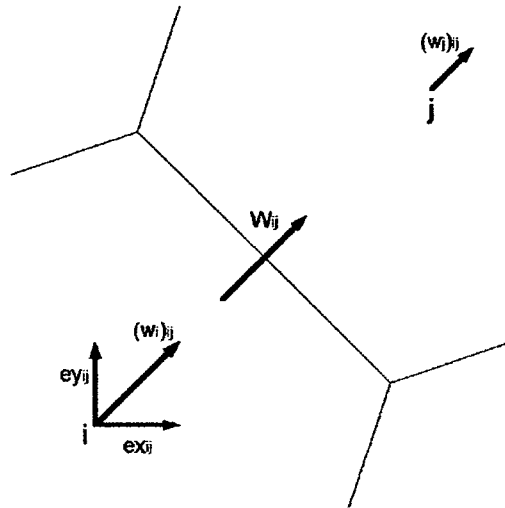


Figura 2 : Componentes normais à face ij sobre os pontos nodais i e j

$$(w_i)_{ij} = u_i ex_{ij} + v_i ey_{ij} \quad (9)$$

$$(w_j)_{ij} = u_j ex_{ij} + v_j ey_{ij} \quad (10)$$

onde :

$z \rightarrow$ é a coordenada auxiliar na direção normal.

Alternativamente, W_{ij} pode ser reescrito substituindo as equações (9) e (10) na equação (8), conforme segue :

$$W_{ij} = \frac{(u_i ex_{ij} + v_i ey_{ij} + u_j ex_{ij} + v_j ey_{ij})}{2}$$

ou

$$W_{ij} = \left(\frac{u_i + u_j}{2} \right) \mathbf{ex}_{ij} + \left(\frac{v_i + v_j}{2} \right) \mathbf{ey}_{ij}$$

Logo :

$$W_{ij} = u_{ij} \mathbf{ex}_{ij} + v_{ij} \mathbf{ey}_{ij} \quad (11)$$

onde :

$$u_{ij} = \frac{u_i + u_j}{2} \rightarrow \text{componente horizontal de } \vec{V} \text{ avaliado sobre a face } ij;$$

$$v_{ij} = \frac{v_i + v_j}{2} \rightarrow \text{componente vertical de } \vec{V} \text{ avaliado sobre a face } ij;$$

Ou seja, para avaliar u_{ij} e v_{ij} , respectivamente, na interface ij podemos também adotar a média aritmética.

Finalmente, para o 3º termo tem-se uma integração pela média, adotando S_i^φ médio do VC como valor constante :

$$\iiint S^\varphi d\mathcal{V} = S_i^\varphi \Delta \mathcal{V}_i \quad (12)$$

onde :

$\varphi \rightarrow$ é o valor médio de S^φ no VC i ;

$\Delta \mathcal{V}_i \rightarrow$ é o volume do VC i .

Reunindo os três termos gera-se a seguinte equação discretizada:

$$Ap_i^0 (\varphi_i - \varphi_i^0) + \sum_{j=1}^{NV(i)} \left[\rho W \varphi - \Gamma^\varphi \frac{\partial \varphi}{\partial z} \right]_{ij} S_{ij} = S_i^\varphi \Delta \mathcal{V}_i \quad (13)$$

Na equação (13) tem-se o termo $\rho W\phi$ que é não linear, quando ϕ assume o valor de alguma componente de velocidade (u ou v). Neste caso, tem-se um produto entre as componentes presentes em W e em ϕ .

Capítulo 3 – Linearização Conforme o Método de Newton.

3.1 Introdução

Após realizado o trabalho de discretização das equações de Navier-Stokes, o sistema de equações resultante contém termos não lineares de segunda ordem que necessitam de um processo de linearização para que possam ser resolvidas como um sistema linear de equações e, também, através de um método iterativo pois normalmente o sistema de equações é esparso e de alta ordem, o que envolve um grande custo computacional na solução do sistema de equações lineares gerado da discretização.

O tipo de linearização utilizada pode afetar significativamente a convergência do sistema de equações resultante. O presente trabalho tem por objetivo avaliar a linearização conforme o método de Newton aplicado a um sistema não linear. A escolha pela linearização através do Método de Newton deve-se a sua convergência quadrática, Faires e Burden (1993), embora nos contornos não seja possível manter essa mesma taxa. No Método de Newton, um sistema de equações não linear é resolvido através de uma seqüência de sistemas linearizados, que são obtidos a partir do sistema não linear resultante da discretização. A seguir apresenta-se a implementação da discretização das equações de Navier-Stokes usando a linearização conforme o método de Newton com interpolação UDS de 1ª ordem.

3.2 Discretização das Equações de Navier-Stokes usando a Linearização conforme o Método Newton

O fluxo advectivo-difusivo de uma grandeza ϕ genérica através de uma superfície é calculado através da seguinte equação :

$$J_{ij} = \left[\rho W\phi - \Gamma^{\phi} \frac{\partial \phi}{\partial z} \right]_{ij}$$

Quando toma-se $\phi = u$ ou $\phi = v$ tem-se uma não linearidade envolvida no termo $\rho \mathbf{W} \phi$, de modo que $\rho \mathbf{W} \phi$ deve ser linearizado para que se possa resolver o sistema algébrico gerado. Tradicionalmente, W_{ij} é obtido a partir de valores de velocidade da iteração anterior (denotados com *) e ϕ é avaliado na nova iteração gerando a seguinte expressão (vide Apêndice C), gerando a linearização por Separação :

$$[\rho \mathbf{W} \phi]_{ij} = \rho_{ij} W_{ij}^* \phi_{ij} \quad (14)$$

O objetivo deste trabalho consiste em avaliar $\rho \mathbf{W} \phi$ usando a linearização via Série de Taylor, conforme o Método de Newton aplicado a um sistema não linear (vide Apêndice A).

Escrevendo $\phi_{ij} = u_{ij}$ no termo advectivo $\rho_{ij} W_{ij} \phi_{ij}$, tem-se o seguinte termo :

$$G(u_{ij}, v_{ij}) = \rho_{ij} W_{ij} u_{ij} \quad (15)$$

Substituindo a equação (11) na equação (15) tem-se :

$$G(u_{ij}, v_{ij}) = \rho_{ij} (u_{ij} \mathbf{e}_x + v_{ij} \mathbf{e}_y) u_{ij} \quad (16)$$

Linearizando o termo $G(u_{ij}, v_{ij})$, equação (16), via Série de Taylor de 1ª ordem tem-se :

$$G(u_{ij}, v_{ij}) \cong G(u_{ij}^*, v_{ij}^*) + \left. \frac{\partial G}{\partial u_{ij}} \right|_{u_{ij}^*, v_{ij}^*} (u_{ij} - u_{ij}^*) + \left. \frac{\partial G}{\partial v_{ij}} \right|_{u_{ij}^*, v_{ij}^*} (v_{ij} - v_{ij}^*) \quad (17)$$

$$\left. \frac{\partial G}{\partial u_{ij}} \right|_{u_{ij}^*, v_{ij}^*} = \rho_{ij} (u_{ij}^* \mathbf{e}_x + v_{ij}^* \mathbf{e}_y) + \rho_{ij} \mathbf{e}_x u_{ij}^* = \rho_{ij} W_{ij}^* + \rho_{ij} \mathbf{e}_x u_{ij}^*$$

$$\left. \frac{\partial G}{\partial v_{ij}} \right|_{u_{ij}^*, v_{ij}^*} = \rho_{ij} \mathbf{e}_y u_{ij}^*$$

Logo :

$$G(u_{ij}, v_{ij}) = \rho_{ij} W_{ij}^* u_{ij}^* + (\rho_{ij} W_{ij}^* + \rho_{ij} \text{ex}_{ij} u_{ij}^*)(u_{ij} - u_{ij}^*) + (\rho_{ij} \text{ey}_{ij} u_{ij}^*)(v_{ij} - v_{ij}^*) \quad (18)$$

Note que quando u_{ij} convergir tem-se $u_{ij} = u_{ij}^*$, logo a equação (18) assume a mesma linearização por separação, conforme a equação (14).

A equação completa do fluxo advectivo-difusivo, equação (7), para $\phi_{ij} = u_{ij}$ assume a seguinte forma :

$$J_{ij} = \left(\rho W \phi - \Gamma \frac{\partial \phi}{\partial z} \right)_{ij} = \left(\rho_{ij} W_{ij} u_{ij} - \mu_{ij} \frac{\partial u}{\partial z} \Big|_{ij} \right) = \left(G(u_{ij}, v_{ij}) - \mu_{ij} \frac{\partial u}{\partial z} \Big|_{ij} \right) \quad (19)$$

onde o termo difusivo é avaliado por diferença central, conforme segue:

$$\frac{\partial u}{\partial z} \Big|_{ij} = \frac{u_j - u_i}{L_{ij}}$$

e o termo $G(u_{ij}, v_{ij}) = \rho_{ij} W_{ij} u_{ij}$ é obtido através da substituição da equação (18) na equação (19). Logo, tem-se :

$$J_{ij} = \rho_{ij} \left\{ u_{ij}^* \left[\text{ex}_{ij} (u_{ij} - u_{ij}^*) + \text{ey}_{ij} (v_{ij} - v_{ij}^*) \right] + v_{ij}^* \text{ey}_{ij} u_{ij} \right\} - \mu_{ij} \left(\frac{u_j - u_i}{L_{ij}} \right) \quad (20)$$

Adotando a resolução segregada do sistema de equações, resolve-se u_i separadamente de v_i e nesta equação despreza-se a variação de v fazendo $v_{ij} = v_{ij}^*$ na equação (20).

Para $\phi_{ij} = v_{ij}$ no termo advectivo, equação (14), tem-se o termo :

$$H(u_{ij}, v_{ij}) = \rho_{ij} W_{ij} v_{ij} \quad (21)$$

Então, substituindo a equação (11) na equação (21) tem-se :

$$H(u_{ij}, v_{ij}) = \rho_{ij}(u_{ij} \mathbf{e}_x + v_{ij} \mathbf{e}_y) v_{ij} \quad (22)$$

Analogamente a $G(u_{ij}, v_{ij})$, promovendo a linearização de $H(u_{ij}, v_{ij})$, dada pela equação (21), através da expansão em Série de Taylor de 1ª ordem, tem-se :

$$H(u_{ij}, v_{ij}) \cong H(u_{ij}^*, v_{ij}^*) + \left. \frac{\partial H}{\partial u_{ij}} \right|_{u_{ij}^*, v_{ij}^*} (u_{ij} - u_{ij}^*) + \left. \frac{\partial H}{\partial v_{ij}} \right|_{u_{ij}^*, v_{ij}^*} (v_{ij} - v_{ij}^*) \quad (23)$$

$$\left. \frac{\partial H}{\partial u_{ij}} \right|_{u_{ij}^*, v_{ij}^*} = \rho_{ij} \mathbf{e}_x v_{ij}^*$$

$$\left. \frac{\partial H}{\partial v_{ij}} \right|_{u_{ij}^*, v_{ij}^*} = \rho_{ij}(u_{ij}^* \mathbf{e}_x + v_{ij}^* \mathbf{e}_y) + \rho_{ij} \mathbf{e}_y u_{ij}^* = \rho_{ij} \mathbf{W}_{ij}^* + \rho_{ij} \mathbf{e}_y u_{ij}^*$$

Logo :

$$H(u_{ij}, v_{ij}) = \rho_{ij} \mathbf{W}_{ij}^* v_{ij} + (\rho_{ij} \mathbf{e}_x v_{ij}^*)(u_{ij} - u_{ij}^*) + (\rho_{ij} \mathbf{W}_{ij}^* + \rho_{ij} \mathbf{e}_y u_{ij}^*)(v_{ij} - v_{ij}^*) \quad (24)$$

A equação completa do fluxo advectivo/difusivo, equação (7), para $\phi_{ij} = v_{ij}$ assume a seguinte forma :

$$J_{ij} = \left(\rho \mathbf{W} \phi - \Gamma \frac{\partial \phi}{\partial z} \right)_{ij} = \left(\rho_{ij} \mathbf{W}_{ij} v_{ij} - \mu_{ij} \frac{\partial v}{\partial z} \right)_{ij} = \left(H(u_{ij}, v_{ij}) - \mu_{ij} \frac{\partial v}{\partial z} \right)_{ij} \quad (25)$$

onde o termo difusivo é avaliado por diferença central, conforme segue :

$$\left. \frac{\partial v}{\partial z} \right|_{ij} = \frac{v_j - v_i}{L_{ij}}$$

e o termo $\rho_{ij} \mathbf{W}_{ij} v_{ij}$ é obtido através da substituição da equação (24) na equação (25). Logo, tem-se :

$$J_{ij} = \rho_{ij} \left\{ v_{ij}^* \left[ex_{ij} (u_{ij} - u_{ij}^*) + ey_{ij} (2v_{ij} - v_{ij}^*) \right] + u_{ij}^* ex_{ij} v_{ij} \right\} - \mu_{ij} \left(\frac{v_j - v_i}{L_{ij}} \right) \quad (26)$$

Adotando a resolução segregada do sistema de equações, resolve-se v_i separadamente de u_i e despreza-se a variação de u nesta equação fazendo $u_{ij} = u_{ij}^*$ na equação (26).

3.3 Interpolação Genérica Para u_{ik} e v_{ik} , entre os Pontos Nodais i e seus vizinhos $r(k)$

A velocidade W em uma interface ik genérica, conforme figura 3, está sendo avaliado pela equação (11), a seguir :

$$W_{ik} = u_{ik} ex_{ik} + v_{ik} ey_{ik} \quad (26)$$

Para avaliar u e v na interface ik adota-se o esquema de interpolação UDS (UpWind Difference Scheme), gerando a seguinte forma de interpolação linear para u_{ik} :

$$u_{ik} = u_i \delta_{ik}^+ + u_{r(k)} \delta_{ik}^- \quad (28)$$

onde :

$$\delta_{ik}^+ = \max \frac{(0, F_{ik})}{|F_{ik}|} = \Psi (F_{ik}) \quad (29)$$

$$\delta_{ik}^- = \max \frac{(0, -F_{ik})}{|F_{ik}|} = \Psi (-F_{ik}) \quad (30)$$

$$\Psi (F_{ik}) = \begin{cases} 1, & \text{se } F_{ik} > 0 \\ 0, & \text{se } F_{ik} \leq 0 \end{cases} \quad (31)$$

onde F_{ik} é o fluxo de massa que atravessa cada interface ik dado por :

$$F_{ik} = \rho_{ik} W_{ik} S_{ik} \quad (32)$$

Ou seja :

$$\begin{cases} \text{se } F_{ik} > 0 \Rightarrow u_{ik} = u_i \\ \text{se } F_{ik} \leq 0 \Rightarrow v_{ik} = v_i \end{cases}$$

Alternativamente, u_{ik} pode ser avaliado por média aritmética simples, também chamada de Diferença Central, como segue :

$$u_{ik} = \frac{(u_i + u_{r(k)})}{2} \quad (33)$$

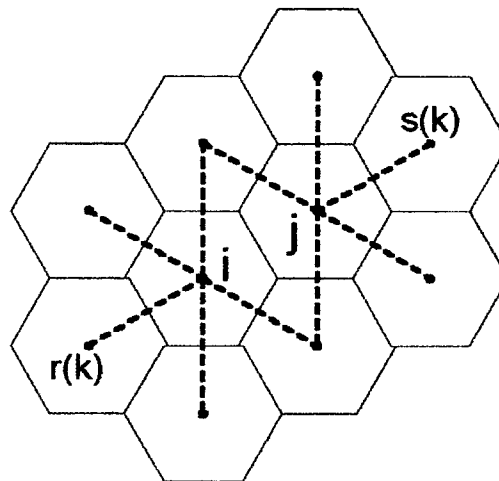


Figura 3 : Vizinhos nodais i e j e suas respectivas vizinhanças $r(k)$ e $s(k)$

Assim, se $W_{ik}^* > 0$ tem-se que $\delta_{ik}^+ = 1$ e $\delta_{ik}^- = 0$, gerando $u_{ik} = u_i$. Por outro lado, se $W_{ik}^* < 0$ tem-se que $\delta_{ik}^+ = 0$ e $\delta_{ik}^- = 1$, gerando $u_{ik} = u_{r(k)}$.

Analogamente, temos para v_{ik} :

$$v_{ik} = v_i \delta_{ik}^+ + v_{r(k)} \delta_{ik}^- \quad (34)$$

onde :

$$\delta_{ik}^+ = \max \frac{(0, F_{ik})}{|F_{ik}|} = \Psi (F_{ik})$$

e

$$\delta_{ik}^- = \max \frac{(0, -F_{ik})}{|F_{ik}|} = \Psi (-F_{ik})$$

onde $\Psi (F_{ik})$ é dado pela equação (31).

Alternativamente, v_{ik} pode ser avaliado por média aritmética simples, também chamada de Diferença Central, como segue:

$$v_{ik} = \frac{(v_i + v_{r(k)})}{2} \quad (35)$$

3.4 Obtenção das Equações do Movimento Discretizadas com Interpolação UDS para u_{ik}

Substituindo-se a equação do fluxo advectivo-difusivo para $\phi_{ik} = u_{ik}$, equação (20) na equação (13), conforme linearização aplicada pelo método de Newton, tem-se :

$$Ap_i^0(u_i - u_i^0) + \sum_{k=1}^{NV(i)} \left\{ \beta_{ik}^* \left\{ u_{ik}^* \left[ex_{ik} (u_{ik} - u_{ik}^*) + ey_{ik} (v_{ik} - v_{ik}^*) \right] + v_{ik}^* ey_{ik} u_{ik} \right\} - \mu_{ik} \left(\frac{u_{r(k)} - u_i}{L_{ik}} \right) \right\} S_{ik} = S_i^u \Delta \theta_i \quad (36)$$

Substituindo as equações (28) e (34) na equação (36) e reorganizando as variáveis, tem-se :

$$Ap_i^u u_i + \sum_{k=1}^{NV(i)} \beta_{ik}^u \delta_{ik}^- v_i = \sum_{k=1}^{NV(i)} (A_{ik}^u u_{r(k)} - \beta_{ik}^u \delta_{ik}^- v_{r(k)}) + \Omega_{ik}^u(i) + Ap_i^0 u_i^0 \quad (37)$$

onde :

$$\alpha_{ik}^u = \rho_{ik} u_{ik}^* \text{ex}_{ik} S_{ik}$$

$$\beta_{ik}^u = \rho_{ik} u_{ik}^* \text{ey}_{ik} S_{ik}$$

$$A_{ik}^u = -(\alpha_{ik}^u + F_{ik}) \delta_{ik}^- + D_{ik}$$

$$Ap_i^u = A_{ik}^u u_{r(k)} - \beta_{ik}^u \delta_{ik}^- v_{r(k)}$$

$$\Omega_{ik}^u(i) = S_i^u \Delta \vartheta_i + \sum_{k=1}^{NV(i)} (\alpha_{ik}^u u_{ik}^* + \beta_{ik}^u v_i^*)$$

Para desconsiderar os efeitos da variação de v_i na equação de u_i , fazemos

$\beta_{ik}^u = 0$ diretamente na equação (37). Logo :

$$Ap_i^u u_i = \sum_{k=1}^{NV(i)} A_{ik}^u u_{r(k)} + b_i^u \tag{38}$$

onde :

$$b_i^u = \Omega_{ik}^u(i) + Ap_i^0 u_i^0$$

$$\Omega_{ik}^u(i) = S_i^u \Delta \vartheta_i + \sum_{k=1}^{NV(i)} (\alpha_{ik}^u u_{ik}^*)$$

$$Ap_i^u = Ap_i^0 + \sum_{k=1}^{NV(i)} (\alpha_{ik}^u + F_{ik}) \delta_{ik}^- + D_{ik}^u$$

$$D_{ik}^u = \mu_{ik} \left(\frac{S_{ik}}{L_{ik}} \right)$$

$$F_{ik} = \rho_{ik} W_{ik}^* S_{ik} \rightarrow \text{fluxo convectivo}$$

$$A_{ik}^u = -(\alpha_{ik}^u + F_{ik})\delta_{ik} + D_{ik}$$

$u_i^0 \rightarrow$ velocidade no ponto i no instante de tempo anterior.

u_{ik}^* , v_{ik}^* e W_{ik}^* são avaliados na interface ik baseado em média aritmética simples entre i e $r(k)$.

Para a equação da conservação da massa tem-se a seguinte integração no VC :

$$\iiint_{\mathcal{G}} (\nabla \cdot \vec{J}) d\mathcal{G} = 0$$

Pelo Teorema da Divergência reduz-se a integral de volume em uma integral de superfície :

$$\iiint_{\mathcal{G}} (\nabla \cdot \vec{J}) d\mathcal{G} = \int_S \rho W dS = \sum_{k=1}^{NV(i)} \rho_{ik} W_{ik} S_{ik} = \sum_{k=1}^{NV(i)} F_{ik} = 0$$

Logo, a soma discreta de todos os fluxos F_{ik} em cada superfície S_{ik} do VC deve ser nula :

$$\sum_{k=1}^{NV(i)} F_{ik} = 0 \quad (39)$$

O fluxo F_{ik} pode ser escrito como :

$$F_{ik} = \rho_{ik} (u_{ik}^* ex_{ik} + v_{ik}^* ey_{ik}) S_{ik}$$

$$F_{ik} = \rho_{ik} u_{ik}^* ex_{ik} S_{ik} + \rho_{ij} v_{ik}^* ey_{ik} S_{ik}$$

$$F_{ik} = \alpha_{ik}^u + \varepsilon_{ik}^u \quad (40)$$

onde :

$$\alpha_{ik}^u = \rho_{ik} u_{ik}^* \mathbf{e}_{x_{ik}} S_{ik}$$

$$\varepsilon_{ik}^u = \rho_{ik} v_{ik}^* \mathbf{e}_{y_{ik}} S_{ik}$$

Para a equação de u_i , tem-se :

$$u_i = \delta_{ik}^+ u_i + \delta_{ik}^- u_i \quad (41)$$

Rescrevendo a equação (39) e multiplicando pela expressão de u_i , dada pela equação (41), tem-se a seguinte equação :

$$\sum_{k=1}^{NV(i)} (\alpha_{ik}^u + \varepsilon_{ik}^u) (\delta_{ik}^+ u_i + \delta_{ik}^- u_i) = 0 \quad (42)$$

Subtraindo à equação da continuidade discretizada multiplicada por u_i , equação (42), da equação (37) e rescrevendo os termos tem-se :

$$\left[Ap_i^0 + \sum_{k=1}^{NV(i)} \{ (\alpha_{ik}^u + F_{ik}) \delta_{ik}^- + D_i^u \} - \alpha_{ik}^u \right] u_i = \sum_{k=1}^{NV(i)} \{ (\alpha_{ik}^u + F_{ik}) \delta_{ik}^- + D_i^u \} u_{r(k)} + b_i^u \quad (43)$$

Logo :

$$Ap_i^u u_i = \sum_{k=1}^{NV(i)} A_{ik}^u u_{r(k)} + b_i^u \quad (44)$$

onde :

$$b_i^u = \Omega_{ik}^u(i) + Ap_i^0 u_i^0$$

$$\Omega_{ik}^u(i) = S_i^u \Delta \mathcal{G}_i + \sum_{k=1}^{NV(i)} \alpha_{ik}^u u_{ik}^*$$

$$A_{ik}^u = -(\alpha_{ik}^u + F_{ik}) \delta_{ik}^- + D_{ik}^u$$

$$Ap_i^u = Ap_i^0 + \sum_{k=1}^{NV(i)} A_{ik}^u - \alpha_i^u$$

Lembrando que para $\varphi = u$ tem-se um termo de pressão embutido no termo fonte S_i^u , Cardoso (1997). Logo :

$$Ap_i^u u_i = \sum_{k=1}^{NV(i)} A_{ik}^u u_{r(k)} + b_i^u - \Delta \mathcal{G}_i (\nabla P)_i^x \quad (45)$$

e

$$Ap_j^u u_j = \sum_{k=1}^{NV(i)} A_{jk}^u u_{s(k)} + b_j^u - \Delta \mathcal{G}_j (\nabla P)_j^x \quad (46)$$

3.5 Obtenção das Equações do Movimento Discretizadas c/ Interpolação UDS para v_{ik}

Substituindo-se a equação do fluxo advectivo-difusivo para $\varphi_{ik} = v_{ik}$, equação (26) na equação (13), conforme linearização aplicada no método de Newton, tem-se :

$$Ap_i^0 (v_i - v_i^0) + \sum_{k=1}^{NV(i)} \left\{ \rho_{ik} \left\{ v_{ik}^* \left[ex_{ik} (u_{ik} - u_{ik}^*) + ey_{ik} (2v_{ik} - v_{ik}^*) \right] + u_{ik}^* ex_{ik} v_{ik} \right\} - \mu_{ik} \left(\frac{v_{r(k)} - v_i}{L_{ik}} \right) \right\} S_{ik} = S_i^v \Delta \mathcal{G}_i \quad (47)$$

Substituindo as equações (28) e (34) na equação (47) e reorganizando as variáveis, tem-se :

$$Ap_i^v v_i + \sum_{k=1}^{NV(i)} \beta_{ik}^v \delta_{ik}^- u_i = \sum_{k=1}^{NV(i)} \left(A_{ik}^v v_{r(k)} - \beta_{ik}^v \delta_{ik}^- u_{r(k)} \right) + \Omega_{ik}^v (i) + Ap_i^0 v_i^0 \quad (48)$$

onde :

$$\alpha_{ik}^v = \rho_{ik} v_{ik}^* ey_{ik} S_{ik}$$

$$\beta_{ik}^v = \rho_{ik} v_{ik}^* ex_{ik} S_{ik}$$

$$A_{ik}^v = -(\alpha_{ik}^v + F_{ik})\delta_{ik}^- + D_{ik}$$

$$Ap_i^v = A_{ik}^v v_{r(k)} - \beta_{ik}^v \delta_{ik}^- u_{r(k)}$$

$$\Omega_{ik}^v(i) = S_i^v \Delta \mathcal{G}_i + \sum_{k=1}^{NV(i)} (\alpha_{ik}^v v_{ik}^* + \beta_{ik}^v u_i^*)$$

Para desconsiderar os efeitos da variação de u_i na equação de v_i , fazemos $\beta_{ik}^v = 0$ diretamente na equação (48). Logo :

$$Ap_i^v v_i = \sum_{k=1}^{NV(i)} A_{ik}^v v_{r(k)} + b_i^v \quad (49)$$

onde :

$$b_i^v = \Omega_{ik}^v(i) + Ap_i^0 v_i^0$$

$$\Omega_{ik}^v(i) = S_i^v \Delta \mathcal{G}_i + \sum_{k=1}^{NV(i)} (\alpha_{ik}^v v_{ik}^* + \beta_{ik}^v u_i^*)$$

$$Ap_i^v = Ap_i^0 + \sum_{k=1}^{NV(i)} (\alpha_{ik}^v + F_{ik})\delta_{ik}^+ + D_{ik}^v$$

$$D_{ik}^v = \mu_{ik} \left(\frac{S_{ik}}{L_{ik}} \right)$$

$$F_{ik} = \rho_{ik} W_{ik}^* S_{ik} \rightarrow \text{fluxo convectivo}$$

$$A_{ik}^v = -(\alpha_{ik}^v + F_{ik})\delta_{ik}^- + D_{ik}$$

$v_i^0 \rightarrow$ velocidade no ponto i no instante de tempo anterior.

u_{ik}^* , v_{ik}^* e W_{ik}^* são avaliados na interface ik baseado na média aritmética simples entre i e $r(k)$.

Para a equação da conservação da massa tem-se a seguinte integração no VC do $d\mathcal{V}$:

$$\iiint_{\mathcal{V}} (\nabla \cdot \vec{J}) d\mathcal{V} = 0$$

Pelo Teorema da Divergência reduz-se a integral de volume em uma integral de superfície :

$$\iiint_{\mathcal{V}} (\nabla \cdot \vec{J}) d\mathcal{V} = \int_S \rho W dS = \sum_{k=1}^{NV(i)} \rho_{ik} W_{ik} S_{ik} = \sum_{k=1}^{NV(i)} F_{ik} = 0$$

Logo, a soma discreta de todos os fluxos F_{ik} em cada superfície S_{ik} do VC deve ser nula :

$$\sum_{k=1}^{NV(i)} F_{ik} = 0 \tag{50}$$

O fluxo F_{ik} pode ser escrito como :

$$F_{ik} = \rho_{ik} (u_{ik}^* ex_{ik} + v_{ik}^* ey_{ik}) S_{ik}$$

$$F_{ik} = \rho_{ik} u_{ik}^* ex_{ik} S_{ik} + \rho_{ik} v_{ik}^* ey_{ik} S_{ik}$$

$$F_{ik} = \alpha_{ik}^v + \epsilon_{ik}^v \tag{51}$$

Para a equação de v_i , tem-se :

$$v_i = \delta_{ik}^+ v_i + \delta_{ik}^- v_i \tag{52}$$

Rescrevendo a equação (50) e multiplicando pela expressão de v_i , dada pela equação (52), tem-se a seguinte equação :

$$\sum_{k=1}^{NV(i)} (\alpha_{ik}^v + \varepsilon_{ik}^v) (\delta_{ik}^+ v_i + \delta_{ik}^- v_i) = 0 \quad (53)$$

onde :

$$\alpha_{ik}^v = \rho_{ik} v_{ik}^* e y_{ik} S_{ik}$$

$$\varepsilon_{ik}^v = \rho_{ik} u_{ik}^* e x_{ik} S_{ik}$$

Subtraindo à equação da continuidade discretizada multiplicada por v_i , equação (53), da equação (48) e rescrevendo os termos tem-se :

$$\left[A p_i^0 + \sum_{k=1}^{NV(i)} \{ (\alpha_{ik}^v + F_{ik}) \delta_{ik}^- + D_i^v \} - \alpha_{ik}^v \right] v_i = \sum_{k=1}^{NV(i)} \{ (\alpha_{ik}^v + F_{ik}) \delta_{ik}^v + D_i^v \} v_{r(k)} + b_i^v \quad (54)$$

Logo :

$$A p_i^v v_i = \sum_{k=1}^{NV(i)} A_{ik}^v v_{r(k)} + b_i^v \quad (55)$$

onde :

$$b_i^v = \Omega_{ik}^v(i) + A p_i^0 v_i^0$$

$$\Omega_{ik}^v(i) = S_i^v \Delta \mathcal{G}_i + \sum_{k=1}^{NV(i)} \alpha_{ik}^v v_{ik}^*$$

$$A_{ik}^v = - (\alpha_{ik}^v + F_{ik}) \delta_{ik}^v + D_{ik}^v$$

$$Ap_i^v = Ap_i^0 + \sum_{k=1}^{NV(i)} A_{ik}^v - \alpha_{ik}^v$$

Lembrando que para $\varphi = v$, tem-se um termo de pressão embutido no termo fonte S_i^v , Cardoso (1997). Logo :

$$Ap_i^v v_i = \sum_{k=1}^{NV(i)} A_{ik}^v v_{r(k)} + b_i^v - \Delta \mathcal{G}_i (\nabla P)_i^x \quad (56)$$

e

$$Ap_j^v v_j = \sum_{k=1}^{NV(i)} A_{jk}^v v_{s(k)} + b_j^v - \Delta \mathcal{G}_j (\nabla P)_j^x \quad (57)$$

3.6 Equação para a Pressão

Até o presente momento tem-se apenas uma equação evolutiva para u e outra para v , regidas por uma equação do tipo advectiva-difusiva. A pressão aparece apenas como uma variável das equações gerais, não tendo uma equação evolutiva específica.

Para resolver este problema adota-se o acoplamento SIMPLE (Semi-Implicit Method for Pressure Linked Equations) entre a pressão e a velocidade, modificado para malhas co-localizadas, Peric *et al.*, (1988). Os passos gerais deste procedimento são descritos por Cardoso (1997).

Compondo as equações de u e v nos pontos nodais i e j , equações (45), (46), (56), (57) e, projetando-as na direção normal \hat{e}_{ij}

$$\hat{e}_{ij} = \frac{(x_j - x_i)\hat{i} + (y_j - y_i)\hat{j}}{\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}} \quad (58)$$

tem-se :

$$(w_i)_{ij} = u_i \mathbf{ex}_{ij} + v_i \mathbf{ey}_{ij} \quad (59)$$

$$(w_j)_{ij} = u_j \mathbf{ex}_{ij} + v_i \mathbf{ey}_{ij} \quad (60)$$

onde :

$$(w_i)_{ij} = \frac{1}{Ap_i^u} \left(\sum_{k=1}^{NV(i)} A_{ik}^u u_{r(k)} + b_i^u - \Delta \mathcal{G}_i(\nabla P)_i^x \right) \mathbf{ex}_{ij} + \frac{1}{Ap_i^v} \left(\sum_{k=1}^{NV(i)} A_{ik}^v v_{r(k)} + b_i^v - \Delta \mathcal{G}_i(\nabla P)_i^y \right) \mathbf{ey}_{ij} \quad (61)$$

$$(w_j)_{ij} = \frac{1}{Ap_j^u} \left(\sum_{k=1}^{NV(j)} A_{jk}^u u_{s(k)} + b_j^u - \Delta \mathcal{G}_j(\nabla P)_j^x \right) \mathbf{ex}_{ij} + \frac{1}{Ap_j^v} \left(\sum_{k=1}^{NV(j)} A_{jk}^v v_{s(k)} + b_j^v - \Delta \mathcal{G}_j(\nabla P)_j^y \right) \mathbf{ey}_{ij} \quad (62)$$

$(w_i)_{ij}$ e $(w_j)_{ij}$ são avaliados na direção normal ij , respectivamente, sobre os pontos nodais i e j . O valor da componente de velocidade sobre a interface ij é dado por W_{ij} e pode ser avaliada por uma média aritmética simples entre $(w_i)_{ij}$ e $(w_j)_{ij}$:

$$W_{ij} = \frac{(w_i)_{ij} + (w_j)_{ij}}{2} \quad (63)$$

Substituindo as equações (61) e (62) na equação (63) pode se compor uma expressão aproximada para W_{ij} , definida na equação (64), de modo que \hat{W}_{ij} considera uma média aritmética dos termos independentes da pressão e $d_{ij}^w(\nabla P)_{ij}^w$ aproxima os termos dependentes do gradiente de pressão :

$$W_{ij} = \hat{W}_{ij} - d_{ij}^w(\nabla P)_{ij}^w \quad (64)$$

onde :

$$\hat{W}_{ij} = 0.5 \left[\frac{1}{Ap_i^u} \left(\sum_{k=1}^{NV(i)} A_{ik}^u u_{r(k)} + b_i^u \right) \mathbf{ex}_{ij} + \frac{1}{Ap_i^v} \left(\sum_{k=1}^{NV(i)} A_{ik}^v v_{r(k)} + b_i^v \right) \mathbf{ey}_{ij} \right] + 0.5 \left[\frac{1}{Ap_j^u} \left(\sum_{k=1}^{NV(j)} A_{jk}^u u_{s(k)} + b_j^u \right) \mathbf{ex}_{ij} + \frac{1}{Ap_j^v} \left(\sum_{k=1}^{NV(j)} A_{jk}^v v_{s(k)} + b_j^v \right) \mathbf{ey}_{ij} \right] \quad (65)$$

$$\mathbf{d}_{ij}^w = 0.25 \left[\Delta \mathcal{G}_i \left(\frac{1}{Ap_i^u} + \frac{1}{Ap_i^v} \right) + \Delta \mathcal{G}_j \left(\frac{1}{Ap_j^u} + \frac{1}{Ap_j^v} \right) \right] \quad (66)$$

$$(\nabla P)_{ij}^w = \frac{(P_j - P_i)}{L_{ij}} \quad (67)$$

As equações (65) e (66) são geradas a partir de uma seqüência de aproximações definidas na equação (63) e descritas no Apêndice C. Note que desta forma o gradiente de pressão utilizado $(\nabla P)_{ij}^w$ na equação (67), considera as pressões vizinhas de W_{ij} , respectivamente os valores a frente e atrás de W_{ij} , o que é uma forma fisicamente consistente, conforme Peric (1988).

W_{ij} também pode ser avaliado de forma análoga a interpolação UDS proposta para u_{ij} e v_{ij} :

$$W_{ij} = \delta_{ij}^+ w_i + \delta_{ij}^- w_j \quad (68)$$

porém, a avaliação feita através da equação (63) mostrou-se numericamente mais estável (vide capítulo 4).

Capítulo 4 – Resultados e Discussões

4.1 Introdução

O presente capítulo tem por finalidade discutir a performance das diferentes linearizações na discretização das equações de Navier-Stokes, comparando a linearização por separação, Pantankar (1980), com a expansão em Série de Taylor, conforme o método de Newton, com interpolação UpWind de 1ª ordem, descrita no capítulo 3.

Além disso, este capítulo faz uma descrição da metodologia empregada e da forma como foi feita a análise e comparação para validação de resultados obtidos.

4.2 Formulação do Problema

O teste utilizado para a validação da discretização das equações de Navier-Stokes usando a expansão via Série de Taylor, conforme o método de Newton, foi o escoamento laminar incompressível em uma cavidade quadrada de profundidade infinita sujeita a uma parede superior deslizante com velocidade constante, semelhante a uma esteira. Um esboço dessa cavidade é apresentado na figura a seguir :

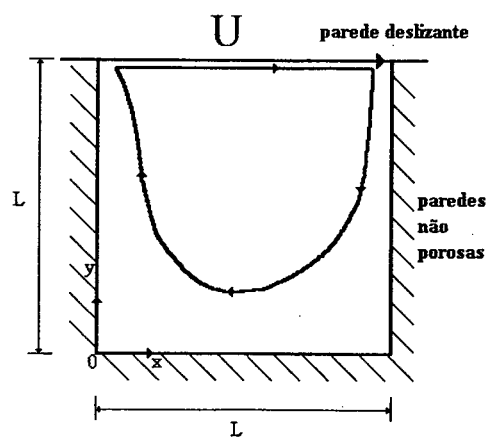


Figura 4 : Parede superior deslizante sobre uma cavidade quadrada.

Este problema da cavidade quadrada, sujeita a uma parede superior deslizando modela matematicamente, por exemplo um mancal deslizando onde uma esteira desliza constantemente sobre uma cavidade com um fluido dentro, o qual pode ser óleo, água, ar, etc. Sendo que a parede inferior e as paredes laterais não permitem entrada ou saída de massa do interior da cavidade, ou seja, são impermeáveis.

As condições de contorno utilizadas foram :

- parede lateral direita : $x = L, y = 0, u = v = 0$;
- parede lateral esquerda : $x = 0, u = v = 0$;
- parede inferior : $y = 0, u = v = 0$;
- parede superior : $y = L, u = 1, v = 0$;
- largura da malha : $L = 32.10$ (valor escolhido em função da malha);

Para a solução do problema proposto foi utilizado um programa computacional escrito em linguagem C, no qual foi implementada a linearização na discretização das equações de Navier-Stokes usando a expansão via Série de Taylor, conforme o método de Newton, com interpolação UpWind de 1ª ordem, simulando regime estacionário de escoamento, embora o programa implementado possibilite a discretização temporal. Maiores detalhes podem ser obtidos em Cardoso (1997). A solução do sistema de equações lineares gerados da discretização foi obtida através do método do Gradiente Conjugado, Mariani (1997) e Golub (1989).

A velocidade de convergência do método utilizado está relacionada ao fator de relaxação empregado e ao número de vezes em que cada uma das variáveis (velocidades e pressão) são resolvidas durante cada iteração. No presente trabalho, utilizou-se relaxação 0.7 para as velocidades u, v e 0.5 para a correção da pressão. O número de repetições internas para as velocidades u e v foi de 2 (dois), enquanto que para a correção da pressão foi de 3 (três). Durante os testes realizados observou-se que a velocidade de convergência do sistema de equações, via linearização de Newton, dependia dos fatores de relaxação empregados, bem como do número de repetições internas. Há de se considerar que o presente trabalho não tem como propósito indicar os melhores valores para esses coeficientes, uma vez que demanda um número grande de testes e de tempo, dependendo do número de Reynolds a ser utilizado. Maiores detalhes sobre sub-relaxação podem ser encontrados em Tanyi e Tatcher (1996).

Os resultados foram coletados com números de Reynolds 100, 1000 e 10000, calculados como segue :

$$Re = \rho \frac{U.L}{\mu} \quad (69)$$

onde :

U : velocidade de movimento da parede superior, $U = 1$;

L : largura da cavidade, $L = 32.10$;

ρ : a massa específica do fluido, $\rho = 1$;

μ : viscosidade dinâmica do fluido varia conforme o número de Reynolds desejado.

Além disso, utilizou-se como condições iniciais velocidades $u = v = 0.5$ e pressão $p = 0$. E, também, a regra de Armijo (Apêndice B) foi introduzida no método de Newton com a finalidade de estabilizar a convergência dos sistema de equações.

Para avaliar os resultados numéricos, realizados com o programa implementado, adotou-se o seguinte critério de convergência :

$$\text{erro}_i^k = \sum_{j=1}^{NV(i)} |F_{ij}^k| < \xi \quad (70)$$

onde :

$\sum_{j=1}^{NV(i)} F_{ij}$: resíduo da conservação da massa do VC i ;

ξ : exatidão deseja, $\xi = 1.10^{-5}$.

O gradiente de pressão utilizado foi o proposto por Taniguchi (Taniguchi *et al.*, 1991; Taniguchi e Kobayashi, 1991).

Nos testes realizados, utilizou-se uma malha de volumes hexagonais com 6400 pontos nodais (figura 5) gerada por diagrama de voronoi, sendo que na geração da malha foi utilizado o programa gerador de Diagrama de Voronoi de Maliska Jr. (1994). O ponto escolhido para a impressão do campo de velocidade (ponto 35, vide item 4.3) encontra-se destacado dos demais porque sua região esta pintada de preto.

Para o contorno, porém, o programa permite apenas a utilização de um único vizinho para todos os pontos nodais ligados a uma parede, o qual foi bastante útil para a resolução do sistema proposto. Convém salientar, no entanto, que as condições de contorno

poderiam ser inseridas nas próprias equações discretizadas, apenas isto não foi feito pois não se necessitava de mais de uma condição de contorno em cada lado da malha.

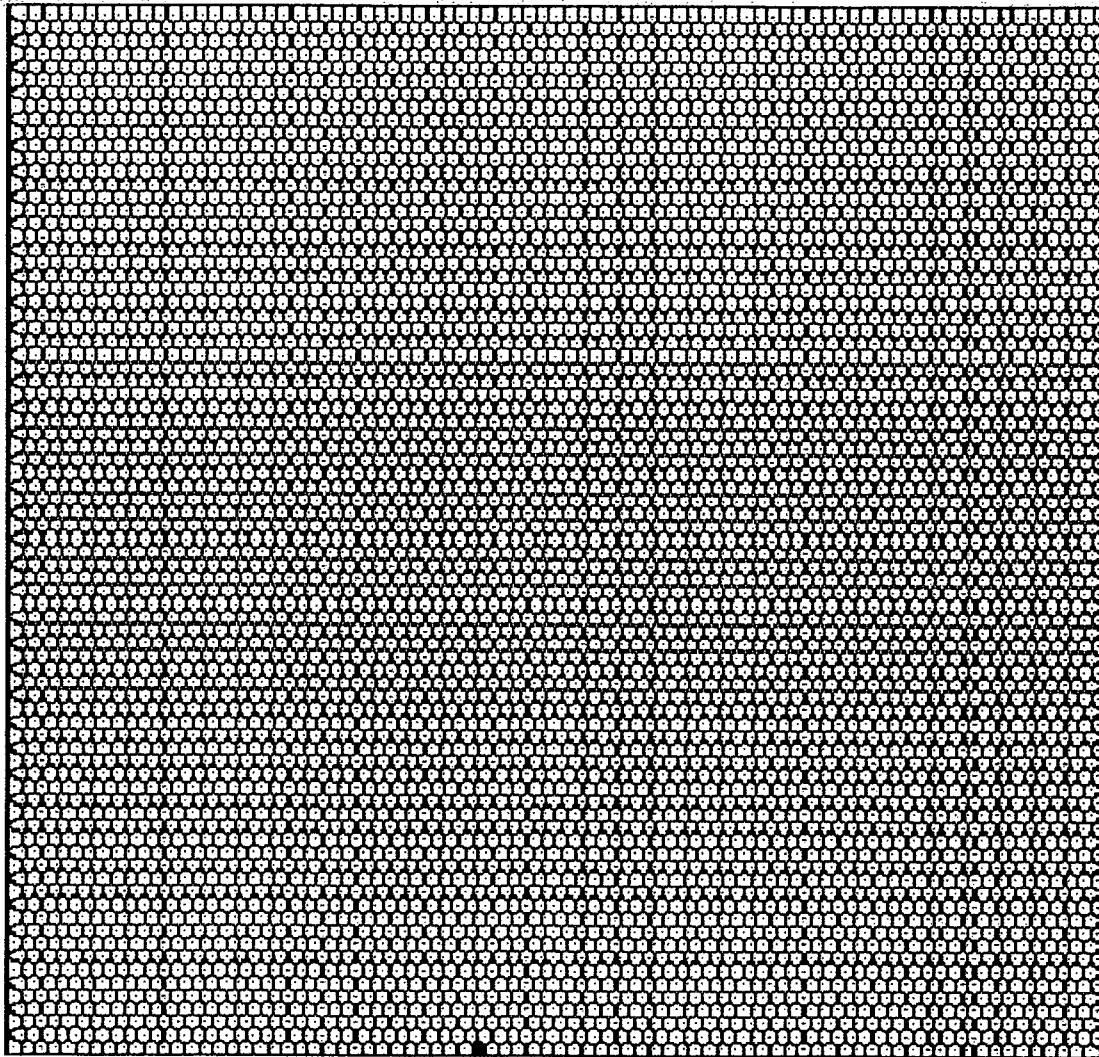


Figura 5 : Malha hexagonal com 6400 pontos nodais

Finalmente, os resultados apresentados neste capítulo foram obtidos utilizando-se uma máquina IBM 9076 SP/2 , o qual é um supercomputador com alta capacidade de processamento científico e que possibilita o desenvolvimento de programas seriais e paralelos. O SP/2 foi disponibilizado, para a realização deste trabalho, pelo Núcleo de Processamento de Dados – NPD da Universidade Federal de Santa Catarina – UFSC. Ressalta-se, contudo, que os programas implementados neste trabalho utilizam processamento do tipo serial.

4.3 Análise de Resultados Obtidos

Os resultados obtidos com a linearização via Método de Newton foram comparados com os obtidos com a linearização por separação (com UDS), conforme Mariani (1997) e Cardoso (1997), sob as mesmas condições descritas na seção anterior, e com os resultados obtidos por Ghia, Ghia et alli (1982)¹.

Conforme descrito anteriormente, durante a realização dos testes a linearização via método de Newton mostrou que sua convergência dependia dos corretos fatores de relaxação utilizados, assim como a estabilidade e a velocidade da convergência dependiam da forma como eram avaliadas segundo as equações de W_{ij} , equações (63) e (68).

A avaliação feita através da equação (68) não convergiu para nenhum dos números de Reynolds testados, conforme as diferentes avaliações das equações de u e v , equações (28) e (34), respectivamente, ou alternativamente através das equações (33) e (35) para u e v , respectivamente. Assim, em todos os testes utilizou-se a equação (63) que mostrou se mais estável e, conseqüentemente, produziu bons resultados. Também existiram diferenças com relação à avaliação das velocidades u e v , feita através das equações (28) e (34) de um lado e de outro as equações (33) e (35), respectivamente, embora estas diferenças tenham afetado somente a velocidade de convergência (em número de iterações).

Em todos os casos foi testada a regra de Armijo (vide apêndice B) com o objetivo de estabilizar e aumentar o campo de convergência do sistema de equações.

As figuras, a seguir, apresentam comparações dos resultados obtidos através da linearização de Newton com a linearização por separação (Cardoso, 1997; Mariani, 1997; Patankar, 1980) e com os resultados obtidos por Ghia, Ghia et alli (1982).

Tais figuras apresentam os perfis das velocidades u e v para Reynolds 100 e 1000 ao longo de uma linha média vertical ($x = L / 2$) e horizontal ($y = L / 2$), respectivamente, para as velocidades nas direções x e y na malha modelada por volumes hexagonais com 6400 pontos. As linhas vertical e horizontal estão localizadas no centro geométrico da cavidade quadrada (figura 4).

Nas figuras, as duas avaliações das velocidades u e v (interpolação UDS Diferença Central) testadas na linearização de Newton podem ser identificadas a partir da legenda apresentada na tabela 2, a seguir :

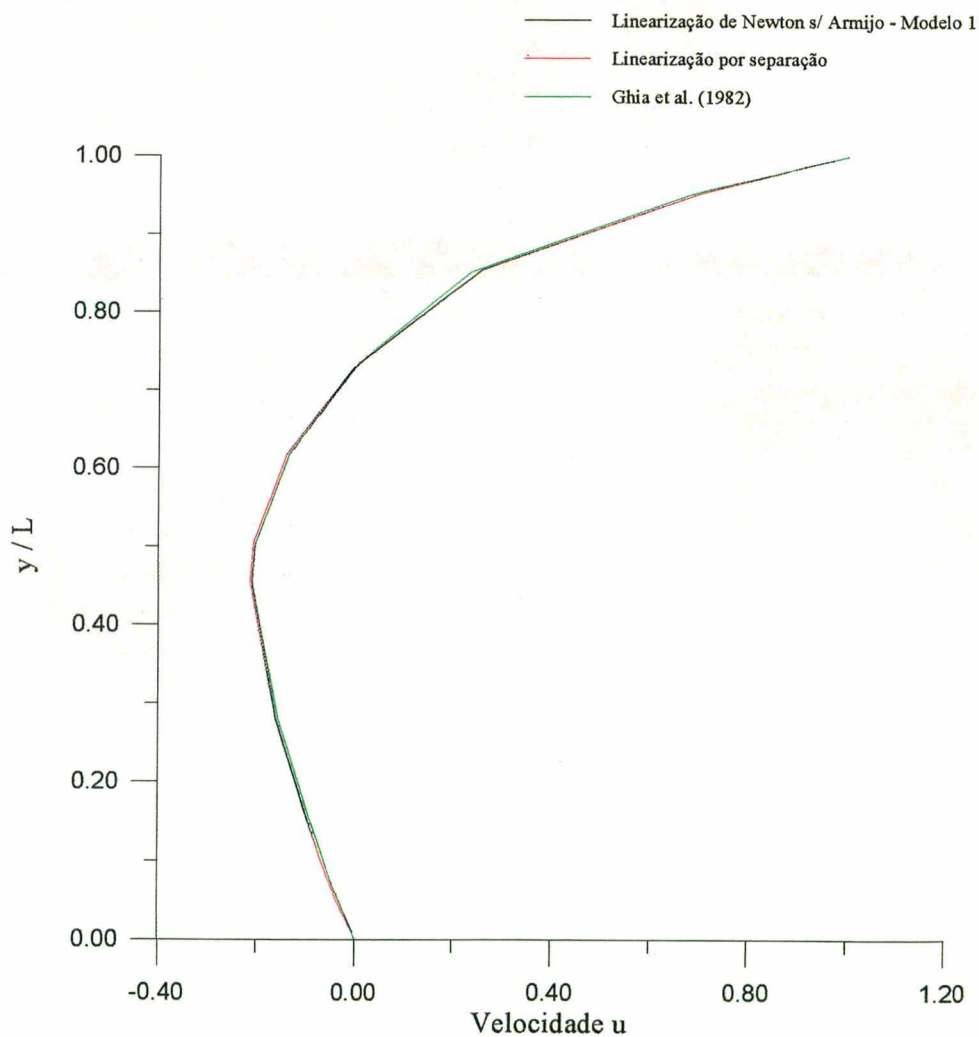
¹ Solução padrão normalmente utilizada em validações e comparações.

Tipo de Avaliação	Velocidades	
	U	v
Modelo 1 (interpolação UDS)	equação (28)	equação (34)
Modelo 2 (média aritmética simples)	equação (33)	equação (35)

Tabela 2 : Legenda das avaliações das velocidades u e v na linearização de Newton

4.3.1 Resultados Obtidos com Reynolds 100

A seguir, analisa-se o perfil da velocidade u ao longo de uma linha vertical central ($x = L / 2$) e da velocidade v ao longo de uma linha horizontal central ($y = L / 2$), comparando as linearizações de Newton e por separação e com os resultados obtidos por Ghia.

Figura 6 : Velocidade u ao longo de uma linha vertical central, $Re = 100$ – Modelo 1

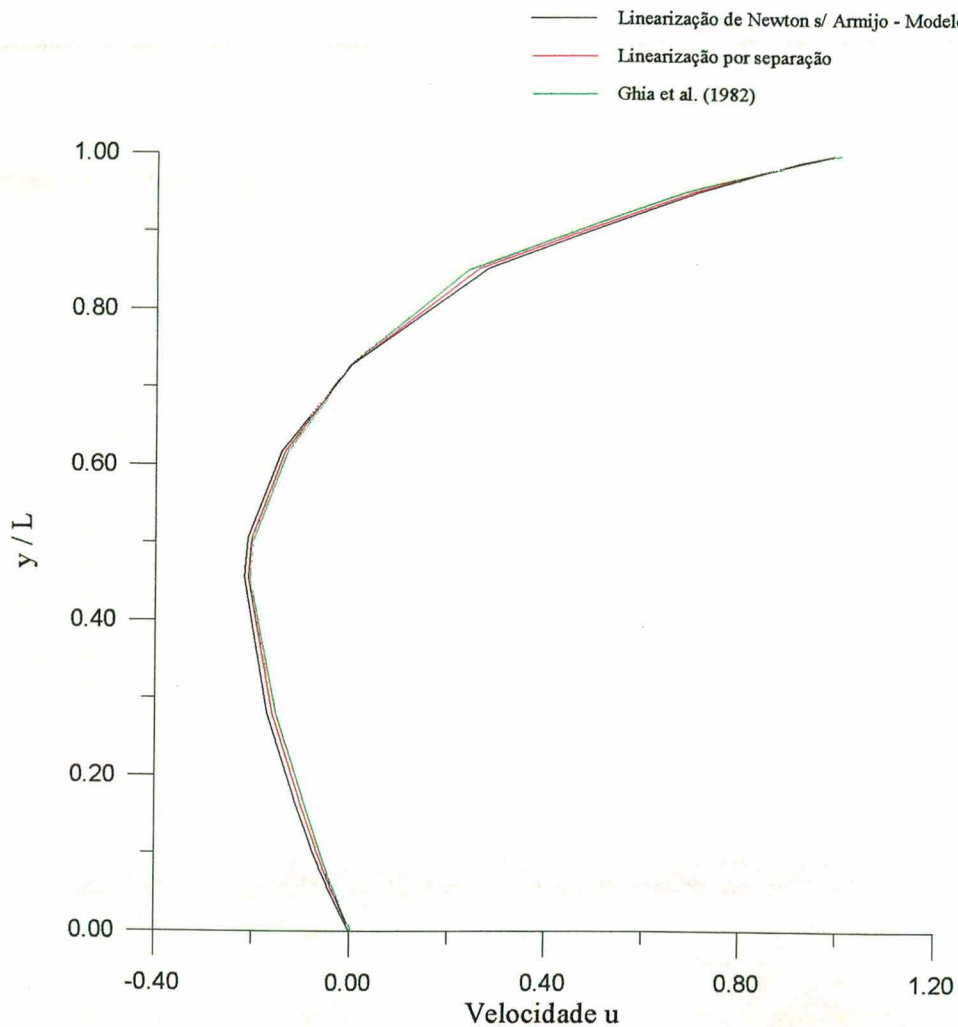


Figura 7 : Velocidade u ao longo de uma linha vertical central, $Re = 100$ – Modelo 2

As figuras 6 e 7, acima, apresentam o perfil da velocidade u ao longo de uma linha vertical central.

Na figura 6 a velocidade u foi avaliada conforme o modelo 1 (vide tabela 2), enquanto que na figura 7 foi utilizado o modelo 2 (vide tabela 2), ambos sem aplicar a regra de Armijo.

Observando as figuras 6 e 7, constata-se que os resultados obtidos com ambas as linearizações, por separação e por Newton, ficaram muito próximas e algumas vezes se sobrepõem aos resultados obtidos por Ghia.

Além disso, na figura 6 a linearização de Newton se sobrepõe totalmente à linearização por separação, o que não acontece na figura 7 onde a linearização de Newton ficou um pouco afastada em alguns pontos.

As figuras 8 e 9, a seguir, apresentam o perfil da velocidade v ao longo de uma linha horizontal central.

Na figura 8 a velocidade v foi avaliada conforme o modelo 1 (vide tabela 2), enquanto que na figura 9 foi utilizado o modelo 2 (vide tabela 2).

Observando as figuras 8 e 9 contata-se que os resultados obtidos com ambas as linearizações, por separação e por Newton, ficaram muito próximas aos resultados obtidos por Ghia.

Além disso, na figura 8 a linearização de Newton se sobrepõe totalmente a linearização por separação, o que não acontece na figura 9 onde a linearização de Newton ficou um pouco afastada em alguns pontos.

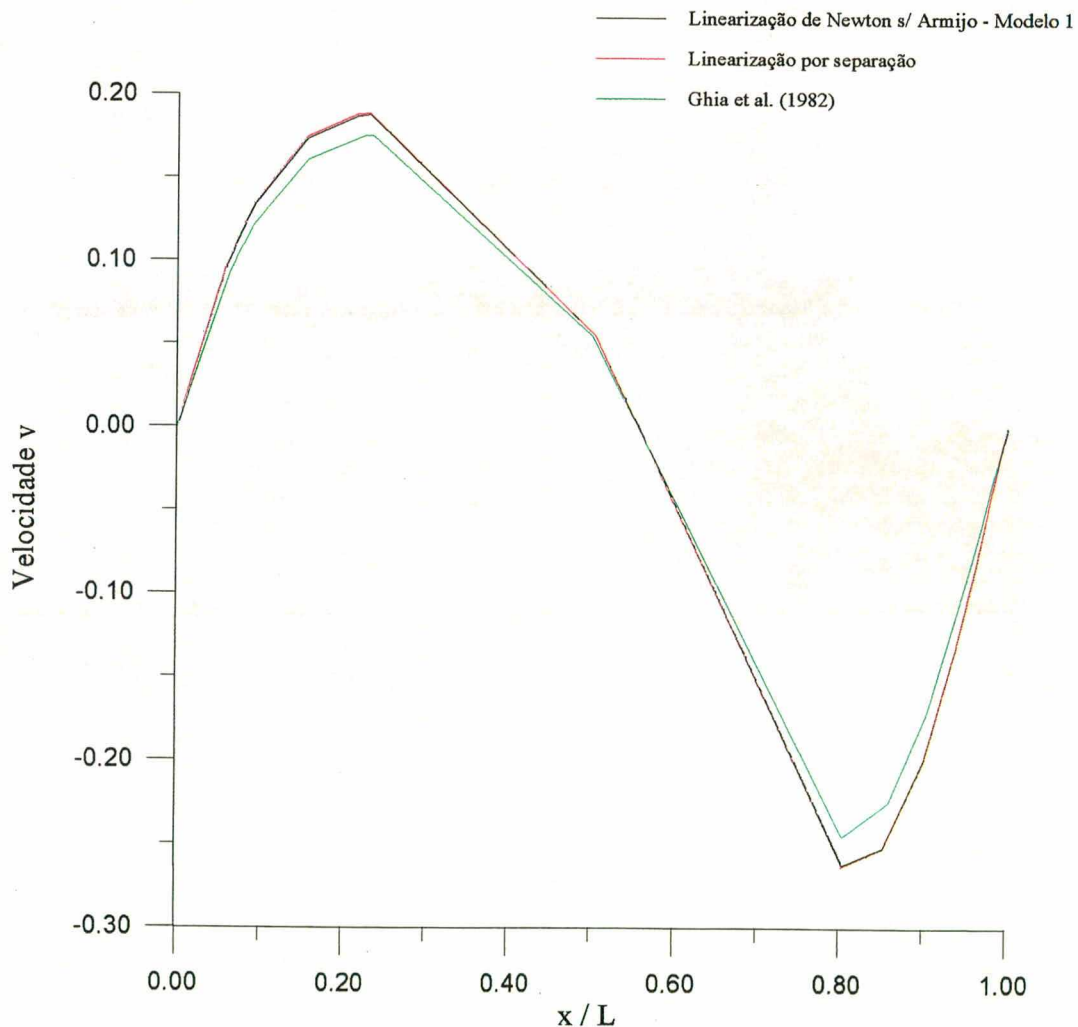


Figura 8 : Velocidade v ao longo de uma linha horizontal central, $Re = 100$ – Modelo 1

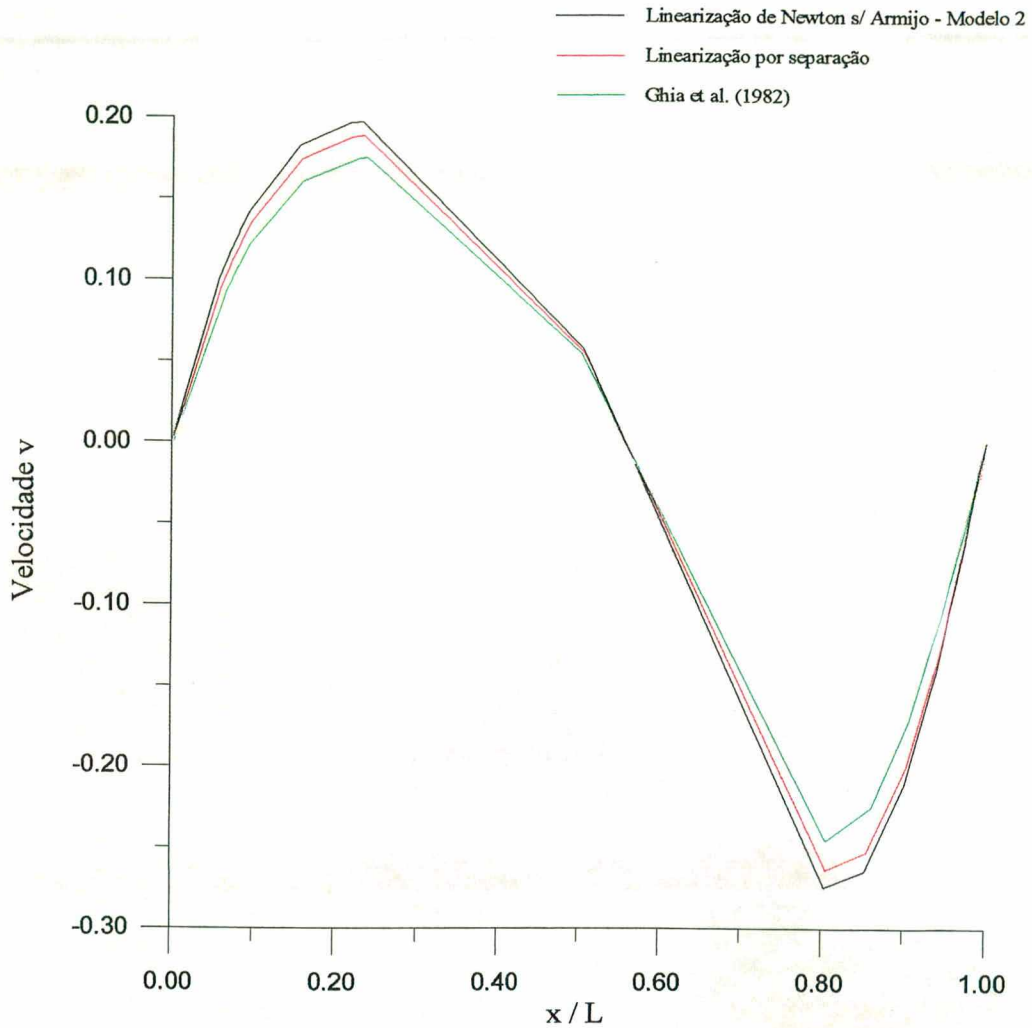


Figura 9 : Velocidade v ao longo de uma linha horizontal central, $Re = 100$ – Modelo 2

Tipo de Linearização	Nº de Iterações	Tempo de CPU (minutos)
Separação	1654	24,05
Newton (modelo 1)	1309	18,28
Newton (modelo 2)	1235	16,18

Tabela 3 : Linearização de Newton x linearização por separação, $Re = 100$

Analisando os resultados numéricos obtidos, constata-se que ambas as avaliações realizadas com a linearização de Newton (tabela 2) obtiveram desempenho melhor do que a linearização por separação, sendo que o resultado obtido com Diferença Central apresentou um perfil para as velocidades u e v ligeiramente mais distantes aos de Ghia, porém, obteve

melhor desempenho em número de iterações e tempo de CPU necessários à convergência do sistema de equações (tabela 3).

A seguir, tem-se os resultados obtidos com a linearização de Newton aplicando a regra de Armijo.

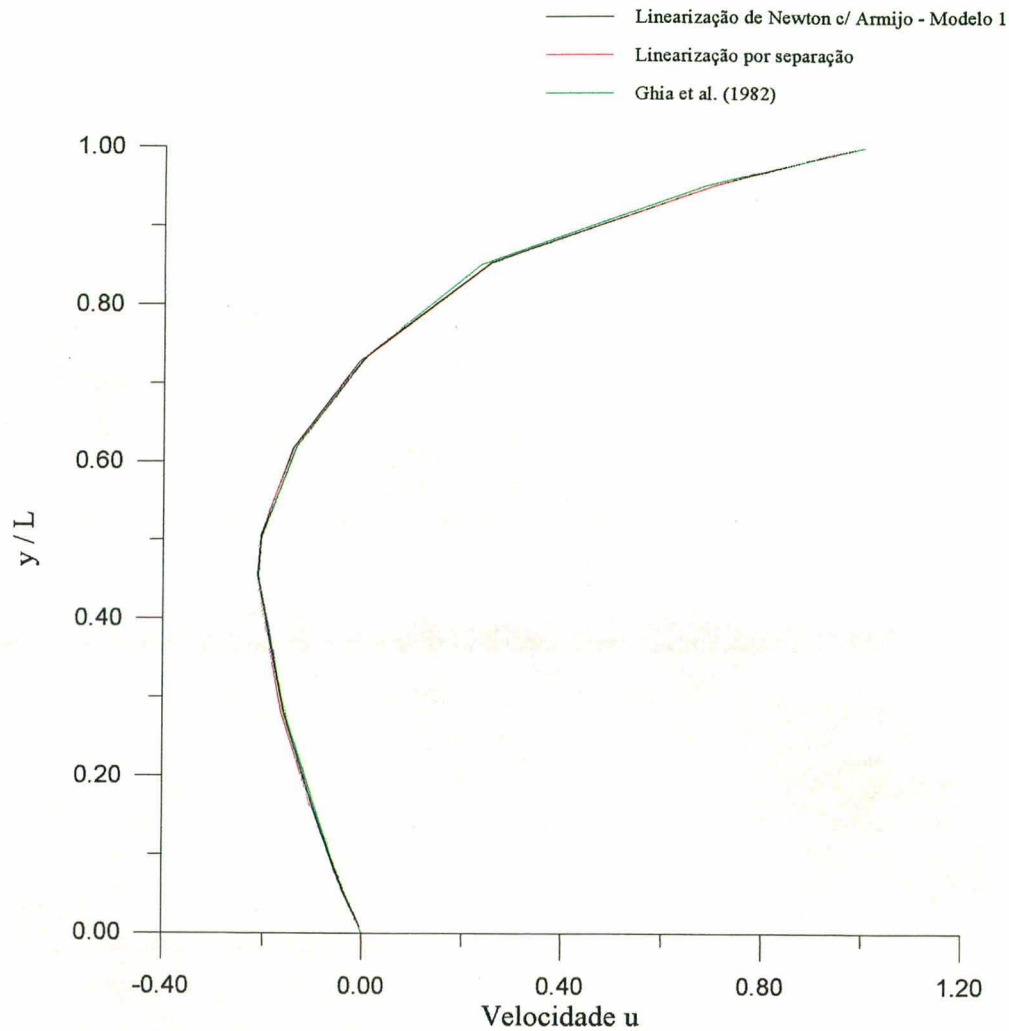


Figura 10 : Velocidade u ao longo de uma linha vertical central, $Re = 100$ – Modelo 1

As figuras 10 e 11 apresentam o perfil da velocidade u ao longo de uma linha vertical central. Na figura 10 a velocidade u foi avaliada conforme o modelo 1 (vide tabela 2), enquanto que na figura 11 foi utilizado o modelo 2 (vide tabela 2), ambas aplicando a regra de Armijo. Observando as figuras 10 e 11, constata-se que os resultados obtidos com ambas as linearizações, por separação e por Newton, ficaram muito próximas e algumas vezes se sobrepõem aos resultados obtidos por Ghia. Além disso, na figura 10 a linearização de Newton

se sobrepõe totalmente à linearização por separação, o que não acontece na figura 11 onde a linearização de Newton ficou um pouco afastada em alguns pontos.

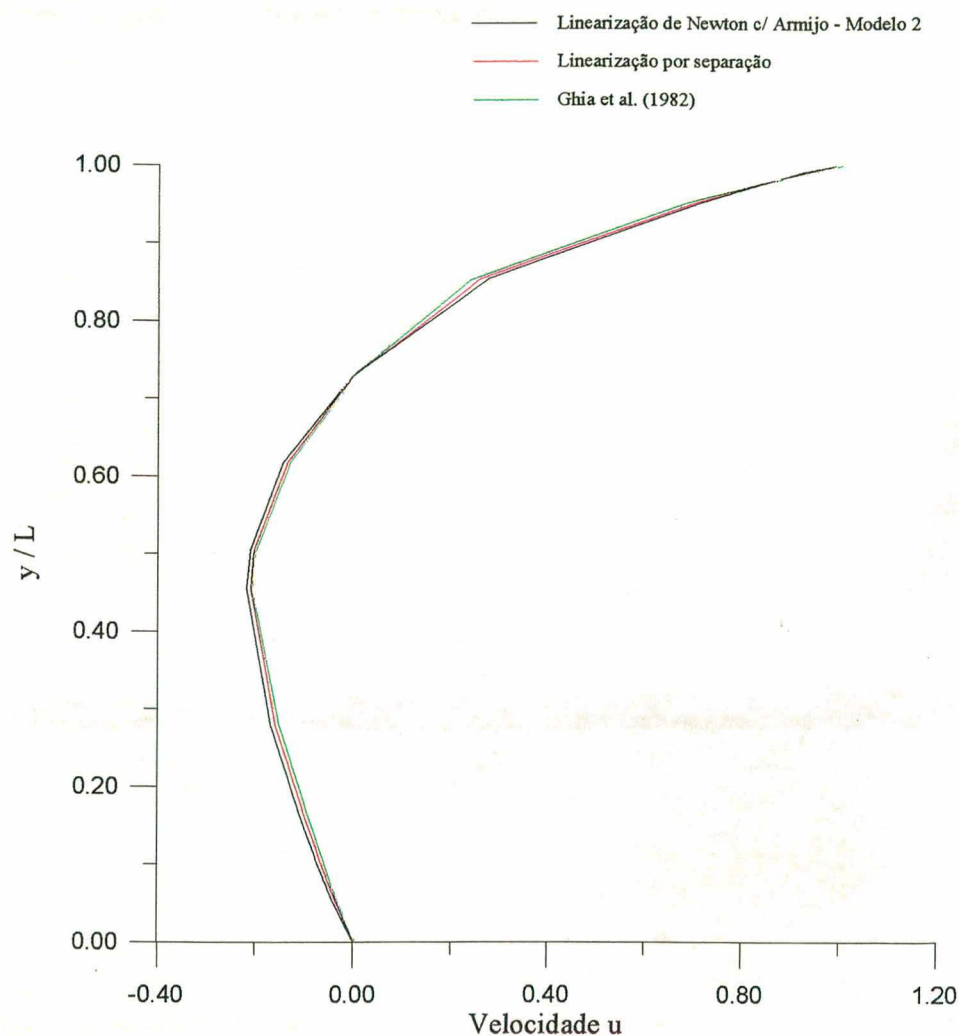


Figura 11 : Velocidade u ao longo de uma linha vertical central, $Re = 100$ – Modelo 2

As figuras 12 e 13 apresentam o perfil da velocidade v ao longo de uma linha horizontal central. Na figura 12 a velocidade v foi avaliada conforme o modelo 1 (vide tabela 2), enquanto que na figura 13 foi utilizado o modelo 2 (vide tabela 2), ambas aplicando a regra de Armijo. Observando as figuras 12 e 13 contata-se que os resultados obtidos com ambas as linearizações, por separação e por Newton, ficaram muito próximas aos resultados obtidos por Ghia. Além disso, na figura 12 a linearização de Newton se sobrepõe em alguns pontos a linearização por separação, o que não acontece na figura 13 onde a linearização de Newton ficou mais afastada em alguns pontos.

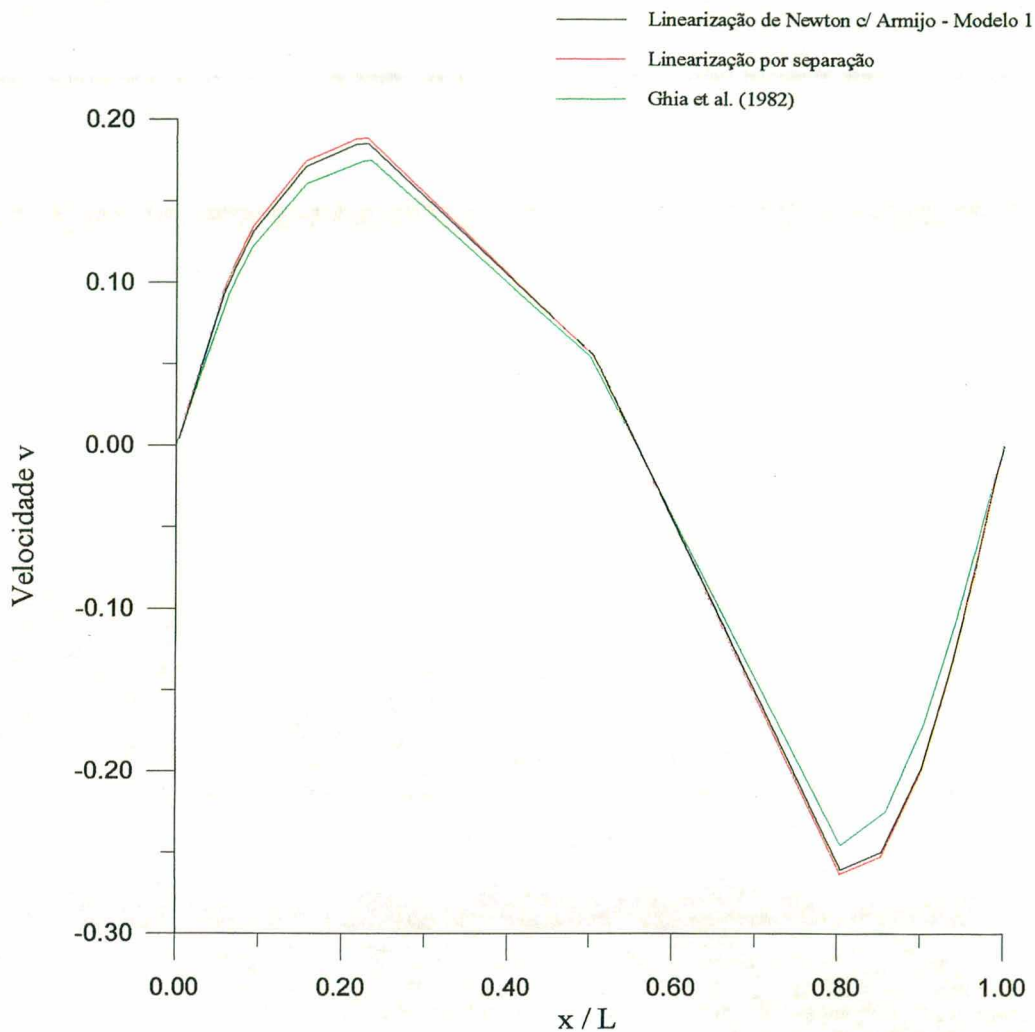


Figura 12 : Velocidade v ao longo de uma linha horizontal central, $Re = 100$ – Modelo 1

Analisando os resultados numéricos obtidos, constata-se que ambas as avaliações realizadas com a linearização de Newton (tabela 2) obtiveram desempenho pior do que a linearização por separação considerando número de iterações e tempo de CPU necessários à convergência dos sistema de equações.

O resultado obtido com Diferença Central, modelo 2, apresentou um perfil para as velocidades u e v mais distantes ao de Ghia, porém, obteve melhor desempenho em números de iterações e tempo de CPU necessários à convergência do sistema de equações (tabela 4) se comparado com a linearização de Newton com a regra de Armijo e velocidades u e v avaliadas através do modelo 1.

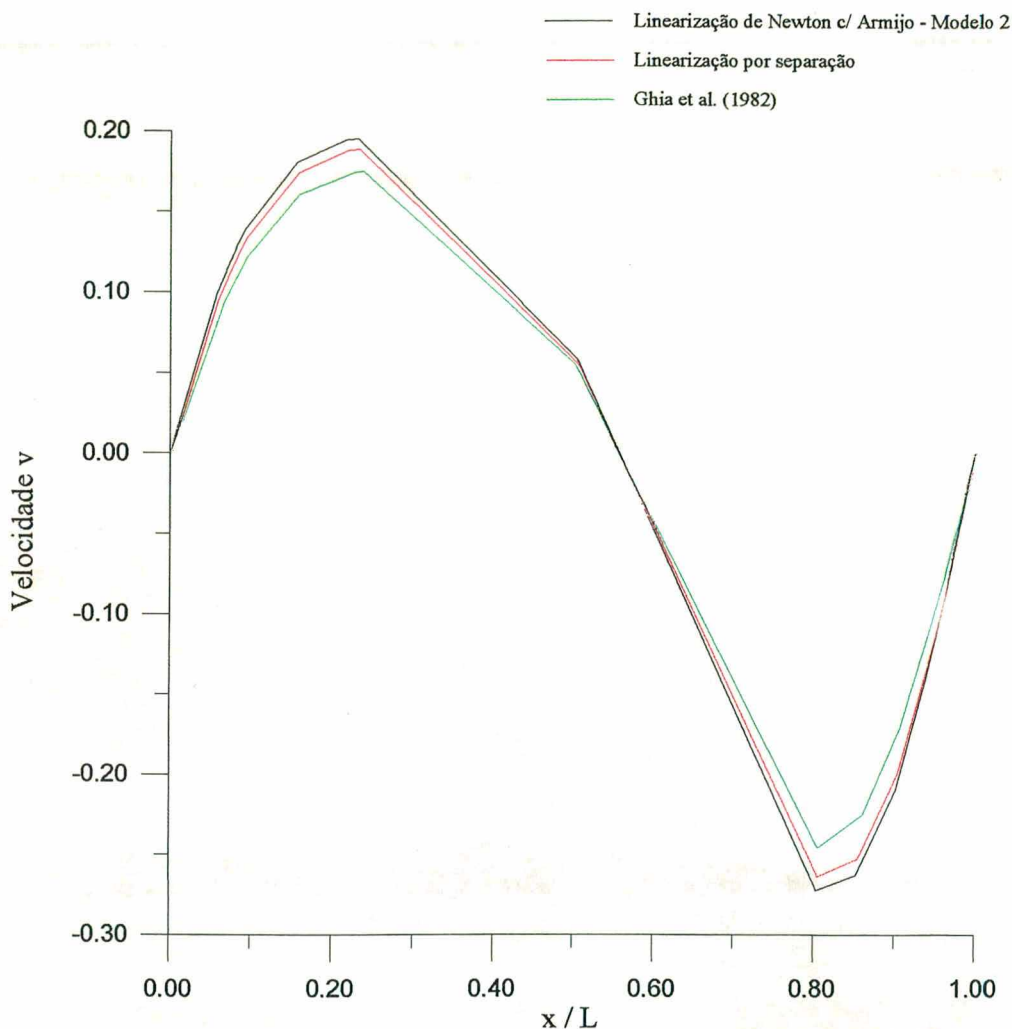


Figura 13 : Velocidade v ao longo de uma linha horizontal central, $Re = 100$ – Modelo 2

Tipo de Linearização	Nº de Iterações	Tempo de CPU (minutos)
Separação	1654	24,05
Newton (modelo 1)	1819	30,47
Newton (modelo 2)	1768	28,17

Tabela 4 : Linearização de Newton com Armijo x linearização por separação, $Re = 100$

Em uma análise final sobre os dados expostos nesta seção, pode-se concluir que o perfil das velocidades u e v avaliadas através da linearização de Newton permaneceu próximo ou, conforme alguns casos, se sobrepondo aos resultados obtidos por Ghia e, também, aos resultados obtidos através da linearização por separação.

Esperava-se que a regra de Armijo aplicada à linearização de Newton aumentasse o campo de convergência ou, pelo menos, estabilizasse o sistema de equações, o que não ocorreu. Além disso, foi mais lento em número de iterações e tempo de CPU se comparado com a linearização de Newton sem a regra de Armijo.

Uma análise posterior faz-se necessário para tentar verificar o motivo do pior comportamento da regra de Armijo aplicada a linearização de Newton, com Reynolds 100.

A avaliação das velocidades u e v , na linearização de Newton, por Diferença Central obteve melhores resultados do que a avaliação por UDS.

A tabela 5 apresenta um comparativo dos testes realizados nesta seção, trazendo os tipos de linearizações (com as avaliações de u e v empregadas), nº de iterações e tempo de CPU necessários para a convergência do sistema de equações, um campo de velocidade num determinado ponto aleatório da malha (neste caso somente da velocidade u no ponto 35).

Tipo de Linearização		Nº de Iterações	Tempo de CPU (minutos)	Velocidade[35]. u
Separação (com UDS)		1654	24,05	-0.004896
Newton sem Armijo	Modelo 1	1309	18,28	-0.004895
	Modelo 2	1235	16,18	-0.005439
Newton com Armijo	Modelo 1	1819	30,47	-0.004789
	Modelo 2	1768	28,17	-0.005365

Tabela 5 : Tipo de Linerização versus nº de iterações – $Re = 100$

4.3.2 Resultados Obtidos com Reynolds 1000

A seguir, analisa-se o perfil da velocidade u ao longo de uma linha vertical central ($x = L / 2$) e da velocidade v ao longo de uma linha horizontal central ($y = L / 2$), comparando a linearização de Newton com a linearização por separação e com os resultados obtidos por Ghia.

As figuras 14 e 15 apresentam o perfil da velocidade u ao longo de uma linha vertical central obtido através das linearizações apresentadas. Na figura 14 a velocidade u foi avaliada através do modelo 1 (tabela 2), enquanto que na figura 15 a velocidade u foi avaliada através do modelo 2 (tabela 2), ambas sem aplicar a regra de Armijo. Ambos os gráficos comparam a linearização de Newton com a linearização por separação e com os resultados obtidos por Ghia.

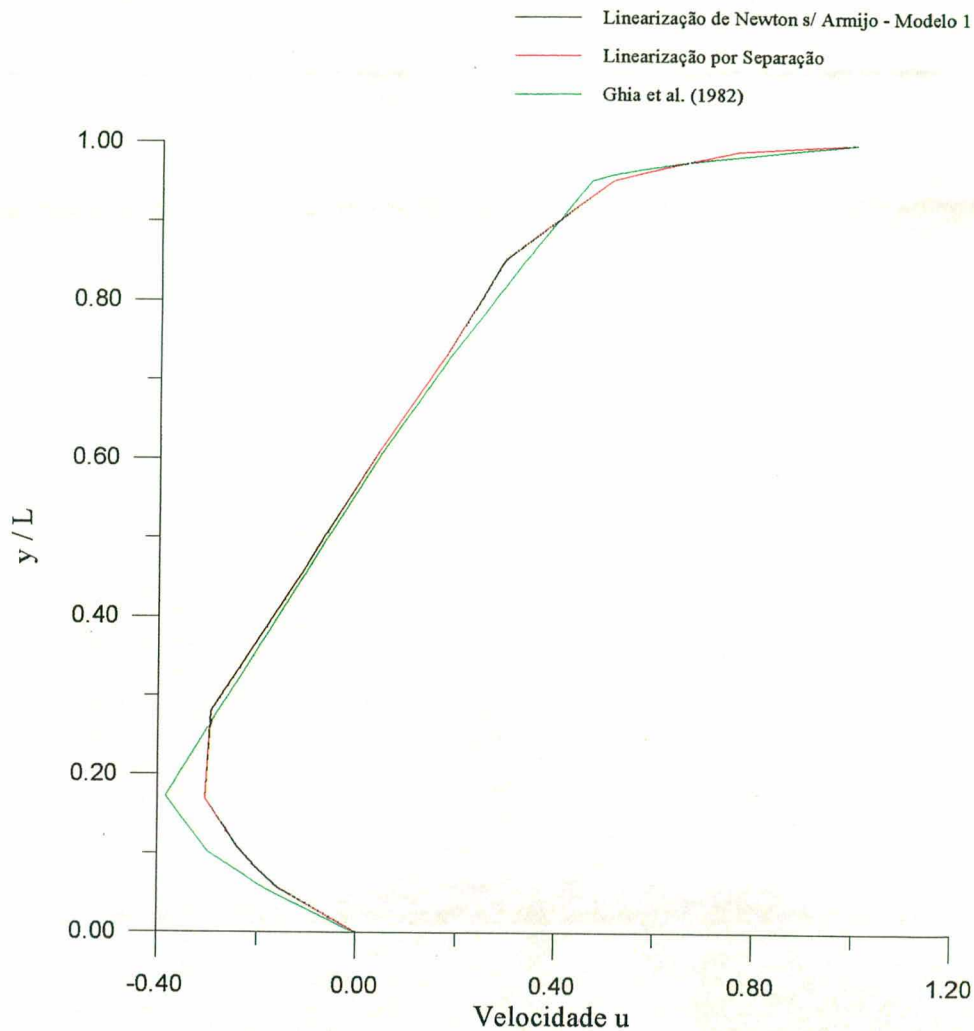


Figura 14 : Velocidade u ao longo de uma linha vertical central, $Re = 1000$ – Modelo 1

Nota-se que o perfil da velocidade u para ambas as linearizações, de Newton e por separação, ficaram próximas dos resultados obtidos por Ghia. No entanto, os resultados obtidos com Reynolds 1000 demonstraram que os resultados obtidos com Reynolds 100 aproximaram-se mais dos resultados obtidos por Ghia. Na figura 14, o perfil da velocidade u sobrepôs os resultados obtidos com a linearização por separação, o que não se repete no desempenho em número de iterações necessária à convergência, onde a linearização de Newton com a avaliação através do modelo 1 (tabela 2) obteve desempenho inferior. Já a avaliação através do modelo 2 (tabela 2) foi a que obteve resultados melhores dentre as três em número de iterações necessárias à convergência, embora o perfil da velocidade u tenha ficado próximo ao da linearização por separação, figura 15.

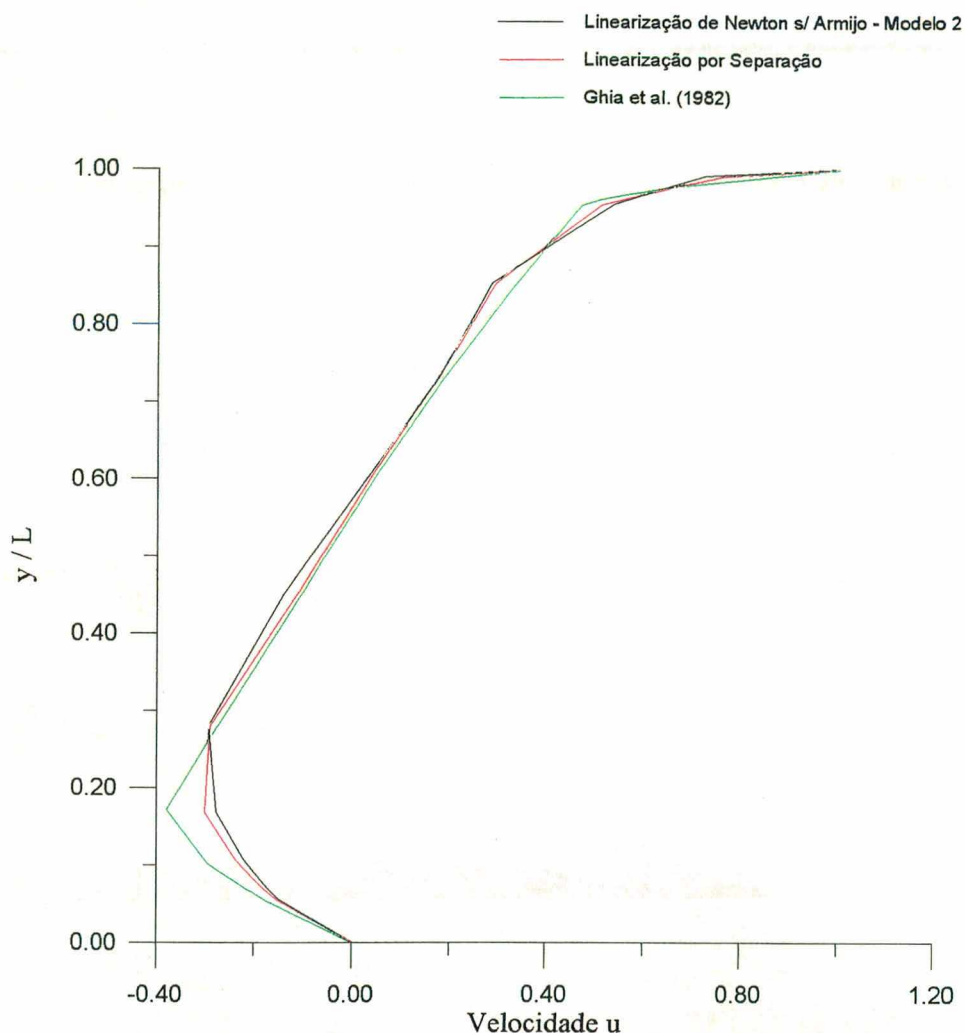


Figura 15 : Velocidade u ao longo de uma linha vertical central, Re = 1000 – Modelo 2

Observando a tabela 6 pode-se constatar que a linearização de Newton avaliada através do modelo 1 obteve o pior desempenho dentre os três testes realizados. Consta-se, também, que apesar da linearização de Newton avaliada através do modelo 2 ter obtido o melhor resultado no critério número de iterações necessárias à convergência do sistema de equações, a mesma obteve um tempo de CPU ligeiramente superior a linearização por Separação.

Tipo de Linearização	Nº de Iterações	Tempo de CPU (minutos)
Separação	2325	32,17
Newton (modelo 1)	2516	37,27
Newton (modelo 2)	2247	32,62

Tabela 6 : Linearização de Newton x linearização por separação, Re = 1000

As figuras 16 e 17 apresentam o perfil da velocidade v ao longo de uma linha horizontal central obtido através da linearização de Newton sem a regra de Armijo. Na figura 16 a velocidade v foi avaliada através do modelo 1 (tabela 2), enquanto que na figura 17 a velocidade u foi avaliada através do modelo 2 (tabela 2). Ambos os gráficos comparam a linearização de Newton, a linearização por separação e os resultados obtidos por Ghia.

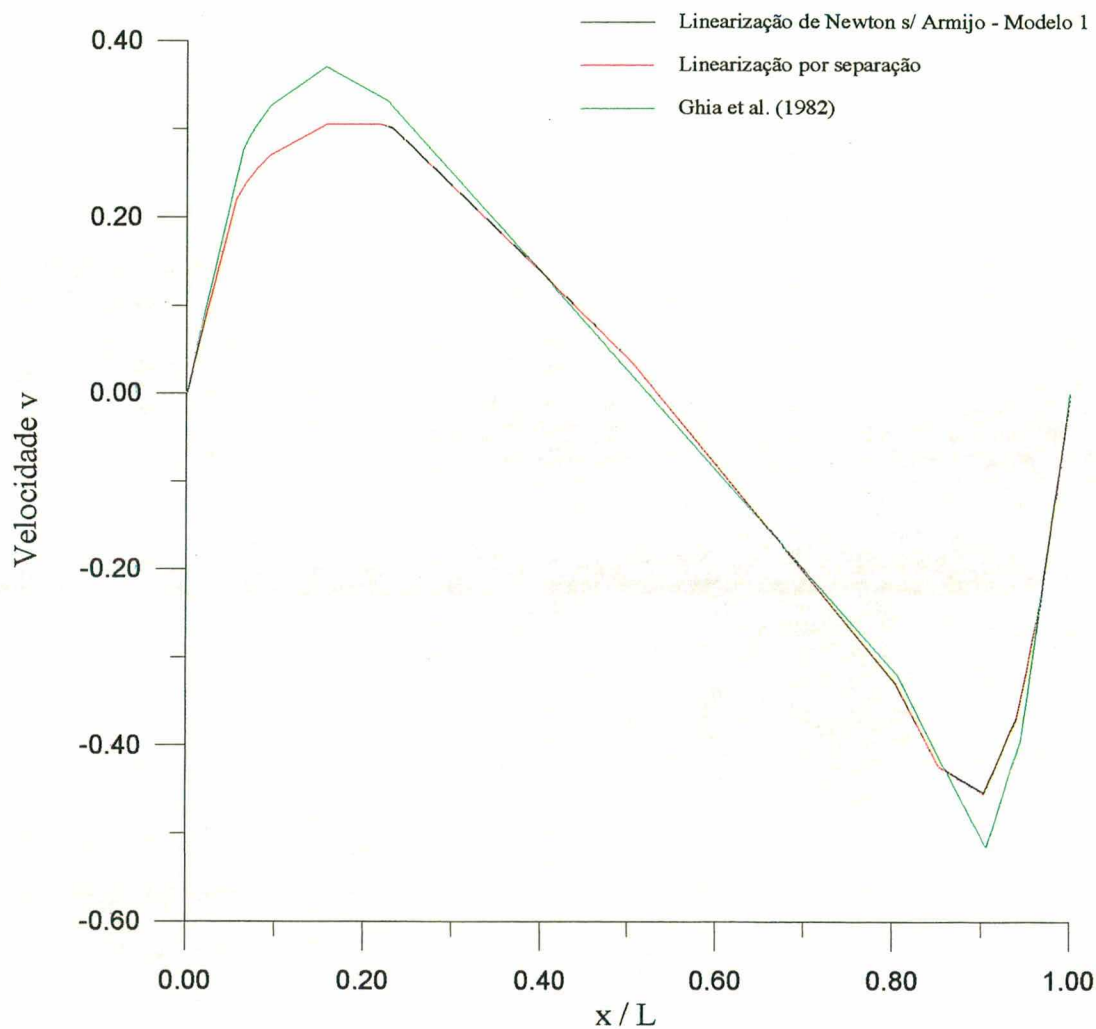


Figura 16 : Velocidade v ao longo de uma linha horizontal central, $Re=1000$ – Modelo 1

Nota-se que o perfil da velocidade v nas figuras 16 e 17 mostra que a linearização de Newton ficou próxima dos resultados obtidos por Ghia, assim como a linearização por separação.

Na figura 16, a linearização de Newton sobrepõe-se à linearização por separação em sua totalidade, o que não acontece na figura 17 onde a linearização de Newton mantém-se próxima. A tabela 6 fornece informações adicionais sobre esse caso apresentado.

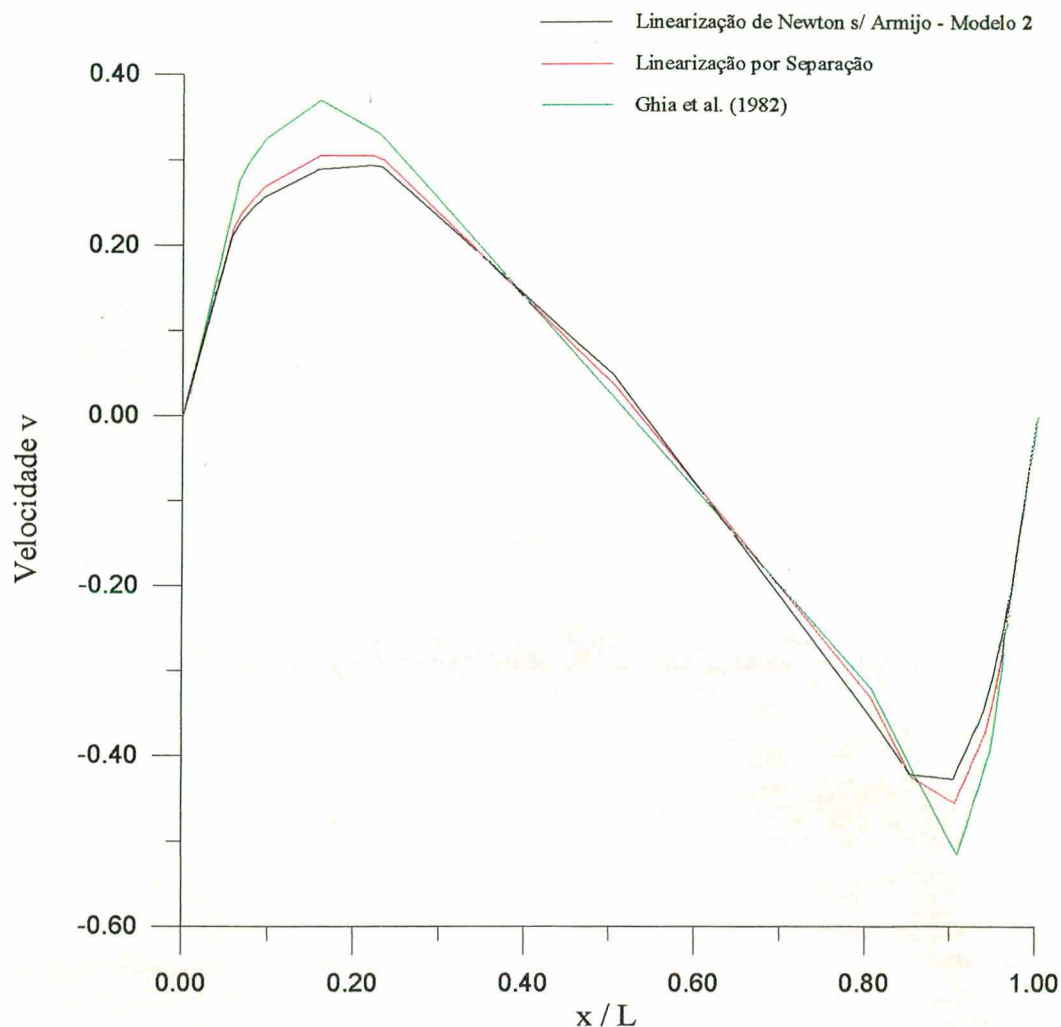


Figura 17 : Velocidade v ao longo de uma linha horizontal central, $Re=1000$ – Modelo 2

Os resultados obtidos com o perfil da velocidade u para a linearização de Newton com regra de Armijo, figuras 18 e 19, são idênticos aos encontrados para a linearização de Newton sem a regra de Armijo, figuras 14 e 15.

Na figura 18, a linearização de Newton novamente se sobrepôs à linearização por separação enquanto que na figura 19 ficou ligeiramente próxima. Se comparado com Reynolds 100, demonstrou ter se aproximado menos.

Observando-se a tabela 7 constata-se que a linearização de Newton com a velocidade u sendo avaliada pelos modelos 1 e 2 (tabela 2) e com a regra de Armijo em ambas

obtiveram baixo desempenho, tanto em número de iterações quanto em tempo de CPU necessários à convergência dos sistema de equações, se comparado com as mesmas avaliações para a velocidade u , porém, sem a regra de Armijo e também em relação à linearização por Separação.

Tipo de Linearização	Nº de Iterações	Tempo de CPU (minutos)
Separação	2325	32,17
Newton (modelo 1)	3305	58,45
Newton (Modelo 2)	2913	48,88

Tabela 7 : Linearização de Newton com Armijo x linearização por separação, $Re = 1000$

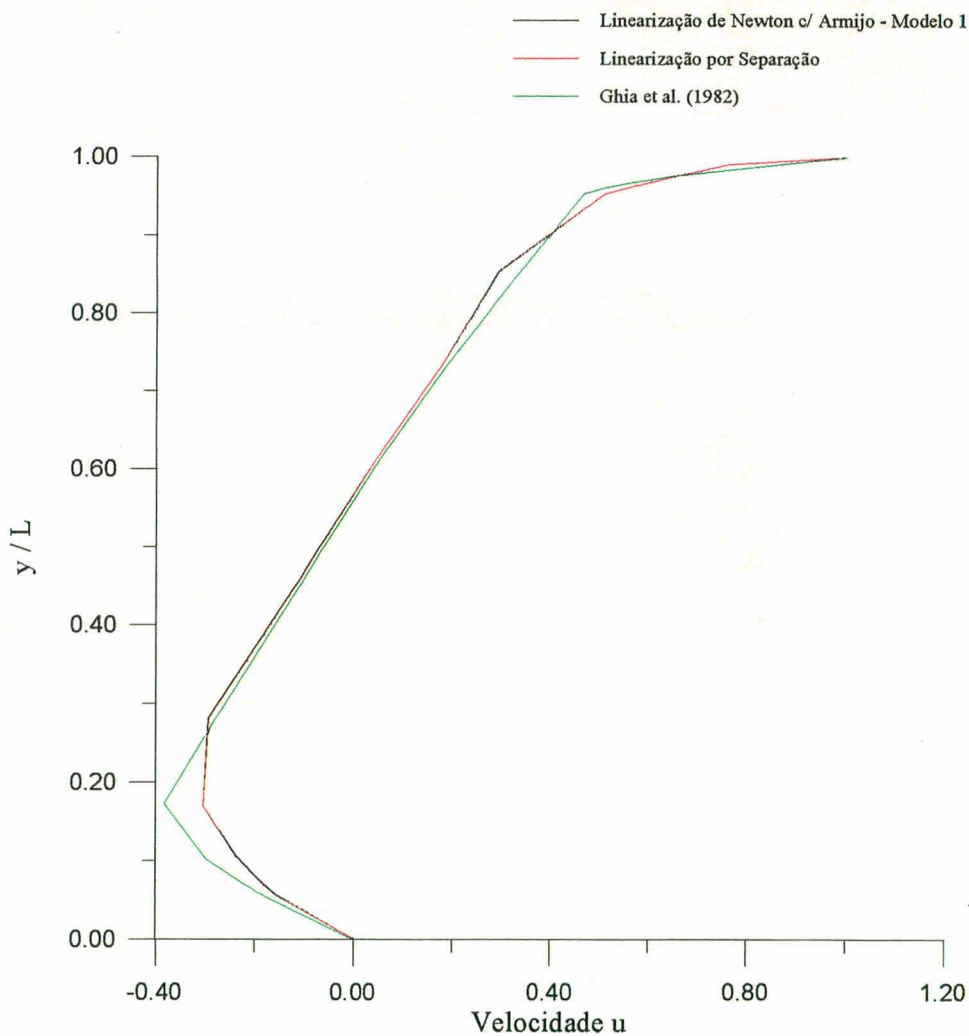


Figura 18 : Velocidade u ao longo de uma linha vertical central, $Re = 1000$ – Modelo 1

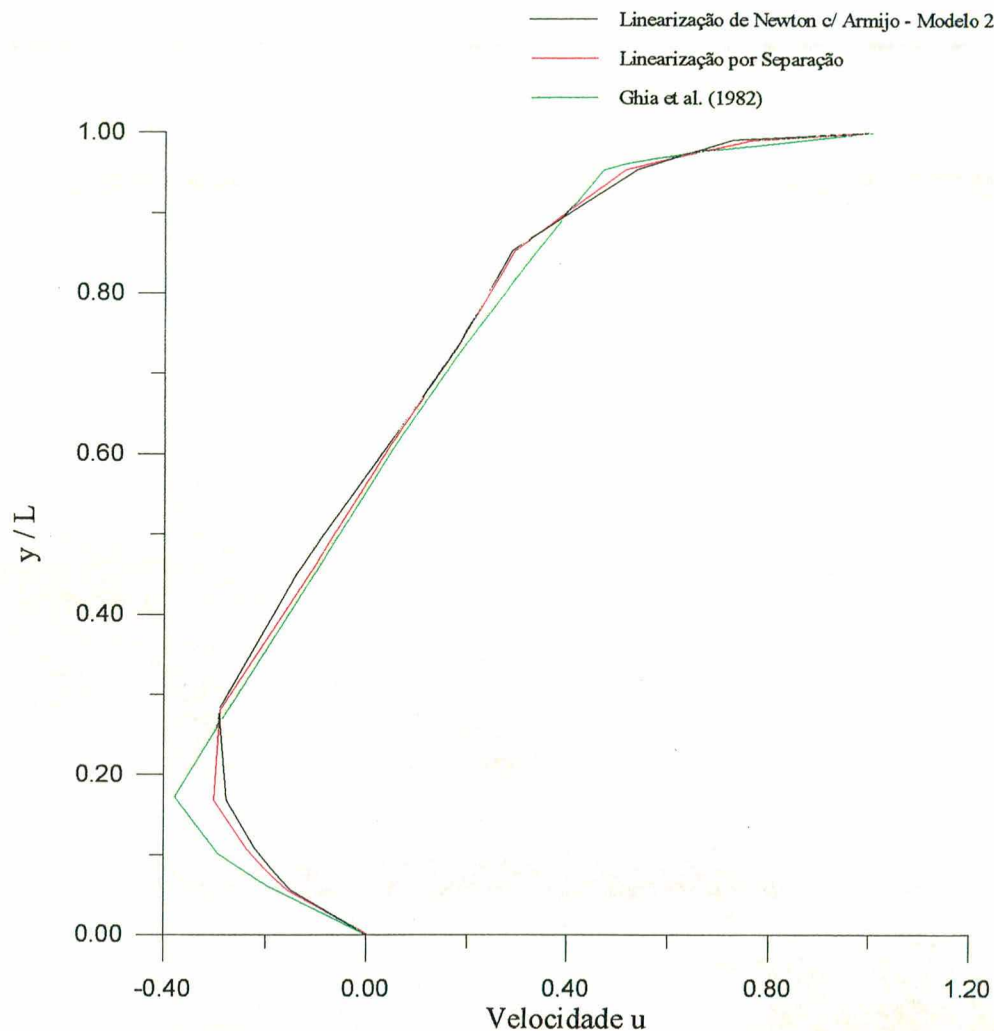


Figura 19 : Velocidade u ao longo de uma linha vertical central, $Re = 1000$ – Modelo 2

As figuras 20 e 21 apresentam o perfil da velocidade v ao longo de uma linha horizontal central obtido através da linearização de Newton com a regra de Armijo.

Na figura 20 a velocidade v foi avaliada através do modelo 1 (tabela 2), enquanto que na figura 21 a velocidade v foi avaliada através do modelo 2 (tabela 2). Ambos os gráficos comparam a linearização de Newton com a linearização por separação e com os resultados obtidos por Ghia.

Analisando as figuras 20 e 21 pode-se constatar que perfil da velocidade v indica que a linearização de Newton ficou próxima dos resultados obtidos por Ghia, assim como a linearização por separação manteve os mesmos resultados encontrados com Reynolds 100.

No gráfico 20 a linearização de Newton, mais uma vez, sobrepõe-se à linearização por separação em sua totalidade, o que não acontece no gráfico 21 onde a linearização de Newton mantém-se próxima. No entanto, o desempenho da linearização de Newton com a

velocidade v sendo avaliada através do modelo 1 (tabela 2) obteve desempenho inferior ao da avaliação através do modelo 2 (tabela 2), tal como ocorreu com Reynolds 100.

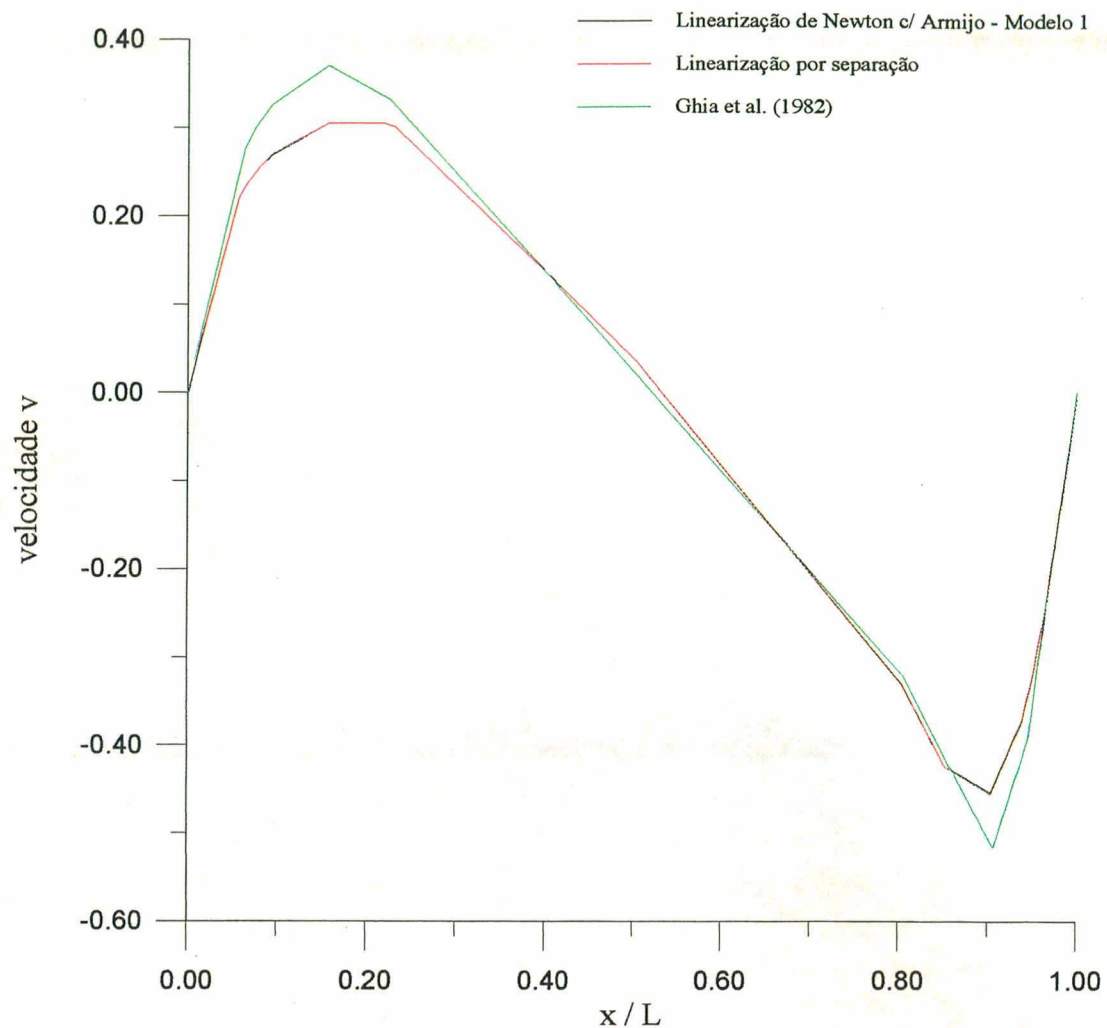


Figura 20 : Velocidade v ao longo de uma linha horizontal central, $Re = 1000$ – Modelo 1

Analisando os gráficos expostos nesta seção, pode-se concluir que os resultados obtidos com o perfil da velocidade u para suas diferentes avaliações aproximaram-se em relação aos resultados obtidos por Ghia, tal como Reynolds 100 sob as mesmas condições.

O desempenho da linearização de Newton sem a regra de Armijo e com a avaliação da velocidade u através do modelo 2 foi a que obteve melhores resultados em número de iterações necessárias à convergência. Entretanto, mais uma vez a regra de Armijo ficou aquém das expectativas.

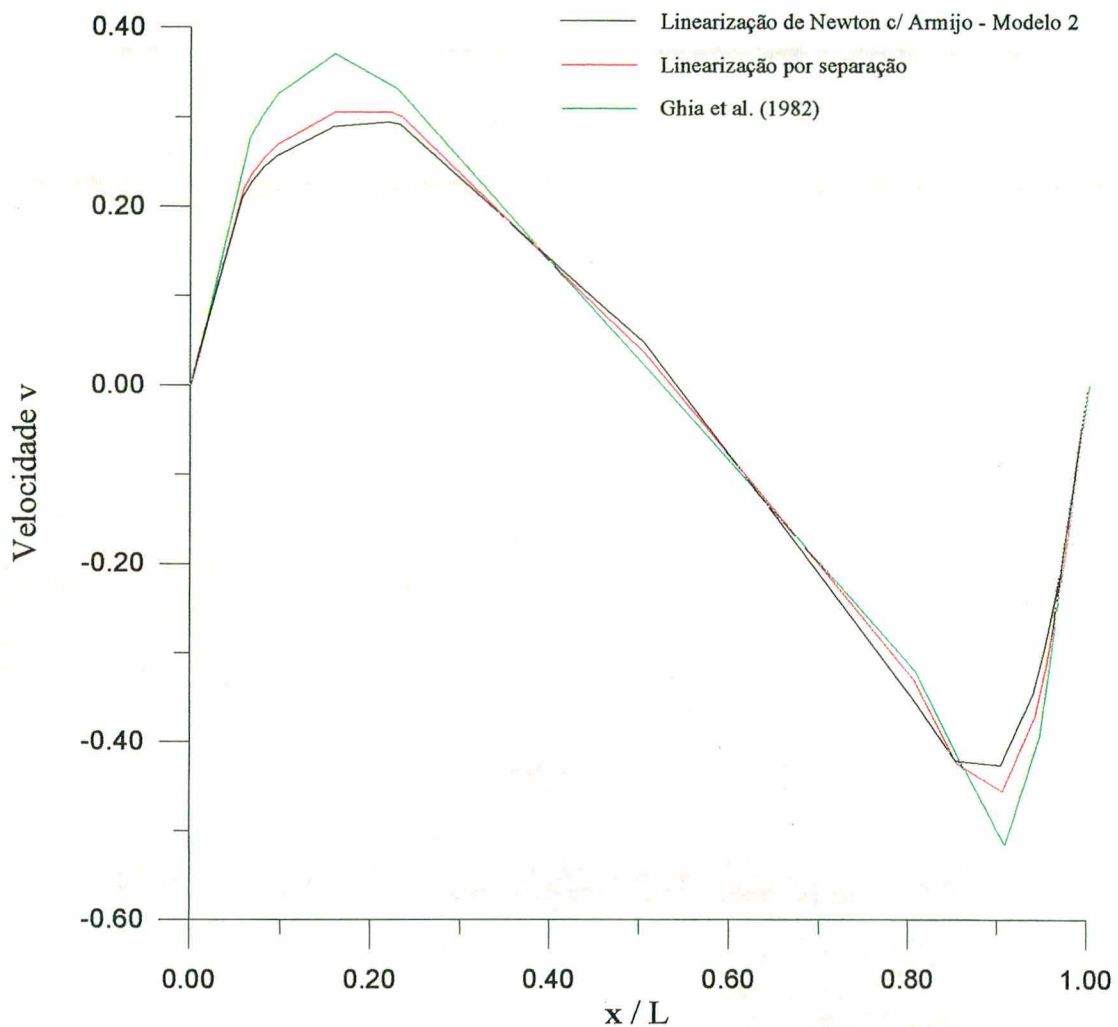


Figura 21 : Velocidade v ao longo de uma linha horizontal central, $Re=1000$ – Modelo 2

Analisando os resultados numéricos obtidos até o presente momento, pode-se concluir que os resultados obtidos com a linearização de Newton para Reynolds 1000 foram idênticos aos apresentados para Reynolds 100, isso considerando o perfil da velocidade u e v .

Novamente a regra de Armijo ficou aquém das expectativas. Por outro lado, a linearização de Newton com as velocidades u e v sendo avaliadas através do modelo 2 obteve desempenho ligeiramente superior à linearização por separação, enquanto que a avaliação das mesmas velocidades feita através do modelo 1 obteve o pior desempenho dentre as três.

A tabela 8, apresenta um comparativo dos testes realizados nesta seção, trazendo os tipos de linearizações (com as avaliações de u e v empregadas), nº de iterações e tempo de CPU necessários para a convergência do sistema de equações, um campo de velocidade num determinado ponto aleatório da malha (neste caso somente da velocidade u no ponto 35).

Tipo de Linearização		Nº de Iterações	Tempo de CPU (minutos)	Velocidade[35],u
Separação (com UDS)		2325	32,17	0,021100
Newton sem	Modelo 1	2516	37,27	0,020983
Armijo	Modelo 2	2247	32,62	0,023952
Newton com	Modelo 1	3305	58,45	0,020985
Armijo	Modelo 2	2913	48,88	0,023936

Tabela 8 : Tipo de Linearização versus nº de iterações – Re = 1000

4.3.3. Reynolds 10000

Para Reynolds 10000, inclusive com a regra de Armijo, a linearização através do método de Newton não convergiu. Ressalta-se que alguns valores (aleatórios) para a relaxação das velocidades u e v , para a pressão, bem como o número de iterações internas em que cada uma das variáveis (velocidades e pressão) são resolvidas durante cada iteração de acordo com o método utilizado (nesse caso o Gradiente Conjugado) foram testados, sem sucesso.

4.4 Conclusões

Com base nos resultados obtidos e apresentados até o momento, pode-se concluir que a linearização de Newton para Reynolds 100 sem a regra de Armijo, se comparada com a linearização por separação, Cardoso (1997) e Mariani (1997), obteve performance superior em número de iterações e tempo de CPU necessários para a convergência do sistema de equações. O desempenho da linearização de Newton foi significativo, em torno de 20% menos iterações com as velocidades u e v sendo avaliadas através do modelo 1 (tabela 2), ou 25% menos iterações se comparada com a avaliação das velocidades u e v através do modelo 2 (tabela 2), respectivamente. Por outro lado, a linearização de Newton avaliada através dos modelos 1 e 2 (tabela 2) apresentaram menor tempo de CPU necessário para a convergência do sistema de equações (tabela 3). Se for considerada somente a linearização de Newton com as velocidades u e v avaliadas através do modelo 1 (tabela 2) em comparação com a avaliação através do modelo 2 (tabela 2) a segunda foi mais rápida, em torno de 5% menos iterações e, também, apresentou menor tempo de CPU, conforme tabela 3.

Para Reynolds 1000 sem a regra de Armijo a diferença não foi muito significativa se comparada com a linearização por separação, Cardoso (1997) e Mariani (1997). A linearização de Newton com as velocidades u e v avaliadas através do modelo 1 (tabela 2) tiveram desempenho mais lento, em torno de 8% mais iterações, enquanto que a avaliação produzida através do modelo 2 (tabela 2) foi mais rápida, em torno de 3% menos iterações. Por outro lado, se for considerado somente o tempo de CPU a linearização por separação foi mais rápida que a linearização de Newton avaliada conforme a tabela 2, apesar de que a linearização de Newton com avaliação através do modelo 2 apresentou menor número de iterações. Se for considerada somente a linearização de Newton com as velocidades u e v avaliadas através do modelo 1 (tabela 2) em comparação com a avaliação através do modelo 2 (tabela 2), a segunda foi mais rápida, em torno de 10% menos iterações e, também, apresentou menor tempo de CPU.

Conforme foi dito anteriormente, a regra de Armijo ficou aquém das expectativas. Considerando Reynolds 100, a linearização de Newton com a regra de Armijo e com as velocidades u e v sendo avaliadas através do modelo 1 (tabela 2) tiveram desempenho inferior a linearização por separação, Cardoso (1997) e Mariani (1997), em torno de 9% mais iterações, enquanto que a avaliação das velocidades u e v do modelo 2 (tabela 2) produziu em torno de 6% mais iterações. Em ambos os casos (modelos 1 e 2) constatou-se um maior tempo de CPU necessário para a convergência do sistema de equações. Considerando somente a linearização de Newton, a mesma com as velocidades u e v avaliadas através do modelo 1 (tabela 2) foi mais lenta que a avaliação através do modelo 2 (tabela 2) em torno de 3% mais iterações, além de que apresentou maior tempo de CPU. Para Reynolds 1000 as diferenças são ainda maiores. Se for considerada a linearização de Newton com as velocidades u e v avaliadas através do modelo 1 (tabela 2) a diferença sobe para 30% mais iterações do que a linearização por separação, conforme Cardoso (1997) e Mariani (1997), enquanto que a avaliação produzida do modelo 2 (tabela 2) foi mais lenta, em torno de 20% mais iterações. Considerando apenas o tempo de CPU, novamente as diferenças são significativas, ou seja, a linearização por separação apresentou-se mais rápida. Considerando somente a linearização de Newton, a mesma com as velocidades u e v avaliadas através do modelo 1 (tabela 2) foi mais lenta que a avaliação através do modelo 2 (tabela 2) em torno de 12% mais iterações e, também, o modelo 2 apresentou um menor tempo de CPU.

A avaliação das velocidades u e v através do modelo 2 (tabela 2) mostrou-se numericamente mais estável do que a do modelo 1 (tabela 2). Assim como a equação para W_{ij} , equação (63), foi numericamente mais estável do que a equação (68).

Para o perfil das velocidades u e v apresentadas nas seções anteriores, é válido lembrar que a malha utilizada por Ghia tem em torno de 16641 pontos nodais, enquanto que a utilizada no presente trabalho possui 6400 pontos nodais dispostos em volumes hexagonais.

Ressalta-se, ainda, que o presente trabalho utilizou-se da resolução segregada dos sistema de equações para as equações (20) e (26) porque o método utilizado para a solução do sistema de equações, Gradiente Conjugado, da forma como foi implementado resolve as velocidade u e v separadamente.

O Gradiente Conjugado é indicado para sistemas cuja matriz é simétrica positiva e não é aplicável ao presente trabalho, porém, este foi utilizado e convergiu.

Capítulo 5 – Perspectivas para Trabalhos Futuros

Com a implementação da linearização de Newton na discretização das equações de Navier-Stokes com Diagramas de Voronoi surgem várias perspectivas para trabalhos futuros a serem desenvolvidos, dentre eles :

- Para uma análise mais justa da linearização de Newton na discretização das equações de Navier-Stokes com Diagramas de Voronoi, sugere-se a implementação da solução simultânea dos três sistemas de equações algébricas gerados : para a pressão e para as velocidades u e v e, novamente, comparar os resultados obtidos com essa nova implementação com os resultados obtidos com a linearização por separação, Mariani (1997) e Cardoso (1997). Ressalta-se que o presente trabalho utiliza a solução segregada do sistema de equações : resolve-se u_i separadamente de v_i desprezando-se a variação de v e fazendo $v_{ij} = v_{ij}^*$ na equação (20) e, também, resolvendo-se v_i separadamente de u_i desprezando-se a variação de u fazendo $u_{ij} = u_{ij}^*$ na equação (26).
- O sistema de equações lineares gerado da discretização é de alta ordem e consome muito tempo de CPU para ser resolvido. Assim, sugere-se resolver tal sistema através de outros métodos como, por exemplo, CGS, BI-CGSTAB, GMRES, TDMA, GAUSS-SEIDEL e comparar o desempenho com relação ao CG utilizado no presente trabalho, bem como os resultados obtidos por Mariani (1997).
- Testar o desempenho da linearização de Newton em outros tipos de malhas como, por exemplo, malhas com pontos nodais aleatórios, malhas estruturadas uniformes, com outros tipos de geometrias ou não e, também, com contorno arbitrário.
- Otimizar o atual programa (Apêndice D) objetivando obter maior organização do código computacional gerado, eficiência nos cálculos e no armazenamento das variáveis.
- Objetivando uma melhora na performance, paralelizar o código computacional gerado, o que é uma tarefa bastante complicada.

Referências Bibliográficas

- BEAM, R. M. and WARMING, R. F. **An Implicit finite Difference Algorithm for Hyperbolic Systems in Conservation-Law form.** Journal of Computational Physics, vol. 22, p. 87-110, 1976.
- BEAM, R. M. and WARMING, R. F. **An Implicit Factored Scheme for the Compressible Navier-Stokes Equations.** AIAA 3rd Computational Fluid Dynamics Conference, Albuquerque, NM, 1977.
- BEJAN, A. **Convection Heat Transfer.** John Wiley & Sons, 1984.
- BRILEY, W. R. and MCDONALD, H. **Solution of the Multidimensional Compressible Navier-Stokes Equations by a Generalized Implicit Method.** Journal of Computational Physics, vol. 24, p. 372-397, 1977.
- BRILEY, W. R. and MCDONALD, H. **On the Structure and Use of Linearized Block Implicit Schemes.** Journal of Computational Physics, vol. 34, p. 54-73, 1980.
- CARDOSO, F. C. **Algoritmo para Simulação Numérica de Equações do Movimento pelo Método dos Volumes Finitos usando Diagramas de Voronoi.** Dissertação de Mestrado aprovada pelo Curso de Pós-graduação em Ciência da Computação - UFSC, Florianópolis, Brasil, 1997.
- D'AMICO, M. **A Newton-Raphson Approach for Non-Linear Diffusion Equations in Radiation Hydrodynamics.** Journal of Quantitative Spectroscopy & Radiative Transfer, vol. 54, n° 4, p. 655-669, 1995.
- FAIRES, J. D., BURDEN, R. L. **Numerical Methods,** PWS Publishing Company, Boston, USA, 1993.
- GHIA, U., GHIA, K. N. and SHIN, C. T. **High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and Multigrid Method.** Journal of Computational Physics, vol. 48, p. 387-411, 1982.
- GOLUB, G. H. and O'LEARY, D. P. **Some History of the Conjugate Gradient and Lanczos Algorithms : 1948-1976.** SIAM Review, vol. 31, n° 1, p. 50-102, 1989.
- JOHAN, Z., HUGHES, T. J. R. and SHAKIB, F. **A Globally Convergent Matrix-Free Algorithm for Implicit Time-Marching Schemes Arising in Finite Element Analysis in**

- Fluids.** Computer Methods in Applied Mechanics and Engineering, vol. 2-3, n° 87, p. 281-304, 1990.
- HAGER, W. W. **Applied Numerical Linear Algebra.** Prentice-Hall Internacional, 1988.
- MACCORMICK, R. W. **Current Status of Numerical Solutions of the Navier-Stokes Equations.** AIAA 23rd Aerospace Sciences Meeting, 1985.
- MALISKA, C. R. **Transferência de Calor e Mecânica dos Fluidos Computacional.** Livros Técnicos e Científicos S. A., 1995.
- MALISKA JR. C. R. **Um Robusto Gerador de Diagramas de Voronoi para Discretização de Domínios Irregulares.** XIV Congresso Ibero Latino-Americano sobre Métodos Computacionais para a Engenharia, p. 548-547, Belo Horizonte, Brasil, 1994.
- MARCONDES, F. **Solução Numérica Usando Métodos Adaptativos-Implicitos e Malhas de Voronoi de Problemas de Reservatório de Petróleo.** Tese de Doutorado, Departamento de Engenharia Mecânica – UFSC, Florianópolis - Brasil, 1996.
- MARIANI, V. C. **Resolução de Sistemas Lineares Gerados na Discretização das Equações de Navier-Stokes em Malhas de Voronoi.** Dissertação de Mestrado aprovada pelo Curso de Pós-graduação em Ciência da Computação – UFSC, Florianópolis, Brasil, 1997.
- PATANKAR, S. V. **Numerical Heat Transfer and Fluid Flow.** Hemisphere-McGraw-Hill, 1980.
- PERIC, M., KESSELER, R. and SCHEUERER, G. **Comparison of Finite Volume Numerical Methods With Staggered and Colocated Grids.** Computers & Fluids, vol. 16, p. 389-403, 1988.
- PETERS, S. **Notas de Aulas da Disciplina de Volumes Finitos.** CPGCC – UFSC, Florianópolis, 1997.
- PETERS, S. **Notas de Aulas da Disciplina de Discretização de Equações Diferenciais com Diagrama de Voronoi.** CPGCC – UFSC, Florianópolis, 1997.
- PUTTI, M. and PANICONI, C. **Picard and Newton Linearization for the Coupled Model of Saltwater Intrusion in Aquifers.** Vol. 18, n° 3, p. 159-170, 1995.
- RAITHBY, P. F. and GALPIN, G. D. **Treatment of Non-Linearities in the Numerical Solution of the Incompressible Navier Stokes Equations.** International Journal for Numerical Methods in Fluids, vol. 6, p. 409-426, 1986.
- RONZANI, E. R. e NIECKELE, A. O. **Método de Solução Numérica de Escoamentos Incompressíveis em Geometrias Complexas.** Congresso Ibero Latino Americano sobre Métodos Computacionais para Engenharia, p. 41-50, Paraná, 1995.

-
- SANTOS, L. A. **O Desacoplamento Par-Ímpar do Campo de Pressão e Algoritmos para Simulação de Escoamentos Incompressíveis por Volumes Finitos.** Dissertação de Mestrado aprovada pelo Programa de Pós-Graduação em Engenharia Mecânica – UFSC, Florianópolis, Brasil, 1996.
- TANIGUCHI, N., and KOBAYASHI, T. **Finite Volume Method on the Unstructured Grid System.** *Computers & Fluids*, vol. 19, nº 34, p. 287-295.
- TANIGUCHI, N., ARAKAWA, C. and KOBAYASHI, T. **Construction of a Flow-Simulating Method With Finite Volume Based on a Voronoi Diagram.** *JSME International Journal, Série III*, vol. 34, p. 18-23, 1991.
- TANYI, B. A. and Thatcher, R. W. **Iterative Solutions of the Incompressible Navier-Stokes Equations on the Meiko Computing Surface.** *International Journal for Numerical Methods in Fluids*, vol. 22, p. 225-240, 1996.
- VERSTEEG, H. K., MALALASEKERA, W. **An Introduction to Computational Fluid Dynamics.** Longman Ltda, London, UK, 1995.

Apêndice A – Linearização na Solução Equações não Lineares

Em equações algébricas podemos achar as raízes das seguintes formas :

1) Pelo método direto (equação de Báskara) de separação explícita.

Exemplo :

$$x^2 - 4 = 0 \Rightarrow \begin{cases} x_1 = +2 \\ x_2 = -2 \end{cases}$$

Esta forma de linearização é limitada pois são poucas as equações que possuem solução direta.

2) Muitas vezes a variável envolvida não pode ser separada explicitamente. Neste caso, podemos propor uma separação implícita a partir de um valor estimado, com atualização sucessiva :

$$x^* \cdot x - 4 = 0$$

$$x = \frac{4}{x^*}$$

onde :

x^* é um valor estimado e x é o valor avaliado.

Logo, podemos gerar a tabela 9, partindo de $x^* = 10$ (arbitrário) e atualizando-o em cada iteração, onde constata-se um processo iterativo oscilatório não convergente e instável que pode ser melhorado com o uso de fatores de sub-relaxação.

K	x^*	$x = 4/x^*$
0	10	0.4
1	0.4	10
2	10	0.4
3	0.4	10

Tabela 9 : Processo de separação a partir de um valor estimado, com atualização sucessiva

3) Expansão em Série de Taylor em torno de x^* (Método de Newton) :

$$f(x) = x^2 - 4 = 0$$

$$f(x) = f(x^*) + \frac{\partial f}{\partial x}(x - x^*) + O^2$$

Desprezando os termos de ordem igual e superior a dois, tem-se:

$$f(x^*) + f'(x^*)(x - x^*) \cong 0$$

$$[(x^*)^2 - 4] + (2x^*)(x - x^*) = 0$$

$$x = x^* - \frac{(x^*)^2 - 4}{2x^*}$$

$$x = \frac{(x^*)^2 + 4}{2x^*}$$

Ou genericamente:

$$x = x^* - \frac{f(x^*)}{f'(x^*)}$$

Alternativamente, pode-se simplificar essa equação expandindo apenas o termo não linear x^2 em torno de x^* :

$$x^2 \cong (x^*)^2 + (2x^*)(x - x^*)$$

Logo :

$$x^2 - 4 = 0$$

$$x^2 - 4 = (x^*)^2 + 2x(x - x^*) - 4 = 0$$

$$x = x^* - \frac{(x^*)^2 - 4}{2x^*}$$

Então :

$$x = \frac{(x^*)^2 + 4}{2x^*}$$

Logo, podemos gerar a seguinte tabela, partindo de um valor $x^* = 10$ (arbitrário) e atualizando-o em cada iteração.

k	x^*	$x = [(x^*)^2 + 4] / 2x^*$
0	10	5.2
1	5.2	2.985
2	2.985	2.162
3	2.162	2.006
		≈ 2

Tabela 10 : Processo de expansão em Série de Taylor em torno de x^*

Constata-se uma convergência monotônica no processo iterativo anterior, chegando-se a uma solução rapidamente.

O sistema de equações gerados da discretização das EDP's de Navier-Stokes pelo MVF é de segunda ordem e necessita de uma forma de linearização para ser resolvida como um sistema linear de equações.

Dentre os três métodos iterativos expostos, o terceiro é o mais consistente pois não apresentou oscilações e, portanto, convergiu mais rapidamente.

Apêndice B - Regra de Armijo (Extrapolação Recursiva)

A Regra de Armijo introduzida no Método de Newton tem como objetivo acelerar a convergência do sistema $f(x) = 0$ (Hager, 1988).

O Método de Newton é dado por :

$$x_{k+1} = x_k - J(x_k)^{-1} f(x_k) \quad (71)$$

Seja s um parâmetro positivo e $y(s) = x_k - sz$, onde z é o incremento na solução dado por : $z = J(x_k)^{-1} f(x_k)$ (observe que $y(0) = x_k$ e $y(1) = x_{k+1}$). Tipicamente, a norma de $f(y(s))$ como uma função de s assemelha-se a figura 2, a seguir :

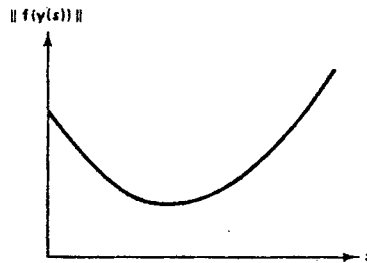


Figura 22 : Gráfico de s versus $\|f(y(s))\|$

Se x_k está aproximando-se da raiz então o mínimo na figura 2 é alcançado quando $s = 1$. Visto que $s = 1$ corresponde a x_{k+1} , segue que $\|f(x_{k+1})\| < \|f(x_k)\|$. Já que f desaparece perto da raiz, $\|f(x_k)\|$ tipicamente aproxima-se monotonicamente de zero quando a hipótese inicial é próxima da raiz.

Quando x_k está distante da raiz, o mínimo da figura 6 é, muitas vezes, alcançado com um valor de s menor do que 1 e pode acontecer que $\|f(y(1))\| > \|f(y(0))\|$, ou equivalentemente, $\|f(x_{k+1})\| > \|f(x_k)\|$. A desigualdade $\|f(x_{k+1})\| < \|f(x_k)\|$ será preservada se o tamanho do passo no Método de Newton for reduzido.

A Regra de Armijo's para determinar s adequado a convergência consiste em avaliar $\|f(y(s))\|$ para valores de $s = 1, 1/2, 1/4, \dots$, parando quando :

$$\|f(y(s))\| \leq \left(1 - \frac{s}{2}\right) \|f(x_k)\| \quad (72)$$

Seja t o primeiro valor de s que satisfaça a equação (72). Então, x_{k+1} será dado por :

$$x_{k+1} = x_k - tJ(x_k)^{-1}f(x_k) \quad (73)$$

Adota-se aqui, para avaliação de s na equação (72), a norma Euclidiana :

$$\|\vec{u}\| = (u_1^2 + u_2^2 + u_3^2 + \dots + u_n^2)^{1/2} \quad (74)$$

Apêndice C – Avaliação de dw_{ij}

A média aritmética simples envolvendo as eqs. (61) e (62) permitem obter uma equação para avaliar a velocidade da interface W_{ij} , através de uma sequência de aproximações. Para melhor ilustrar a obtenção de d_{ij}^w , apresentado na equação (66), tomam-se os termos de pressão de cada uma das componentes $(w_i)_{ij}$ e $(w_j)_{ij}$, de acordo com a média aritmética envolvida na eq. (63) (termo (**)) da equação (75) :

$$\text{termo(**)} = 0.5 * \left\{ -\Delta\vartheta_i \left[\left(\frac{1}{Ap_i^u} \right) (\nabla P)_i^x \text{ex}_{ij} + \left(\frac{1}{Ap_i^v} \right) (\nabla P)_i^y \text{ey}_{ij} \right] - \Delta\vartheta_j \left[\left(\frac{1}{Ap_j^u} \right) (\nabla P)_j^x \text{ex}_{ij} + \left(\frac{1}{Ap_j^v} \right) (\nabla P)_j^y \text{ey}_{ij} \right] \right\} \quad (75)$$

Para poder reduzir este termo com várias componentes de pressão a uma única pressão equivalente, pode-se substituir os coeficientes centrais de u e v, Ap_i^u e Ap_i^v , por um valor médio equivalente, considerando que apesar da linearização via série de Taylor, os termos Ap_i^u e Ap_i^v são da mesma magnitude.

Assim,

$$\frac{1}{Ap_i^u} \approx \frac{1}{Ap_i^v} \approx 0.5 * \left(\frac{1}{Ap_i^u} + \frac{1}{Ap_i^v} \right) = \left(\frac{1}{Ap_i} \right) \quad (76)$$

$$\frac{1}{Ap_j^u} \approx \frac{1}{Ap_j^v} \approx 0.5 * \left(\frac{1}{Ap_j^u} + \frac{1}{Ap_j^v} \right) = \left(\frac{1}{Ap_j} \right) \quad (77)$$

Desta forma pode-se separar estes termos colocando-os em evidência na média dada pela equação (75) :

$$\text{termo(**)} = 0.5 * \left\{ -\Delta\vartheta_i \left[\left(\frac{1}{Ap_i} \right) (\nabla P)_i^x \text{ex}_{ij} + \left(\frac{1}{Ap_i} \right) (\nabla P)_i^y \text{ey}_{ij} \right] - \Delta\vartheta_j \left[\left(\frac{1}{Ap_j} \right) (\nabla P)_j^x \text{ex}_{ij} + \left(\frac{1}{Ap_j} \right) (\nabla P)_j^y \text{ey}_{ij} \right] \right\} \quad (78)$$

$$\text{termo(**)} = 0.5 * \left\{ -\Delta\vartheta_i \left(\frac{1}{Ap_i} \right) \left[(\nabla P)_i^x \text{ex}_{ij} + (\nabla P)_i^y \text{ey}_{ij} \right] - \Delta\vartheta_j \left(\frac{1}{Ap_j} \right) \left[(\nabla P)_j^x \text{ex}_{ij} + (\nabla P)_j^y \text{ey}_{ij} \right] \right\} \quad (79)$$

As projeções das componentes de pressão na direção normal são denotadas da seguinte forma:

$$(\nabla P)_i^w = (\nabla P)_i^x \text{ex}_{ij} + (\nabla P)_i^y \text{ey}_{ij}$$

$$(\nabla P)_j^w = (\nabla P)_j^x \text{ex}_{ij} + (\nabla P)_j^y \text{ey}_{ij}$$

Assim,

$$\text{termo(**)} = 0.5 * \left\{ -\Delta\vartheta_i \left(\frac{1}{Ap_i} \right) (\nabla P)_i^w - \Delta\vartheta_j \left(\frac{1}{Ap_j} \right) (\nabla P)_j^w \right\}$$

Tomando uma média aritmética entre os termos dos vizinhos i e j e adotando um gradiente de pressão equivalente, como se a malha fosse desencontrada, tem-se um procedimento análogo ao proposto por Peric et alli (1988).

$$\Delta\vartheta_i \left(\frac{1}{Ap_i} \right) \approx \Delta\vartheta_j \left(\frac{1}{Ap_j} \right) \approx 0.5 * \left(\Delta\vartheta_i \left(\frac{1}{Ap_i} \right) + \Delta\vartheta_j \left(\frac{1}{Ap_j} \right) \right) = \overline{\left(\frac{\Delta\vartheta}{Ap} \right)}$$

$$\text{termo(**)} = 0.5 * \left\{ -\overline{\left(\frac{\Delta\vartheta}{Ap} \right)} (\nabla P)_i^w - \overline{\left(\frac{\Delta\vartheta}{Ap} \right)} (\nabla P)_j^w \right\}$$

$$\text{termo(**)} = -\overline{\left(\frac{\Delta\vartheta}{Ap} \right)} * 0.5 * \left\{ (\nabla P)_i^w + (\nabla P)_j^w \right\}$$

onde a média entre os gradientes de pressão em i e j pode ser substituído por um gradiente equivalente avaliado na interface entre os V.C. i e j:

$$0.5 * \{(\nabla P)_i^w + (\nabla P)_j^w\} = (\nabla P)_{ij}^w$$

Logo,

$$\text{termo(**)} = -\left(\overline{\frac{\Delta \mathcal{G}}{A_p}}\right) * (\nabla P)_{ij}^w = -d_{ij}^w (\nabla P)_{ij}^w$$

onde o termo $\left(\overline{\frac{\Delta \mathcal{G}}{A_p}}\right)$ constitui o d_{ij}^w apresentado pela equação (66), pois

$$\left(\overline{\frac{\Delta \mathcal{G}}{A_p}}\right) = 0.5 * \left[\Delta \mathcal{G}_i \left(\frac{1}{A_{p_i}} \right) + \Delta \mathcal{G}_j \left(\frac{1}{A_{p_j}} \right) \right]$$

$$\left(\overline{\frac{\Delta \mathcal{G}}{A_p}}\right) = 0.5 * \left[\Delta \mathcal{G}_i \left(0.5 * \left(\frac{1}{A_{p_i^u}} + \frac{1}{A_{p_i^v}} \right) \right) + \Delta \mathcal{G}_j \left(0.5 * \left(\frac{1}{A_{p_j^u}} + \frac{1}{A_{p_j^v}} \right) \right) \right]$$

$$\left(\overline{\frac{\Delta \mathcal{G}}{A_p}}\right) = 0.25 * \left[\Delta \mathcal{G}_i * \left(\frac{1}{A_{p_i^u}} + \frac{1}{A_{p_i^v}} \right) + \Delta \mathcal{G}_j * \left(\frac{1}{A_{p_j^u}} + \frac{1}{A_{p_j^v}} \right) \right]$$

$$d_{ij}^w = 0.25 * \left[\Delta \mathcal{G}_i * \left(\frac{1}{A_{p_i^u}} + \frac{1}{A_{p_i^v}} \right) + \Delta \mathcal{G}_j * \left(\frac{1}{A_{p_j^u}} + \frac{1}{A_{p_j^v}} \right) \right]$$

Apêndice D - Programa Computacional Implementado em Linguagem C

```
/******  
/* DEFINIÇÃO DAS ESTRUTURAS UTILIZADAS*/  
/******  
  
#ifndef __VORONOI_H  
  
#define __VORONOI_H  
  
#define N_ITERACOES 300 // Numero de iteracoes do algoritmo  
#define N_PONTOS 6441 // Numero de pontos da malha + 1  
#define MAX_VIZ 10 // Numero maximo de vizinhos + 1  
  
#define LEFT_WALL 6410 // Ponto na parede isolada, no lado esquerdo  
#define BOTTOM_WALL 6420 // Ponto na parede isolada, na parte de baixo  
#define RIGHT_WALL 6430 // Ponto na parede isolada, no lado direito  
#define TOP_WALL_SLIP 6440 // Ponto na parede deslizante, na parte de cima  
  
typedef struct{  
int n_pts; // Numero de vizinhos (e de vertices)  
  
int viz1[MAX_VIZ]; // Pontos vizinhos  
double x[MAX_VIZ], // Coordenadas dos vertices (pontos de Voronoi)  
y[MAX_VIZ];  
double cx, cy; // Coordenadas do ponto nodal  
double u;  
double uva;  
double z;  
double u_old;  
double v;  
double v_old;  
double p;  
double p_old;  
double p_lin;  
double Grad_Px;  
double Grad_Py;  
double Grad_Pw[MAX_VIZ];  
double Grad_px_lin;  
double Grad_py_lin;  
double vol_diag;  
double dw[MAX_VIZ];  
double deltavapi[MAX_VIZ];  
double deltavapj[MAX_VIZ];  
double termo[MAX_VIZ];  
int boolean;  
} tipo_volume;  
  
typedef struct{  
double Rho[MAX_VIZ];  
double rho;  
double Gama[MAX_VIZ];
```

```

double gama;
double L[MAX_VIZ];
double Soma_L;
double Soma_Lx;
double Soma_Ly;
double S[MAX_VIZ];
double ex[MAX_VIZ];
double ey[MAX_VIZ];
double W[MAX_VIZ];
double Soma_gx;
double Soma_gy;
double depois[MAX_VIZ];
double antes[MAX_VIZ];
double diferenca[MAX_VIZ];
} tipo_param;

typedef struct{
double F[MAX_VIZ];
double D[MAX_VIZ];
double AU[MAX_VIZ];
double AV[MAX_VIZ];
double auxu[MAX_VIZ];
double auxv[MAX_VIZ];
double a[MAX_VIZ];
double A0;
double Apu;
double Apv;
double ap;
double bu;
double bv;
double pos[MAX_VIZ];
double neg[MAX_VIZ];
} tipo_coefic;

#endif

/*****
/* RESOLUCAO DO SISTEMA DE EQUACOES PELO METODO CG */
*****/

#include <iostream.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <stdio.h>
#include <math.h>
#include "voronoi.h"

//DECLARACAO DAS VARIAVEIS USADAS NO CG
int num_pts, num_rep_p=3, num_rep_u_v=2;
tipo_volume volume[N_PONTOS];
tipo_param param[N_PONTOS];
tipo_coefic coefic[N_PONTOS];
double Soma_A[N_PONTOS];
double S[N_PONTOS], pAp;
static int k, i, j;
double aux1, aux2, alfa, beta, rho_0, rho, vz, errov, erro, erro_lin;
double r[N_PONTOS], p[N_PONTOS], aux[N_PONTOS], z[N_PONTOS];

```

```

double w[N_PONTOS], v[N_PONTOS], q[N_PONTOS], u[N_PONTOS];

/*****
/* DEFINICAO DA FUNCAO QUE CALCULA A VELOCIDADE U PELO CG*/
*****/

void CG_u(tipo_volume volume[]){

aux1 = 0.0;
for(i=1; i<=num_pts; i++){
  aux[i] = 0.0;
  for(j=1; j<=volume[i].n_pts; j++){
    aux[i] += (coefic[i].AU[j] * volume[volume[i].viz1[j]].u);
  }
  r[i] = (coefic[i].bu - (volume[i].vol_diag * volume[i].Grad_Px)) - ((coefic[i].Apu * volume[i].u) - aux[i]);
  p[i] = 0.0;
  aux1 += (r[i] * r[i]);
}

aux1 = sqrt((double)aux1);
rho_0 = 1.0;

for(k=1; k<=num_rep_u_v; k++){
  rho = 0.0;
  for(i=1; i<=num_pts; i++)
    rho += (r[i] * r[i]);

  beta = rho/rho_0;
  for(i=1; i<=num_pts; i++)
    p[i] = r[i] + (beta * p[i]);

  for(i=1; i<=num_pts; i++){
    aux[i] = 0.0;
    for(j=1; j<=volume[i].n_pts; j++){
      aux[i] += (coefic[i].AU[j] * p[volume[i].viz1[j]]);
    }
  }

  pAp = 0.0;
  for(i=1; i<=num_pts; i++){
    aux[i] = (coefic[i].Apu * p[i]) - aux[i]; //A*p
    pAp += (p[i] * aux[i]);
  }

  aux2 = 0.0;
  alfa = rho/pAp;
  for(i=1; i<=num_pts; i++){
    volume[i].u += (alfa * p[i]); //nova temperatura
    r[i] -= (alfa * aux[i]); //novo residuo
    aux2 += (r[i] * r[i]);
  }

rho_0 = rho;
} //fecha o for k
} //fecha o CG

/*****
/* DEFINICAO DA FUNCAO QUE CALCULA A VELOCIDADE V PELO CG */
*****/

void CG_v(tipo_volume volume[]){

```

```

aux1 = 0.0;
for(i=1; i<=num_pts; i++){
    aux[i] = 0.0;
    for(j=1; j<=volume[i].n_pts; j++){
        aux[i] += (coefic[i].AV[j] * volume[volume[i].viz1[j]].v);
    }

    r[i] = (coefic[i].bv - (volume[i].vol_diag * volume[i].Grad_Py)) - ((coefic[i].Apv * volume[i].v) - aux[i]);
    p[i] = 0.0;
    aux1 += (r[i] * r[i]);
}

aux1 = sqrt((double)aux1);
rho_0 = 1.0;

for(k=1; k<=num_rep_u_v; k++){
    rho = 0.0;
    for(i=1; i<=num_pts; i++)
        rho += (r[i] * r[i]);

    beta = rho/rho_0;
    for(i=1; i<=num_pts; i++)
        p[i] = r[i] + (beta * p[i]);

    for(i=1; i<=num_pts; i++){
        aux[i] = 0.0;
        for(j=1; j<=volume[i].n_pts; j++){
            aux[i] += (coefic[i].AV[j] * p[volume[i].viz1[j]]);
        }
    }

    pAp = 0.0;
    for(i=1; i<=num_pts; i++){
        aux[i] = (coefic[i].Apv * p[i]) - aux[i]; //A*p
        pAp += (p[i] * aux[i]);
    }

    aux2 = 0.0;
    alfa = rho/pAp;
    for(i=1; i<=num_pts; i++){
        volume[i].v += (alfa * p[i]); //nova temperatura
        r[i] = r[i] - (alfa * aux[i]); //novo residuo
        aux2 += (r[i] * r[i]);
    }

    rho_0 = rho;
} //fechar o for k
} //fechar o CG

/*****
/* DEFINICAO DA FUNCAO QUE CALCULA PRESSAO PELO CG */
*****/

void CG_p_lin(tipo_volume volume[], tipo_coefic coefic[]){
int k;
double Soma_RWS[N_PONTOS];
double pref_lin=0;

```

```

for(i=1; i<=num_pts; i++){
  Soma_RWS[i] = 0.0;
  for(j=1; j<=volume[i].n_pts; j++){
    Soma_RWS[i] += (param[i].Rho[j] * param[i].W[j] * param[i].S[j]);
  }
}

aux1 = 0.0;
for(i=1; i<=num_pts; i++){
  aux[i] = 0.0;
  for(j=1; j<=volume[i].n_pts; j++){
    if(volume[i].viz1[j] <= num_pts){
      aux[i] += (coefic[i].a[j] * volume[volume[i].viz1[j]].p_lin);
    }
  }
  r[i] = - Soma_RWS[i] - ((coefic[i].ap * volume[i].p_lin) - aux[i]);
  p[i] = 0.0;
  aux1 += (r[i] * r[i]);
}

aux1 = sqrt((double)aux1);
rho_0 = 1.0;

for(k=1; k<=num_rep_p; k++){
  rho = 0.0;
  for(i=1; i<=num_pts; i++)
    rho += (r[i] * r[i]);

  beta = rho/rho_0;
  for(i=1; i<=num_pts; i++)
    p[i] = r[i] + (beta * p[i]);

  for(i=1; i<=num_pts; i++){
    aux[i] = 0.0;
    for(j=1; j<=volume[i].n_pts; j++){
      aux[i] += (coefic[i].a[j] * p[volume[i].viz1[j]]);
    }
  }

  pAp = 0.0;
  for(i=1; i<=num_pts; i++){
    aux[i] = (coefic[i].ap * p[i]) - aux[i]; //A*p
    pAp += (p[i] * aux[i]);
  }

  aux2 = 0.0;
  alfa = rho/pAp;
  for(i=1; i<=num_pts; i++){
    volume[i].p_lin += (alfa * p[i]); //nova temperatura
    r[i] -= (alfa * aux[i]); //novo residuo
    aux2 += (r[i] * r[i]);
  }

  rho_0 = rho;
} //fechar o for k
} //fechar o CG p_lin

/*****
/*PROGRAMA PRINCIPAL*/
*****/

```

```

#include <math.h>
#include <stdio.h>
#include "voronoi.h"
/*#include "cgs.h"*/
#include "cg.h"

// DEFINICAO DAS CONSTANTES GLOBAIS AO SISTEMA
const double DELTA_t = 1e30;
const double ALFA_p_lin = 0.5;
const double ALFA_p = 0.5;
const double ALFA = 0.7;
const double ERRO = 1e-5;
const double REYNOLDS = 1000.0;
const int medida_x = 10;

// DECLARACAO DE VARIAVEIS GLOBAIS AO SISTEMA
double pos, neg;
double norma;
double norm,norma0, normal;
double termo1, termo2, s, su, sv, somai, somaj, mediai, mediaj;
int num_iter;

double Soma_1[N_PONTOS], Soma_2[N_PONTOS], Soma_3[N_PONTOS], Soma_4[N_PONTOS],
      Soma_5[N_PONTOS];
double g[N_PONTOS][MAX_VIZ];
int grad_type = 4;          // tipo do gradiente a ser utilizado
                          // = 1 -> Maliska
                          // = 2 -> Cardoso
                          // = 3 -> Jameson e Mavriplis
                          // = 4 -> Taniguchi et alli
                          // = 5 -> Dissertacao

// DEFINICAO DA FUNCAO QUE RETORNA O SINAL DE Fij
double retorna_sinal(double sig){

if (sig <= 0.0){
    pos = 0.0;
    neg = 1.0;
    return pos,neg;
}
else{
    pos = 1.0;
    neg = 0.0;
    return pos,neg;
}
}

// DEFINICAO DA FUNCAO QUE LE OS N_PONTOS NODAIS
void le_coordenadas_pontos_nodais(){
double pass1, pass2;
FILE *fp;
if ((fp = fopen("pontos.vor", "r")) == NULL){
    printf ("\nArquivo pontos.vor nao encontrado.\n");
    exit(1);
}

fscanf (fp, "%d\n", &num_pts);
printf ("\nLendo arquivo pontos.vor...");
for (i=1; i<=num_pts; i++){

```

```

    fscanf (fp, "%lf\t%lf\n", &pass1, &pass2);
    volume[i].cx = pass1;
    volume[i].cy = pass2;
}
fclose(fp);
}

// DEFINICAO DA FUNCAO QUE LE OS PONTOS NODAIS VIZINHOS PARA OS N_PONTOS
void le_vizinhos(){
int pass3, pass4;
FILE *fp;

if ((fp = fopen("vizinhos.vor", "r")) == NULL){
    printf ("\nArquivo vizinhos.vor nao encontrado.\n");
    exit(1);
}
printf ("\nLendo arquivo vizinhos.vor...");
for (i=1; i<=num_pts; i++){
    fscanf (fp, "%d",&pass3);
    volume[i].n_pts = pass3;
    for (j=1; j<=volume[i].n_pts; j++){
        fscanf (fp, "%d ",&pass4);
        volume[i].viz1[j] = pass4;
    }
}
fclose(fp);
}

// DEFINICAO DA FUNCAO QUE LE OS VERTICES DOS VOLUMES DE CADA PONTO NODAL
void le_vertices(){
int pass5=0;
double pass6, pass7;
FILE *fp;

if ((fp = fopen("vertices.vor", "r")) == NULL){
    printf ("\nArquivo vertices.vor nao encontrado.\n");
    exit(1);
}

printf ("\nLendo arquivo vertices.vor...");
for (i=1; i<=num_pts; i++){
    fscanf (fp, "%d\n",&pass5);
    for (j=1; j<=volume[i].n_pts+1; j++){
        fscanf (fp, "%lf\t%lf",&pass6, &pass7);
        volume[i].x[j] = pass6;
        volume[i].y[j] = pass7;
    }
}
fclose(fp);
}

// DEFINICAO DA FUNCAO QUE LE A MALHA
void le_malha(){
le_coordenadas_pontos_nodais();
le_vizinhos();
le_vertices();
}

// DEFINICAO DA FUNCAO QUE ATRIBUI OS PARAMETROS INICIAIS
void atribui_param_iniciais(tipo_volume volume[], tipo_param param[]){

```



```

volume[num_pts + 10].u = 0.0;
volume[num_pts + 20].u = 0.0;
volume[num_pts + 30].u = 0.0;
volume[num_pts + 40].u = 1.0;
volume[num_pts + 10].v = 0.0;
volume[num_pts + 20].v = 0.0;
volume[num_pts + 30].v = 0.0;
volume[num_pts + 40].v = 0.0;
volume[num_pts + 10].p = 0.0;
volume[num_pts + 20].p = 0.0;
volume[num_pts + 30].p = 0.0;
volume[num_pts + 40].p = 0.0;
volume[num_pts + 10].p_lin = 0.0;
volume[num_pts + 20].p_lin = 0.0;
volume[num_pts + 30].p_lin = 0.0;
volume[num_pts + 40].p_lin = 0.0;
param[num_pts + 10].gama = 32.10/REYNOLDS;
param[num_pts + 20].gama = 32.10/REYNOLDS;
param[num_pts + 30].gama = 32.10/REYNOLDS;
param[num_pts + 40].gama = 32.10/REYNOLDS;
param[num_pts + 10].rho = 1.0;
param[num_pts + 20].rho = 1.0;
param[num_pts + 30].rho = 1.0;
param[num_pts + 40].rho = 1.0;
S[num_pts + 10] = 0.0;
S[num_pts + 20] = 0.0;
S[num_pts + 30] = 0.0;
S[num_pts + 40] = 0.0;

for (i=1; i<=num_pts; i++){
    volume[i].p = 0.0;
    volume[i].u = 0.5;
    volume[i].v = 0.5;
    param[i].gama = 32.10/REYNOLDS;
    param[i].rho = 1.0;
    S[i] = 0.0;
}
}

// DEFINICAO DA FUNCAO QUE CALCULA A DISTANCIA ENTRE OS PONTOS NODAIS E
// A DISTANCIA ENTRE OS PONTOS DOS VERTICES DE CADA VOLUME (AREA DA FACE)
void calcula_L_e_S(tipo_volume volume[], tipo_param param[]){
for (i=1; i<=num_pts; i++){
    int n_pts = volume[i].n_pts;
    for (j=1; j<=n_pts; j++){
        param[i].S[j] = sqrt(pow(volume[i].x[j+1] - volume[i].x[j], 2) +
pow(volume[i].y[j+1] - volume[i].y[j], 2));
        int viz1 = volume[i].viz1[j];
        switch (viz1){
            case BOTTOM_WALL:{
                param[i].L[j] = volume[i].cy;
            }
            break;
            case RIGHT_WALL:{
                param[i].L[j] = 32.10 - volume[i].cx;
            }
            break;
            case LEFT_WALL:{
                param[i].L[j] = volume[i].cx;
            }
        }
    }
}
}

```

```

    }
    break;
    case TOP_WALL_SLIP:{
        param[i].L[j] = 32.10 - volume[i].cy;
    }
    break;
    default:{
        param[i].L[j] = sqrt(pow((volume[viz1].cx - volume[i].cx), 2) + pow((volume[viz1].cy -
volume[i].cy), 2));}
    }
}
}

// DEFINICAO DA FUNCAO QUE CALCULA A NORMA DO VETOR NORMAL DE CADA FACE DOS
VOLUMES DE CONTROLE
void calcula_norma(tipo_volume volume[], tipo_param param[]){
double norma;
double delta_x;
double delta_y;

for (i=1; i<=num_pts; i++){
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts){
            delta_x = volume[volume[i].viz1[j]].cx - volume[i].cx;
            delta_y = volume[volume[i].viz1[j]].cy - volume[i].cy;
            norma = sqrt((delta_x * delta_x) + (delta_y * delta_y));
            param[i].ex[j] = (delta_x)/norma;
            param[i].ey[j] = (delta_y)/norma;
        }
        else{
            if (volume[i].viz1[j] == (num_pts + 40)){
                param[i].ex[j] = 0.0;
                param[i].ey[j] = -1.0;
            }
            if (volume[i].viz1[j] == (num_pts + 30)){
                param[i].ex[j] = 1.0;
                param[i].ey[j] = 0.0;
            }
            if (volume[i].viz1[j] == (num_pts + 20)){
                param[i].ex[j] = 0.0;
                param[i].ey[j] = 1.0;
            }
            if (volume[i].viz1[j] == (num_pts + 10)){
                param[i].ex[j] = -1.0;
                param[i].ey[j] = 0.0;
            }
        }
    }
}

/*CUIDADO SE O CONTOURNO FOR DEFINIDO POR RETAS HORIZONTAIS OU VERTICAIS, DARA'
ERRO: DIVISAO POR ZERO */
/* angulo_teta = arctan (-(volume[i].x[j+1] - volume[i].x[j])/(volume[i].y[j] - volume[i].y[j-1]));
param[i].ex[j] = cos (angulo_teta);
param[i].ey[j] = sin (angulo_teta); */
}
}
}

// DEFINICAO DA FUNCAO QUE CALCULA O SOMATORIO DO Lij
void soma_L(tipo_volume volume[], tipo_param param[]){
for (i=1; i<=num_pts; i++){

```

```

    param[i].Soma_L = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts){
            param[i].Soma_L += param[i].L[j];
        }
    }
}
}

// DEFINICAO DA FUNCAO QUE CALCULA O SOMATORIO DO Lij PROJETADO NA DIRECAO x
void soma_Lx(tipo_volume volume[], tipo_param param[]){
for (i=1; i<=num_pts; i++){
    param[i].Soma_Lx = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts){
            param[i].Soma_Lx += (param[i].L[j] * fabs(param[i].ex[j]));
        }
    }
}
}

// DEFINICAO DA FUNCAO QUE CALCULA O SOMATORIO DO Lij PROJETADO NA DIRECAO y
void soma_Ly(tipo_volume volume[], tipo_param param[]){
for (i=1; i<=num_pts; i++){
    param[i].Soma_Ly = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts)
            param[i].Soma_Ly += (param[i].L[j] * fabs(param[i].ey[j]));
    }
}
}

// DEFINICAO DA FUNCAO QUE CALCULA A AREA DE UM TRIANGULO ATRAVES DO
// DETERMINANTE DE UMA MATRIZ FORMADA PELA PRIMEIRA
// COLUNA DE 1S (UNS) E AS OUTRAS PELOS VERTICES DE UM TRIANGULO
double determinante(double xa, double ya, double xb, double yb, double xc, double yc){
double A;
A = ((yc * xb) + (yb * xa) + (ya * xc) - ((ya * xb) + (yb * xc) + (yc * xa))) * 0.5;
return A;
}

// DEFINICAO DA FUNCAO QUE CALCULA O VOLUME DE UM DIAGRAMA DE VORONOI
// DIVIDINDO-O EM TRIANGULOS
void calcula_volume_diagrama(tipo_volume volume[], tipo_param param[]){
double Area, x1, x2, x3, y1, y2, y3;
for (i=1; i<=num_pts; i++){
    volume[i].vol_diag = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        x3 = volume[i].x[j];
        x2 = volume[i].x[j+1];
        x1 = volume[i].cx;
        y3 = volume[i].y[j];
        y2 = volume[i].y[j+1];
        y1 = volume[i].cy;
        Area = determinante(x1, y1, x2, y2, x3, y3);
        volume[i].vol_diag += Area/*param[i].S[j] * param[i].L[j] * 0.25;*/
    }
}
}

```

```

// DEFINICAO DA FUNCAO QUE CALCULA OS PARAMETROS GEOMETRICOS
void calcula_param_geom(tipo_volume volume[], tipo_param param[]){
calcula_L_e_S(volume, param);

switch (grad_type){
    case 1:{
        soma_L(volume, param);
    }
    break;
    case 2:{
        soma_Lx(volume, param);
        soma_Ly(volume, param);
    }
    break;
}
calcula_volume_diagrama(volume,param);
}

// DEFINICAO DA FUNCAO QUE CALCULA A MASSA ESPECIFICA
void calcula_Rho(tipo_volume volume[], tipo_param param[]){
for (i=1; i<=num_pts; i++){
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts){
            param[i].Rho[j] = 0.5 * (param[i].rho + param[volume[i].viz1[j]].rho);
        }
        else{
            param[i].Rho[j] = 1.0;
        }
    }
}
}

// DEFINICAO DA FUNCAO QUE CALCULA O COEFICIENTE DE CONDUTIVIDADE
void calcula_Gama(tipo_volume volume[], tipo_param param[]){
for (i=1; i<=num_pts; i++){
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts){
            param[i].Gama[j] = (2 * param[i].gama * param[volume[i].viz1[j]].gama)
            / (param[i].gama + param[volume[i].viz1[j]].gama);
        }
        else{
            param[i].Gama[j] = 32.10/REYNOLDS;
        }
    }
}
}

// DEFINICAO DA FUNCAO QUE CALCULA O COEFICIENTE TEMPORAL
void calcula_A0(tipo_volume volume[], tipo_coefic coefic[], tipo_param param[]){
for (i=1; i<=num_pts; i++){
    coefic[i].A0 = (param[i].rho * volume[i].vol_diag)/DELTA_t;
}
}

// DEFINICAO DA FUNCAO QUE CALCULA O FLUXO DIFUSIVO
void calcula_D(tipo_volume volume[], tipo_param param[], tipo_coefic coefic[]){
for (i=1; i<=num_pts; i++){
    for (j=1; j<=volume[i].n_pts; j++){
        coefic[i].D[j] = (param[i].Gama[j] * param[i].S[j])/param[i].L[j];
    }
}
}

```

```

}

// DEFINICAO DA FUNCAO QUE CALCULA OS SOMATORIOS PARA CALCULO DO GRADIENTE NA
// DIRECAO x E y PELO METODO DE TANIGUCHI
void calcula_soma_Tan(){
for (i=1; i<=num_pts; i++){
    for (j=1; j<=volume[i].n_pts; j++){
        g[i][j] = param[i].S[j]/param[i].L[j];
    }
for (i=1; i<=num_pts; i++){
    Soma_1[i] = 0.0;
    Soma_2[i] = 0.0;
    Soma_4[i] = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts){
            Soma_1[i] += (g[i][j] * param[i].ex[j] * param[i].ex[j]);
            Soma_2[i] += (g[i][j] * param[i].ex[j] * param[i].ey[j]);
            Soma_4[i] += (g[i][j] * param[i].ey[j] * param[i].ey[j]);
        }
    }
}
}
}

```

```

// DEFINICAO DA FUNCAO QUE CALCULA A VELOCIDADE NODAL NORMAL A FACEij PARA A
PRIMEIRA ITERACAO

```

```

void calcula_W(tipo_volume volume[], tipo_param param[], tipo_coefic coefic[]){
double wi;
double wj;

for (i=1; i<=num_pts; i++){
    for (j=1; j<=volume[i].n_pts; j++){
        wi = 0.0;
        wj = 0.0;
        param[i].W[j] = 0.0;
        if (volume[i].viz1[j] <= num_pts){
            wi = (volume[i].u * param[i].ex[j]) + (volume[i].v * param[i].ey[j]);
            wj = (volume[volume[i].viz1[j]].u * param[i].ex[j]) + (volume[volume[i].viz1[j]].v * param[i].ey[j]);
            param[i].W[j] = 0.5 * (wi + wj);
        }
        else{
            param[i].W[j] = 0.0;
        }
    }
}
}
}

```

```

// DEFINICAO DA FUNCAO QUE CALCULA A VELOCIDADE NODAL NORMAL A FACEij PARA A
PRIMEIRA ITERACAO

```

```

void calcula_W2(tipo_volume volume[], tipo_param param[], tipo_coefic coefic[]){
int ki, kj;
double soma1, soma2;
double soma_Aikuwi;
double soma_Aikvwi;
double soma_Ajkuwj;
double soma_Ajkvwj;

for (i=1; i<=num_pts; i++){
    for (j=1; j<=volume[i].n_pts; j++){
        soma1 = 0.0;
        soma2 = 0.0;
        ki = 0;

```

```

kj = 0;
param[i].W[j] = 0.0;
soma_Aikuwi = 0.0;
soma_Aikvwi = 0.0;
soma_Ajkuwj = 0.0;
soma_Ajkvwj = 0.0;

if (volume[i].viz1[j] <= num_pts){
  for (ki = 1; ki <= volume[i].n_pts; ki++){
    soma_Aikuwi += coefic[i].AU[ki] * volume[volume[i].viz1[ki]].u;
    soma_Aikvwi += coefic[i].AV[ki] * volume[volume[i].viz1[ki]].v;
  }
  for (kj = 1; kj <= volume[volume[i].viz1[j]].n_pts; kj++){
    soma_Ajkuwj += coefic[volume[i].viz1[j]].AU[kj] * volume[volume[volume[i].viz1[j]].viz1[kj]].u;
    soma_Ajkvwj += coefic[volume[i].viz1[j]].AV[kj] * volume[volume[volume[i].viz1[j]].viz1[kj]].v;
  }
  soma1 = (soma_Aikuwi + coefic[i].bu) * param[i].ex[j] / coefic[i].Apu + (soma_Aikvwi + coefic[i].bv) *
param[i].ey[j] / coefic[i].Apv;
  soma2 = (soma_Ajkuwj + coefic[volume[i].viz1[j]].bu) * param[i].ex[j] / coefic[volume[i].viz1[j]].Apu
+ (soma_Ajkvwj + coefic[volume[i].viz1[j]].bv) * param[i].ey[j] / coefic[volume[i].viz1[j]].Apv;
  param[i].W[j] = (( soma1 + soma2 ) * 0.5) - (volume[i].dw[j] * (volume[volume[i].viz1[j]].p - volume[i].p))
/ param[i].L[j];
  /*retorna sinal(coefic[i].F[j]);
  param[i].W[j] = (( soma1 * pos) + (soma2 * neg)) - (volume[i].dw[j] * (volume[volume[i].viz1[j]].p -
volume[i].p)) / param[i].L[j];*/
}
else{
  param[i].W[j] = 0.0;
}
}
}
}

// DEFINICAO DA FUNCAO QUE CALCULA O FLUXO CONVECTIVO
void calcula_F(tipo_volume volume[], tipo_param param[], tipo_coefic coefic[]){
for (i=1; i<=num_pts; i++){
  for (j=1; j<=volume[i].n_pts; j++){
    coefic[i].F[j] = param[i].Rho[j] * param[i].W[j] * param[i].S[j];
  }
}
}

//DEFINICAO DA ROTINA QUE CALCULA OS COEFICIENTE AUXU E AUXV
void calcula_coefic_a(tipo_volume volume[], tipo_param param[], tipo_coefic coefic[]){
for (i=1; i<=num_pts; i++){
  for (j=1; j<=volume[i].n_pts; j++){
    if (volume[i].viz1[j] <= num_pts){
      /*retorna sinal(coefic[i].F[j]);
      coefic[i].auxu[j] = param[i].Rho[j] * ((pos * volume[i].u) + (neg * volume[volume[i].viz1[j]].u))
* param[i].ex[j] * param[i].S[j];
      coefic[i].auxv[j] = param[i].Rho[j] * ((pos * volume[i].v) + (neg * volume[volume[i].viz1[j]].v))
* param[i].ey[j] * param[i].S[j];*/
      coefic[i].auxu[j] = param[i].Rho[j] * (0.5 * (volume[i].u + volume[volume[i].viz1[j]].u))
* param[i].ex[j] * param[i].S[j];
      coefic[i].auxv[j] = param[i].Rho[j] * (0.5 * (volume[i].v + volume[volume[i].viz1[j]].v))
* param[i].ey[j] * param[i].S[j];
    }
    else{
      coefic[i].auxu[j] = 0.0;
      coefic[i].auxv[j] = 0.0;
    }
  }
}
}

```

```

    }
  }
}

// DEFINICAO DA FUNCAO QUE CALCULA O COEFICIENTE DE LIGACAO ENTRE i E j
void calcula_AU(tipo_volume volume[], tipo_coefic coefic[]){
for (i=1; i<=num_pts; i++){
  for (j=1; j<=volume[i].n_pts; j++){
    retorna_sinal(coefic[i].F[j]);
    coefic[i].AU[j] = - ( coefic[i].auxu[j] + coefic[i].F[j] ) * neg + coefic[i].D[j];
  }
}
}

// DEFINICAO DA FUNCAO QUE CALCULA O COEFICIENTE DE LIGACAO ENTRE i E j
void calcula_AV(tipo_volume volume[], tipo_coefic coefic[]){
for (i=1; i<=num_pts; i++){
  for (j=1; j<=volume[i].n_pts; j++){
    retorna_sinal(coefic[i].F[j]);
    coefic[i].AV[j] = - ( coefic[i].auxv[j] + coefic[i].F[j] ) * neg + coefic[i].D[j];
  }
}
}

// DEFINICAO DA FUNCAO QUE CALCULA O COEFICIENTE CENTRAL
void calcula_Apu(tipo_coefic coefic[]){
for (i=1; i <= num_pts; i++){
  Soma_A[i] = 0.0;
  for (j=1; j <= volume[i].n_pts; j++){
    Soma_A[i] += coefic[i].AU[j] + coefic[i].auxu[j];
  }
}
for (i=1; i <= num_pts; i++){
  coefic[i].Apu = (Soma_A[i] + coefic[i].A0) / ALFA;
}
}

// DEFINICAO DA FUNCAO QUE CALCULA O COEFICIENTE CENTRAL
void calcula_Apv(tipo_coefic coefic[]){
for (i=1; i <= num_pts; i++){
  Soma_A[i] = 0.0;
  for (j=1; j <= volume[i].n_pts; j++){
    Soma_A[i] += coefic[i].AV[j] + coefic[i].auxv[j];
  }
}
for (i=1; i <= num_pts; i++){
  coefic[i].Apv = (Soma_A[i] + coefic[i].A0) / ALFA;
}
}

// DEFINICAO DA FUNCAO QUE CALCULA O TERMO FONTE PARA A VELOCIDADE u
void calcula_bu(tipo_volume volume[], tipo_coefic coefic[]){
for (i=1; i<=num_pts; i++){
  Soma_A[i] = 0.0;
  for (j = 1; j <= volume[i].n_pts; j++){
    /*retorna_sinal(coefic[i].F[j]);
    Soma_A[i] += coefic[i].auxu[j] * ((pos * volume[i].u) + (neg * volume[volume[i].viz1[j]].u));*/
    Soma_A[i] += coefic[i].auxu[j] * (0.5 * (volume[i].u + volume[volume[i].viz1[j]].u));
  }
}
}

```

```

}

for (i=1; i<=num_pts; i++){
    coefic[i].bu = (S[i] * volume[i].vol_diag) + Soma_A[i] + (volume[i].u_old * coefic[i].A0)
                + (coefic[i].Apu * (1 - ALFA) * volume[i].u);
}
}

// DEFINICAO DA FUNCAO QUE CALCULA O TERMO FONTE PARA A VELOCIDADE v
void calcula_bv(tipo_volume volume[], tipo_coefic coefic[]){
for (i=1; i<=num_pts; i++){
    Soma_A[i] = 0.0;
    for (j = 1; j <= volume[i].n_pts; j++){
        /*retorna_sinal(coefic[i].F[j]);
        Soma_A[i] += coefic[i].auxv[j] * ((pos * volume[i].v) + (neg * volume[volume[i].viz1[j]].v));*/
        Soma_A[i] += coefic[i].auxv[j] * (0.5 * (volume[i].v + volume[volume[i].viz1[j]].v));
    }
}
for (i=1; i<=num_pts; i++){
    coefic[i].bv = (S[i] * volume[i].vol_diag) + Soma_A[i] + (volume[i].v_old * coefic[i].A0)
                + (coefic[i].Apv * (1 - ALFA) * volume[i].v);
}
}

// DEFINICAO DA ROTINA QUE ESCREVE OS COEFICIENTES
void escreve_coeficientes(tipo_coefic coefic[]){
for (i=1; i<=num_pts; i++){
    for (j=1; j<=volume[i].n_pts; j++){
        printf("\n F[%i][%i] = %lf",i,j,coefic[i].F[j]);
        printf("\t AU[%i][%i] = %lf",i,j,coefic[i].AU[j]);
        printf("\t AV[%i][%i] = %lf",i,j,coefic[i].AV[j]);
        printf("\t Apu[%i] = %lf",i,coefic[i].Apu);
        printf("\t Apv[%i] = %lf",i,coefic[i].Apv);
        printf("\t bu[%i] = %lf",i,coefic[i].bu);
        printf("\t bv[%i] = %lf\n",i,coefic[i].bv);
    }
}
}

// DEFINICAO DA FUNCAO QUE CALCULA OS COEFICIENTES
void calcula_coeficientes(){
// Rotina para calculo do Fij
calcula_F(volume, param, coefic);

// Rotina para calculo dos coefic_a
calcula_coefic_a(volume, param, coefic);

// Rotina para calculo do Aij
calcula_AU(volume, coefic);

// Rotina para calculo do Aij
calcula_AV(volume, coefic);

// Rotina para calculo do Api
calcula_Apu(coefic);

// Rotina para calculo do Api
calcula_Apv(coefic);

```



```

// Rotina para calculo do bui
calcula_bu(volume, coefic);

// Rotina para calculo do bui
calcula_bv(volume, coefic);

// Rotina escreve coeficientes
//escreve_coeficientes(coefic);
}
// DEFINICAO DA FUNCAO QUE CALCULA OS SOMATORIOS PARA CALCULO DO GRADIENTE NA
// DIRECAO x E y PELO METODO DE TANIGUCHI, LEVANDO EM CONTA A PRESSAO
void calcula_soma_Tan_pressao(){
for (i=1; i<=num_pts; i++){
    Soma_3[i] = 0.0;
    Soma_5[i] = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts){
            Soma_3[i] += (g[i][j] * param[i].ex[j] * (volume[volume[i].viz1[j]].p - volume[i].p)
                / param[i].L[j]);
            Soma_5[i] += (g[i][j] * param[i].ey[j] * (volume[volume[i].viz1[j]].p - volume[i].p)
                / param[i].L[j]);
        }
    }
}
}

// DEFINICOES DAS FUNCOES QUE CALCULAM O GRADIENTE DE PRESSAO NA DIRECAO x

// Metodo das medias ponderadas dos gradientes existentes nas interfaces
void calcula_grad_pressao_x_Mal(tipo_volume volume[], tipo_param param[]){
double Soma_Grad_x[N_PONTOS];
for (i=1; i<=num_pts; i++){
    Soma_Grad_x[i] = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts){
            Soma_Grad_x[i] += ((volume[volume[i].viz1[j]].p - volume[i].p) * param[i].ex[j]);
        }
    }
}
for (i=1; i<=num_pts; i++){
    volume[i].Grad_Px = Soma_Grad_x[i]/param[i].Soma_L;
}
}

// Metodo das medias ponderadas dos gradientes existentes nas interfaces modificado, Cardoso
void calcula_grad_pressao_x_Cardoso(tipo_volume volume[], tipo_param param[]){
double Soma_Grad_x[N_PONTOS];
for (i=1; i<=num_pts; i++){
    Soma_Grad_x[i] = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts){
            Soma_Grad_x[i] += ((volume[volume[i].viz1[j]].p - volume[i].p) * param[i].ex[j]);
        }
    }
}
for (i=1; i<=num_pts; i++){
    volume[i].Grad_Px = Soma_Grad_x[i]/param[i].Soma_Lx;
}
}

```

```

// Metodo Jameson e Mavriplis (AIAA Journal, Vol. 24, pp. 611-18, 1986)
void calcula_grad_pressao_x_JeM(tipo_volume volume[], tipo_param param[]){
for (i=1; i<=num_pts; i++){
    volume[i].Grad_Px = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts)
            volume[i].Grad_Px += (((volume[volume[i].viz1[j]].p * param[i].S[j] * param[i].ex[j]) *
                                0.5) / volume[i].vol_diag);
    }
}
}

// Metodo Taniguchi et alli (...)
void calcula_grad_pressao_x_Tan(tipo_volume volume[]){
for (i=1; i<=num_pts; i++){
    volume[i].Grad_Px = ((Soma_3[i] * Soma_4[i]) - (Soma_2[i] * Soma_5[i]))/((Soma_1[i] * Soma_4[i]) -
    (Soma_2[i] * Soma_2[i]));
}
}

// Metodo Cardoso modificado
void calcula_grad_pressao_x_Dissertacao(tipo_volume volume[], tipo_param param[]){
double Soma_Grad_x[N_PONTOS];
for(i=1; i<=num_pts; i++){
    Soma_Grad_x[i] = 0.0;
    for(j=1; j<=volume[i].n_pts; j++){
        if(volume[i].viz1[j] <= num_pts){
            Soma_Grad_x[i] += (((volume[volume[i].viz1[j]].p - volume[i].p)/param[i].L[j]) * param[i].ex[j] *
                                g[i][j]);
        }
    }
}
for (i=1; i<=num_pts; i++){
    for (j=1; j<=volume[i].n_pts; j++){
        g[i][j] = param[i].S[j]/param[i].L[j];
    }
}
for(i=1; i<=num_pts; i++){
    param[i].Soma_gx = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts)
            param[i].Soma_gx += (g[i][j] * fabs(param[i].ex[j]));
    }
}
for (i=1; i<=num_pts; i++){
    volume[i].Grad_Px = Soma_Grad_x[i]/param[i].Soma_gx;
}
}

// DEFINICAO DA FUNCAO QUE CALCULA A NORMA EUCLIDIANA DOS RESIDUOS DE U E V
(SEPARADAMENTE)
double calcula_norma_u(tipo_volume volume[], tipo_coefic coefic[]){
norma = 0.0;
for(i=1; i<=num_pts; i++){
    aux[i] = 0.0;
    for(j=1; j<=volume[i].n_pts; j++){
        aux[i] += (coefic[i].AU[j] * volume[volume[i].viz1[j]].u);
    }
    r[i] = (coefic[i].bu - (volume[i].vol_diag * volume[i].Grad_Px)) - ((coefic[i].Apu * volume[i].u) - aux[i]);
    p[i] = 0.0;
}
}

```

```

    norma += (r[i] * r[i]);
}
norma = sqrt((double)norma);
return norma;
}

// DEFINICAO DA FUNCAO QUE CALCULA AS VELOCIDADES u
void calcula_velocidade_u() {
// Rotina para calculo do gradiente de pressao
switch (grad_type) {
    case 1: {
        calcula_grad_pressao_x_Mal(volume, param);
    } break;
    case 2: {
        calcula_grad_pressao_x_Cardoso(volume, param);
    } break;
    case 3: {
        calcula_grad_pressao_x_JeM(volume, param);
    } break;
    case 4: {
        calcula_soma_Tan_pressao();
        calcula_grad_pressao_x_Tan(volume);
    } break;
    case 5: {
        calcula_grad_pressao_x_Dissertacao(volume, param);
    } break;
}
// Rotina para resolucao do sistema de equacoes para u

// CALCULA ATUALIZACAO DAS VARIAVEIS SEGUNDO A REGRA DE ARMIJO P/ U
s = 1.0;
for(i=1; i<=num_pts; i++){
    volume[i].uva = volume[i].u;
}
calcula_norma_u(volume,coefic);
norm = norma;
norma0 = (1 - 0.5 * s) * norm;
CG_u(volume);
calcula_norma_u(volume,coefic);
norma1 = norma;
for(i=1; i<=num_pts; i++){
    volume[i].z = volume[i].u - volume[i].uva;
}
do {
    s = s * 0.5;
    for(i=1; i<=num_pts; i++){
        volume[i].u = volume[i].uva + (s * volume[i].z);
    }
    calcula_norma_u(volume,coefic);
    norma1 = norma;
    norma0 = (1 - 0.5 * s) * norm;
    su = s;
}
while (norma1 >= norma0);
}

// DEFINICAO DA FUNCAO QUE CALCULA O GRADIENTE DE PRESSAO NA DIRECAO y
// Metodo das medias ponderadas dos gradientes existentes nas interfaces
void calcula_grad_pressao_y_Mal(tipo_volume volume[], tipo_param param[]){
double Soma_Grad_y[N_PONTOS];

```

```

for (i=1; i<=num_pts; i++){
  Soma_Grad_y[i] = 0.0;
  for (j=1; j<=volume[i].n_pts; j++){
    if (volume[i].viz1[j] <= num_pts)
      Soma_Grad_y[i] += ((volume[volume[i].viz1[j]].p - volume[i].p) * param[i].ey[j]);
  }
}
for (i=1; i<=num_pts; i++){
  volume[i].Grad_Py = Soma_Grad_y[i]/param[i].Soma_L;
}
}

// Metodo das medias ponderadas dos gradientes existentes nas interfaces modificado, Cardoso
void calcula_grad_pressao_y_Cardoso(tipo_volume volume[], tipo_param param[]){
double Soma_Grad_y[N_PONTOS];
for (i=1; i<=num_pts; i++){
  Soma_Grad_y[i] = 0.0;
  for (j=1; j<=volume[i].n_pts; j++){
    if (volume[i].viz1[j] <= num_pts)
      Soma_Grad_y[i] += ((volume[volume[i].viz1[j]].p - volume[i].p) * param[i].ey[j]);
  }
}
for (i=1; i<=num_pts; i++){
  volume[i].Grad_Py = Soma_Grad_y[i]/param[i].Soma_Ly;
}
}

// Metodo Jameson e Mavriplis (AIAA Journal, Vol. 24, pp. 611-18, 1986)
void calcula_grad_pressao_y_JeM(tipo_volume volume[], tipo_param param[]){
for (i=1; i<=num_pts; i++){
  volume[i].Grad_Py = 0.0;
  for (j=1; j<=volume[i].n_pts; j++){
    if (volume[i].viz1[j] <= num_pts)
      volume[i].Grad_Py += ((volume[volume[i].viz1[j]].p * param[i].S[j] * param[i].ey[j] *
        0.5)/volume[i].vol_diag);
  }
}
}

// Metodo Taniguchi et alli (...)
void calcula_grad_pressao_y_Tan(tipo_volume volume[]){
for (i=1; i<=num_pts; i++){
  volume[i].Grad_Py = ((Soma_1[i] * Soma_5[i]) - (Soma_2[i] * Soma_3[i]))/((Soma_1[i] * Soma_4[i])
    - (Soma_2[i] * Soma_2[i]));
}
}

// Metodo Cardoso modificado
void calcula_grad_pressao_y_Dissertacao(tipo_volume volume[], tipo_param param[]){
double Soma_Grad_y[N_PONTOS];
for(i=1; i<=num_pts; i++){
  Soma_Grad_y[i] = 0.0;
  for(j=1; j<=volume[i].n_pts; j++){
    if(volume[i].viz1[j] <= num_pts)
      Soma_Grad_y[i] += (((volume[volume[i].viz1[j]].p - volume[i].p)/param[i].L[j]) * param[i].ey[j] *
        g[i][j]);
  }
}
for (i=1; i<=num_pts; i++){
  for (j=1; j<=volume[i].n_pts; j++){

```

```

        g[i][j] = param[i].S[j]/param[i].L[j];
    }
}
for(i=1; i<=num_pts; i++){
    param[i].Soma_gy = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts)
            param[i].Soma_gy += (g[i][j] * fabs(param[i].ey[j]));
    }
}
for (i=1; i<=num_pts; i++){
    volume[i].Grad_Py = Soma_Grad_y[i]/param[i].Soma_gy;
}
}

// DEFINICAO DA FUNCAO QUE CALCULA A NORMA EUCLIDIANA DOS RESIDUOS DE U E V
(SEPARADAMENTE)
double calcula_norma_v(tipo_volume volume[], tipo_coefic coefic[]){
    norma = 0.0;
    for(i=1; i<=num_pts; i++){
        aux[i] = 0.0;
        for(j=1; j<=volume[i].n_pts; j++){
            aux[i] += (coefic[i].AV[j] * volume[volume[i].viz1[j]].v);
        }
        r[i] = (coefic[i].bv - (volume[i].vol_diag * volume[i].Grad_Py)) - ((coefic[i].Apv * volume[i].v) - aux[i]);
        p[i] = 0.0;
        norma += (r[i] * r[i]);
    }
    norma = sqrt((double)norma);
    return norma;
}

// DEFINICAO DA FUNCAO QUE CALCULA A VELOCIDADE v
void calcula_velocidade_v(){
// Rotina para calculo do gradiente de pressao
switch (grad_type){
    case 1:{
        calcula_grad_pressao_y_Mal(volume, param);
    }break;
    case 2:{
        calcula_grad_pressao_y_Cardoso(volume, param);
    }break;
    case 3:{
        calcula_grad_pressao_y_JeM(volume, param);
    }break;
    case 4:{
        calcula_grad_pressao_y_Tan(volume);
    }break;
    case 5:{
        calcula_grad_pressao_y_Dissertacao(volume, param);
    }break;
}
// Rotina para resolucao do sistema de equacoes para v
// CALCULA ATUALIZACAO DAS VARIAVEIS SEGUNDO A REGRA DE ARMIJO P/ V
s = 1.0;
for(i=1; i<=num_pts; i++){
    volume[i].uva = volume[i].v;
}
calcula_norma_v(volume,coefic);
norm = norma;

```

```

norma0 = (1 - 0.5 * s) * norm;
CG_v(volume);
calcula_norma_v(volume,coefic);
norma1 = norma;
for(i=1; i<=num_pts; i++){
    volume[i].z = volume[i].v - volume[i].uva;
}
do {
    s = s * 0.5;
    for(i=1; i<=num_pts; i++){
        volume[i].v = volume[i].uva + (s * volume[i].z);
    }
    calcula_norma_v(volume,coefic);
    norma1 = norma;
    norma0 = (1 - 0.5 * s) * norm;
    sv = s;
}
while (norma1 >= norma0);
}

// DEFINICAO DA FUNCAO QUE ZERA p LINHA
void zera_p_lin(){
for (i=1; i<=num_pts; i++){
    volume[i].p_lin = 0.0;
}
}

// DEFINICAO DA FUNCAO QUE CALCULA DW
void calcula_dw(tipo_volume volume[], tipo_coefic coefic[]){
for(i=1; i<=num_pts; i++){
    for(j=1; j<=volume[i].n_pts; j++){
        if(volume[i].viz1[j] <= num_pts){
            // calcula do dw para o acoplamento simple
            volume[i].dw[j] = 0.25 * ( ( volume[i].vol_diag * ( (1 / coefic[i].Apu) + (1 /coefic[i].Apv) ) ) +
                (volume[volume[i].viz1[j]].vol_diag * ( (1 / coefic[volume[i].viz1[j]].Apu) +
                    (1 / coefic[volume[i].viz1[j]].Apv) ) ) );
        }
        else{
            volume[i].dw[j] = 0.0;
        }
    }
}
}

// DEFINICAO DA FUNCAO QUE CALCULA AS INFLUENCIAS DOS VIZINHOS
void calcula_a(tipo_volume volume[], tipo_param param[], tipo_coefic coefic[]){
for (i=1; i<=num_pts; i++){
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts){
            coefic[i].a[j] = (param[i].Rho[j] * param[i].S[j] * volume[i].dw[j])/param[i].L[j];
        }
    }
}
}

// DEFINICAO DA FUNCAO QUE CALCULA O COEFICIENTE CENTRAL
void calcula_ap(tipo_volume volume[], tipo_coefic coefic[]){
double Soma_a[N_PONTOS];
for (i=1; i<=num_pts; i++){
    Soma_a[i] = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts){

```

```

        Soma_a[i] += coefic[i].a[j];
    }
}
coefic[i].ap = Soma_a[i];
}
}

// DEFINICAO DA ROTINA QUE ESCREVE OS COEFICIENTES
void escreve_dw_e_W(tipo_volume volume[], tipo_param param[]){
for (i=1; i<=num_pts; i++){
    for (j=1; j<=volume[i].n_pts; j++){
        printf("\n dw[%i][%i] = %lf \t",i,j,volume[i].dw[j]);
        printf("W[%i][%i] = %lf \n",i,j,param[i].W[j]);
    }
}
}

// DEFINICAO DA FUNCAO QUE CALCULA A PRESSAO p
void calcula_p_lin(){
// Rotina para calculo do dw
calcula_dw(volume, coefic);

// Rotina para calculo das influencias dos vizinhos
calcula_a(volume, param, coefic);

// Rotina para calculo do gradiente de pressao
calcula_ap(volume, coefic);

// Rotina para escrever dw
// escreve_dw_e_W(volume,param);

// Rotina para resolucao do sistema de equacoes para p_lin
CG_p_lin(volume, coefic);
}

// DEFINICAO DA FUNCAO QUE CALCULA A PRESSAO JA' CORRIGIDA
void calcula_p_corrigido(tipo_volume volume[]){
double pref=0;
for (i=1; i<=num_pts; i++){
    volume[i].p += (ALFA_p * volume[i].p_lin);
}
pref = volume[1].p;
for (i=1; i<=num_pts; i++){
    volume[i].p -= pref;
}
}

// DEFINICAO DA FUNCAO QUE CALCULA A VELOCIDADE CORRIGIDA NAS FACES DOS
VOLUMES DE CONTROLE
void calcula_W_corrigido(tipo_volume volume[], tipo_param param[]){
double Grad_PW_lin[N_PONTOS][MAX_VIZ];
double W_lin[N_PONTOS][MAX_VIZ];
for (i=1; i<=num_pts; i++)
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts){
            Grad_PW_lin[i][j] = (volume[volume[i].viz1[j]].p_lin - volume[i].p_lin)/param[i].L[j];
            W_lin[i][j] = - volume[i].dw[j] * Grad_PW_lin[i][j];
            param[i].W[j] += W_lin[i][j];
        }
    }
}

```

```

    }
    else{
        param[i].W[j] += 0.0;
    }
}
}

// DEFINICAO DA FUNCAO QUE CALCULA OS SOMATORIOS PARA CALCULO DO GRADIENTE NA
// DIRECAO x E y PELO METODO DE TANIGUCHI, LEVANDO EM CONTA A PRESSAO LINHA
void calcula_soma_Tan_pressao_lin(){
for (i=1; i<=num_pts; i++){
    Soma_3[i] = 0.0;
    Soma_5[i] = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts){
            Soma_3[i] += (g[i][j] * param[i].ex[j] * (volume[volume[i].viz1[j]].p_lin -
                volume[i].p_lin)/param[i].L[j]);
            Soma_5[i] += (g[i][j] * param[i].ey[j] * (volume[volume[i].viz1[j]].p_lin -
                volume[i].p_lin)/param[i].L[j]);
        }
    }
}
}

// DEFINICAO DA FUNCAO QUE CALCULA O GRADIENTE DE PRESSAOLINHA NA DIRECAO x
// Metodo das medias ponderadas dos gradientes existentes nas interfaces
void calcula_grad_px_lin_Mal(tipo_volume volume[], tipo_param param[]){
double Soma_Grad_x[N_PONTOS];
for (i=1; i<=num_pts; i++){
    Soma_Grad_x[i] = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts)
            Soma_Grad_x[i] += ((volume[volume[i].viz1[j]].p_lin - volume[i].p_lin) * param[i].ex[j]);
    }
}
for (i=1; i<=num_pts; i++){
    volume[i].Grad_px_lin = Soma_Grad_x[i]/param[i].Soma_L;
}
}

// Metodo das medias ponderadas dos gradientes existentes nas interfaces modificado, Cardoso
void calcula_grad_px_lin_Cardoso(tipo_volume volume[], tipo_param param[]){
double Soma_Grad_x[N_PONTOS];
for (i=1; i<=num_pts; i++){
    Soma_Grad_x[i] = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts)
            Soma_Grad_x[i] += ((volume[volume[i].viz1[j]].p_lin - volume[i].p_lin) * param[i].ex[j]);
    }
}
for (i=1; i<=num_pts; i++){
    volume[i].Grad_px_lin = Soma_Grad_x[i]/param[i].Soma_Lx;
}
}

// Metodo Jameson e Mavriplis (AIAA Journal, Vol. 24, pp. 611-18, 1986)
void calcula_grad_px_lin_JeM(tipo_volume volume[], tipo_param param[]){
for (i=1; i<=num_pts; i++){
    volume[i].Grad_px_lin = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){

```



```

        if (volume[i].viz1[j] <= num_pts)
            volume[i].Grad_px_lin = volume[i].Grad_px_lin + (((volume[volume[i].viz1[j]].p_lin
                * param[i].S[j] * param[i].cy[j])/volume[i].vol_diag * 0.5);
        }
    }
}

// Metodo Taniguchi et alli (...)
void calcula_grad_px_lin_Tan(tipo_volume volume[]){
for (i=1; i<=num_pts; i++){
    volume[i].Grad_px_lin = ((Soma_3[i] * Soma_4[i]) - (Soma_2[i] * Soma_5[i]))/((Soma_1[i] * Soma_4[i]) -
        (Soma_2[i] * Soma_2[i]));
}
}

// Metodo Cardoso modificado
void calcula_grad_px_lin_Dissertacao(tipo_volume volume[], tipo_param param[]){
double Soma_Grad_x[N_PONTOS];
for(i=1; i<=num_pts; i++){
    Soma_Grad_x[i] = 0.0;
    for(j=1; j<=volume[i].n_pts; j++){
        if(volume[i].viz1[j] <= num_pts){
            Soma_Grad_x[i] += (((volume[volume[i].viz1[j]].p_lin - volume[i].p_lin)/param[i].L[j]) *
                param[i].ex[j] * g[i][j]);
        }
    }
}
for (i=1; i<=num_pts; i++){
    for (j=1; j<=volume[i].n_pts; j++){
        g[i][j] = param[i].S[j]/param[i].L[j];
    }
}
for(i=1; i<=num_pts; i++){
    param[i].Soma_gx = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts){
            param[i].Soma_gx += (g[i][j] * fabs(param[i].ex[j]));
        }
    }
}
for (i=1; i<=num_pts; i++){
    volume[i].Grad_px_lin = Soma_Grad_x[i]/param[i].Soma_gx;
}
}

// DEFINICAO DA FUNCAO QUE CALCULA A VELOCIDADE U CORRIGIDA
void calcula_u_corrigido(){
double u_lin[N_PONTOS];
switch (grad_type){
    case 1:{
        calcula_grad_px_lin_Mal(volume, param);
    }break;
    case 2:{
        calcula_grad_px_lin_Cardoso(volume, param);
    }break;
    case 3:{
        calcula_grad_px_lin_JeM(volume, param);
    }break;
    case 4:{
        calcula_soma_Tan_pressao_lin();
    }
}
}

```

```

        calcula_grad_px_lin_Tan(volume);
    }break;
    case 5:{
        calcula_grad_px_lin_Dissertacao(volume, param);
    }break;
}
for(i=1; i<=num_pts; i++){
    u_lin[i] = (- volume[i].vol_diag * volume[i].Grad_px_lin)/coefic[i].Apu;
}
for(i=1; i<=num_pts; i++)
    volume[i].u += u_lin[i];
}

// DEFINICAO DA FUNCAO QUE CALCULA O GRADIENTE DE PRESSAO LINHA NA DIRECAO y
// Metodo das medias ponderadas dos gradientes existentes nas interfaces
void calcula_grad_py_lin_Mal(tipo_volume volume[], tipo_param param[]){
double Soma_Grad_y[N_PONTOS];
for(i=1; i<=num_pts; i++){
    Soma_Grad_y[i] = 0.0;
    for(j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts)
            Soma_Grad_y[i] += ((volume[volume[i].viz1[j]].p_lin - volume[i].p_lin) * param[i].ey[j]);
    }
}
for (i=1; i<=num_pts; i++){
    volume[i].Grad_py_lin = Soma_Grad_y[i]/param[i].Soma_L;
}
}

// Metodo das medias ponderadas dos gradientes existentes nas interfaces modificado, Cardoso
void calcula_grad_py_lin_Cardoso(tipo_volume volume[], tipo_param param[]){
double Soma_Grad_y[N_PONTOS];
for (i=1; i<=num_pts; i++){
    Soma_Grad_y[i] = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts)
            Soma_Grad_y[i] += ((volume[volume[i].viz1[j]].p_lin - volume[i].p_lin) * param[i].ey[j]);
    }
}
for (i=1; i<=num_pts; i++){
    volume[i].Grad_py_lin = Soma_Grad_y[i]/param[i].Soma_Ly;
}
}

// Metodo Jameson e Mavriplis (AIAA Journal, Vol. 24, pp. 611-18, 1986)
void calcula_grad_py_lin_JeM(tipo_volume volume[], tipo_param param[]){
for (i=1; i<=num_pts; i++){
    volume[i].Grad_py_lin = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        if (volume[i].viz1[j] <= num_pts)
            volume[i].Grad_py_lin += ((volume[volume[i].viz1[j]].p_lin * 0.5 * param[i].S[j] *
            param[i].ey[j])/volume[i].vol_diag);
    }
}
}

// Metodo Taniguchi et alli (...)
void calcula_grad_py_lin_Tan(tipo_volume volume[]){
for (i=1; i<=num_pts; i++){
    volume[i].Grad_py_lin = ((Soma_1[i] * Soma_5[i]) - (Soma_2[i] * Soma_3[i]))/((Soma_1[i] * Soma_4[i]) -

```

```

        (Soma_2[i] * Soma_2[i]);
    }
}

// Metodo Cardoso modificado
void calcula_grad_py_lin_Dissertacao(tipo_volume volume[], tipo_param param[]){
double Soma_Grad_y[N_PONTOS];
for(i=1; i<=num_pts; i++){
    Soma_Grad_y[i] = 0.0;
    for(j=1; j<=volume[i].n_pts; j++){
        if(volume[i].viz1[j] <= num_pts)
            Soma_Grad_y[i] += (((volume[volume[i].viz1[j]].p_lin - volume[i].p_lin)/ param[i].L[j]) *
                param[i].ey[j] * g[i][j]);
    }
}
for (i=1; i<=num_pts; i++){
    for (j=1; j<=volume[i].n_pts; j++){
        g[i][j] = param[i].S[j]/param[i].L[j];
    }
}
for(i=1; i<=num_pts; i++){
    param[i].Soma_gy = 0.0;
    for(j=1; j<=volume[i].n_pts; j++){
        if(volume[i].viz1[j] <= num_pts)
            param[i].Soma_gy += (g[i][j] * fabs(param[i].ey[j]));
    }
}

for (i=1; i<=num_pts; i++){
    volume[i].Grad_py_lin = Soma_Grad_y[i]/param[i].Soma_gy;
}
}

// DEFINICAO DA FUNCAO QUE CALCULA A VELOCIDADE U CORRIGIDA
void calcula_v_corrigido(){
double v_lin[N_PONTOS];
switch(grad_type){
    case 1:{
        calcula_grad_py_lin_Mal(volume, param);
    }break;
    case 2:{
        calcula_grad_py_lin_Cardoso(volume, param);
    }break;
    case 3:{
        calcula_grad_py_lin_JeM(volume, param);
    }break;
    case 4:{
        calcula_grad_py_lin_Tan(volume);
    }break;
    case 5:{
        calcula_grad_py_lin_Dissertacao(volume, param);
    }break;
}
for(i=1; i<=num_pts; i++)
    v_lin[i] = (- volume[i].vol_diag * volume[i].Grad_py_lin)/coefic[i].Apv;
for(i=1; i<=num_pts; i++){
    volume[i].v += v_lin[i];
}
}
}

```

```

// DEFINICAO DA FUNCAO QUE CALCULA O CRITERIO DE PARADA
double calcula_criterio_parada(){
double erro[N_PONTOS];
double Erro = 0.0;
calcula_F(volume, param, coefic);
for (i=1; i<=num_pts; i++){
    erro[i] = 0.0;
    for (j=1; j<=volume[i].n_pts; j++){
        erro[i] += coefic[i].F[j];
    }
    if (fabs(erro[i]) > Erro)
        Erro = fabs(erro[i]);
}
return Erro;
}

// DEFINICAO DA FUNCAO QUE IMPRIME OS RESULTADOS FINAIS
void imprime_result(tipo_volume volume[]){
int l, k1, k2;
double t1, t2;
printf ("\ntipo do gradiente: %d", grad_type);
printf ("\np[68] = %lf", volume[68].p);*/
for (i=0; i<=79; i++){
    k1 = 40 + (80 * i);
    k2 = 41 + (80 * i);
    t1 = (volume[k1].cy/32.10);
    t2 = (volume[k2].cy/32.10);
    printf("\n %lf \t %i \t %lf \t %lf \t %i \t %lf", t1, k1, volume[k1].u, t2, k2, volume[k2].u);
}
for(l=3201; l<=3280; l++){
    t2 = (volume[l].cx/32.10);
    printf("\n %lf \t %i \t %lf", t2,l,volume[l].v);
}
}

// DEFINICAO DA FUNCAO PRINCIPAL
void main(){
int tempo;
double Erro;
num_iter = 0;
for (i=1; i<=num_pts; i++){
    for (j=1; j<=volume[i].n_pts; j++){
        coefic[i].auxu[j] = 0.0;
        coefic[i].auxv[j] = 0.0;
    }
}
// Rotinas para ler a malha
le_malha();

// Rotina para atribuir os parametros iniciais
atribui_param_iniciais(volume, param);

// Rotina para calculo da norma do vetor normal de cada face dos volumes de controle
calcula_norma(volume, param);

// Rotina para calculo dos parametros geometricos
calcula_param_geom(volume, param);

// Rotina para calculo do Rhoij
calcula_Rho(volume, param);

```

```
// Rotina para calculo do Gamaij
calcula_Gama(volume, param);

// Rotina para calculo do A0i
calcula_A0(volume, coefic, param);

// Rotina para calculo do Dijpara Gamaij constante
calcula_D(volume, param, coefic);

// Rotina para calculo do Wij
calcula_W(volume, param,coefic);

if (grad_type == 4)
    calcula_soma_Tan();

struct timeval time1, time2;
gettimeofday(&time1,NULL);
for (tempo=1; tempo<=1; tempo++){
    for (i=1; i<=num_pts; i++){
        volume[i].u_old = volume[i].u;
        volume[i].v_old = volume[i].v;
        volume[i].p_old = volume[i].p;
    }
}
do{
    // Rotinas para calculo dos coeficientes
    calcula_coeficientes();

    // Rotina para calculo da velocidade u
    calcula_velocidade_u();

    // Rotina para calculo da velocidade v
    calcula_velocidade_v();

    // Rotina para calculo do Wij
    if (num_iter <= 800){
        calcula_W(volume, param,coefic);
    }
    else{
        calcula_W2(volume, param,coefic);
    }

    // Rotina para zerar p linha
    zera_p_lin();

    // Rotina para calculo da correcao de pressao p linha
    calcula_p_lin();

    // Rotina para calculo da pressao corrigida
    calcula_p_corrigido(volume);

    // Rotina para calculo do criterio de parada
    Erro = calcula_criterio_parada();

    // Rotina para correcao da velocidade nas faces dos volumes de controle para teste
    // da conservacao da massa
    calcula_W_corrigido(volume, param);

    // Rotina para corrigir a velocidade u
    calcula_u_corrigido();
```

```
// Rotina para corrigir a velocidade v
calcula_v_corrigido();

num_iter++;

printf ("\n volume[35].u = %lf\t Erro = %.21lf\n", volume[35].u, Erro);
}

while ((num_iter <= 5000) && (Erro >= ERRO));
printf ("\n %i ", num_iter);

gettimeofday(&time2,NULL);
printf("\nTempo Inicial: Seg=%ld Micro-Segundos=%ld",time1.tv_sec,time1.tv_usec);
printf("\nTempo Final : Seg=%ld Micro-Segundos=%ld",time2.tv_sec,time2.tv_usec);
}
```