Universidade do Estado do Rio de Janeiro

Faculdade de Engenharia Mecânica

Virginia Silva da Costa

Técnicas Computacionais para Resolução de Sistemas Lineares aplicadas a Problemas de Enchimento de Reservatórios

Rio de Janeiro
2008

Virginia Silva da Costa

Técnicas Computacionais para Resolução de Sistemas Lineares aplicadas a Problemas de Enchimento de Reservatórios

Dissertação apresentada como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Mecânica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Fenômenos de Transporte.

Orientador: Prof. Dr. Luiz Mariano Paes de Carvalho Filho
Co-orientador: Prof. Dr. Norberto Mangiavacchi

Rio de Janeiro
2008

CATALOGAÇÃO NA FONTE
UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

C837 Costa, Virginia S.
Técnicas Computacionais para resolução de sistemas lineares aplicadas a problemas de enchimento de reservatórios / Virginia Silva da Costa. – 2008.
106 f.: il.

Orientador: Luiz Mariano Paes de Carvalho Filho.
Co-orientador: Norberto Mangiavacchi.
Dissertação (Mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia Mecânica.
Bibliografia: f. 96-99.

1. Reservatórios – Escoamento – Simulação por computador – Teses. 2. Dinâmica dos fluidos – Simulação por computador – Teses. I. Carvalho Filho, Luiz Mariano Paes de. II. Universidade do Estado do Rio de Janeiro. Faculdade de Engenharia. III. Título.

CDU 627.03

Assinatura Data

Virginia Silva da Costa

Técnicas Computacionais para Resolução de Sistemas Lineares aplicadas a Problemas de Enchimento de Reservatórios

Dissertação apresentada como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Mecânica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Fenômenos de Transporte.

Aprovado em _____

Banca Examinadora: _____

_____
Prof. Dr. Luiz Mariano Paes de Carvalho Filho (Orientador)
Instituto de Matemática e Estatística da UERJ

_____
Prof. Dr. Norberto Mangiavacchi (Co-orientador)
Faculdade de Engenharia Mecânica da UERJ

_____
Prof. Dr. Carlos Antônio de Moura
Instituto de Matemática e Estatística da UERJ

_____
Dr. Cássio Pereira Botelho Soares
Furnas Centrais Elétricas S.A.

Rio de Janeiro
2008

AGRADECIMENTOS

Aos meus pais, Álvaro da Costa e Genilda Maria da Silva, por ter dedicado anos de suas vidas à minha educação. Ao meu querido marido Carlos Roberto da Cunha pelo apoio e a minha filha Vitória da Costa Cunha por servir de estímulo aos meus estudos.

Aos meus orientadores, Prof. Luiz Mariano Paes de Carvalho e Prof. Norberto Mangiavacchi, por terem me ajudado durante toda a pós-graduação.

Aos colaboradores do Projeto GESAR.

Aos meus amigos, Wagner Rodrigues Fortes e Valéria Saldanha Motta.

RESUMO

COSTA, Virginia S. *Técnicas Computacionais para Resolução de Sistemas Lineares aplicadas a Problemas de Enchimento de Reservatórios*. *Brasil*. 2008. 106 f. Dissertação (Mestrado em Engenharia Mecânica) – Faculdade de Engenharia Mecânica, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2008.

Nos últimos anos, o surgimento de novas tecnologias na área da computação tem possibilitado que diversos ramos da ciência desenvolvam técnicas de simulação de fenômenos físicos que, por sua vez, se utilizam de algoritmos poderosos que exigem um grande conhecimento acerca da utilização de recursos computacionais tanto de *hardware* (processamento, memória, etc) quanto de *software* (linguagens de programação, orientação a objeto, etc). Um desses ramos, em especial, é a dinâmica dos fluidos, que estuda a física de processos que envolvem escoamentos dos fluidos. Devido à variedade de aplicações relacionadas com a dinâmica de fluidos computacional, muitas técnicas diferentes para se aproximar os fenômenos físicos foram desenvolvidas. Estas técnicas, na sua grande maioria, buscam a representação discreta destes processos, tanto na representação geométrica do problema -discretização por malhas estruturadas, por malhas não estruturadas e sem utilização de malha - quanto na discretização dos modelos teóricos existentes - métodos de diferenças finitas, métodos de volumes finitos e métodos de elementos finitos, entre outros. A partir da discretização de fenômenos da dinâmica de fluidos computacional, assim como de outros fenômenos físicos, obtém-se sistemas de equações cujas dimensões e cujo percentual de esparsidade variam de acordo com as técnicas acima citadas e de acordo com o domínio contínuo representado. O surgimento desses grandes sistemas de equações tem encorajado as pesquisas em torno de métodos matemáticos para sua resolução e aproximação de sua solução e, paralelamente ao desenvolvimento destes métodos, observa-se a necessidade de se desenvolver técnicas computacionais que utilizem de forma otimizada a tecnologia existente nos dias de hoje. O presente trabalho propõe uma análise de algumas dessas técnicas computacionais aplicadas na solução de Sistemas Lineares de Grande Porte oriundos da discretização de fenômenos físicos, principalmente, da dinâmica de fluidos, juntamente com a análise de alguns *métodos matemáticos de resolução destes sistemas* bastante difundidos no meio científico e de *precondicionadores* que melhoram o desempenho destes métodos.

Palavras-chave: *BLAS*, Mecânica dos Fluidos, Métodos Iterativos, Pré-condicionadores, *Trilinos*.

ABSTRACT

In the last years, the appearance of new computational technologies has allowed a variety of science branches to develop techniques for simulation of physical phenomena, which use a lot of powerful algorithms that require a great knowledge about how to use computer resources: either hardware resources, like processors and memories, or software resources, like object oriented programming languages. One of these branches, in particular, is the Fluid Dynamics. From approximations of the partial differential equations that model these phenomena, It is possible to get systems of equations where the coefficient matrix has the number of elements and the sparsity percent defined by the continuous domain discretized by some techniques like Finite Elements, for example. These large systems of equation has encouraged researches about mathematical methods to solve them (or to get a good approximation of their solutions) and, at the same time, the development of computational techniques that allowed to use efficiently the power of contemporary. This work proposes an analysis of some of these computational techniques applied in the solution of large linear systems that are from physical phenomena discretization, especially, from Fluid Dynamics. Besides, it does also propose a brief analysis of some mathematical methods for the resolution of these systems and some preconditioners that improve the performance of these methods.

Keywords: *BLAS*, Fluid Mechanics, Iterative Solvers, Preconditioners, *Trilinos*.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

INTRODUÇÃO

Motivação

Esta dissertação foi baseada nos estudos realizados nos Laboratórios do GESAR[1] (Grupo de Ensaios e Simulações em Ambientes de Reservatório) que têm como objetivo desenvolver um software capaz de simular numericamente o enchimento de grandes reservatórios de água analisando possíveis impactos ambientais e econômicos através da observação de fenômenos inerentes à Dinâmica de Fluidos.

No que se refere à reservatórios de água, se faz necessário o alagamento de regiões onde se tem uma considerável distribuição vegetal. Um exemplo pode ser visto na Figura 1, que mostra o reservatório de Manso (Mato Grosso) antes e depois do alagamento e sua vegetação submersa.

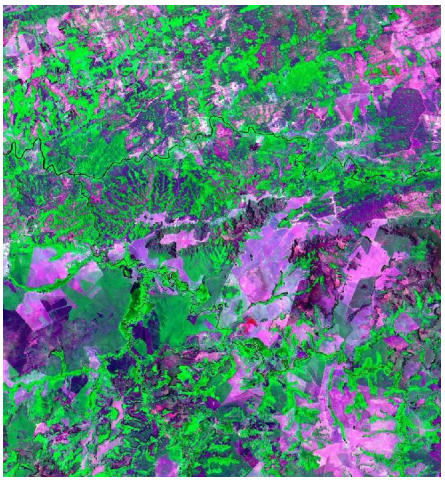
Com o alagamento de uma região de vegetação, ocorre a decomposição de matéria orgânica (solo, folhas, tronco, etc) com o decorrer do tempo, o que pode afetar a qualidade da água que deve estar própria para consumo humano. A velocidade com que esta decomposição ocorre depende de características, como *velocidade* e *pressão*, do fluido, no caso a água. Estas características podem ser calculadas a partir de equações que modelam o comportamento do fluido (*Equações de Navier-Stokes*).

No Capítulo 1, será visto um exemplo de como estas equações podem ser desenvolvidas e aproximadas, baseado no estudo de um dos colaboradores do Projeto Gesar, Fabrício Simeoni, em (SIMEONI, 2005), onde se encontra a implementação de um método numérico para simular escoamentos multifásicos em malhas dinâmicas não estruturadas. Em (SIMEONI, 2005), as Equações de Navier-Stokes são desenvolvidas em uma formulação Lagrangeana-Euleriana arbitrária e são aproximadas utilizando-se o método de elementos finitos.

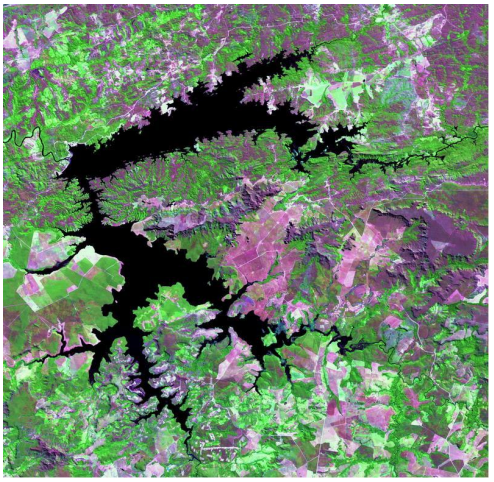
Após a obtenção dessa aproximação, tem-se sistemas lineares cuja ordem varia de acordo com o refinamento da malha gerada, o que demanda o uso de *métodos iterativos* para resolvê-los. Dependendo das condições de contorno do problema a ser simulado, esses sistemas lineares podem apresentar determinadas características que interferem no método iterativo a ser utilizado, isto é, para algumas condições de contorno, as matrizes que representam os coeficientes dos sistemas podem ser simétricas e positivas-definidas (veja definição A.2) o que permite o uso do método *gradiente conjugado pré-condicionado* (ou *PCG*, isto é, *Preconditioned Conjugate Gradient* (GOLUB; LOAN, 1996)). Já em outras situações, as matrizes **M** dos sistemas são não simétricas e não singulares, o que permite, neste caso, a aplicação de mé-

[1] Projeto Gesar, financiado por Furnas Centrais Elétricas S.A., com sede no Centro de Estudos e Pesquisa em Energia Renovável, Faculdade de Engenharia Mecânica, Universidade do Estado do Rio de Janeiro - UERJ

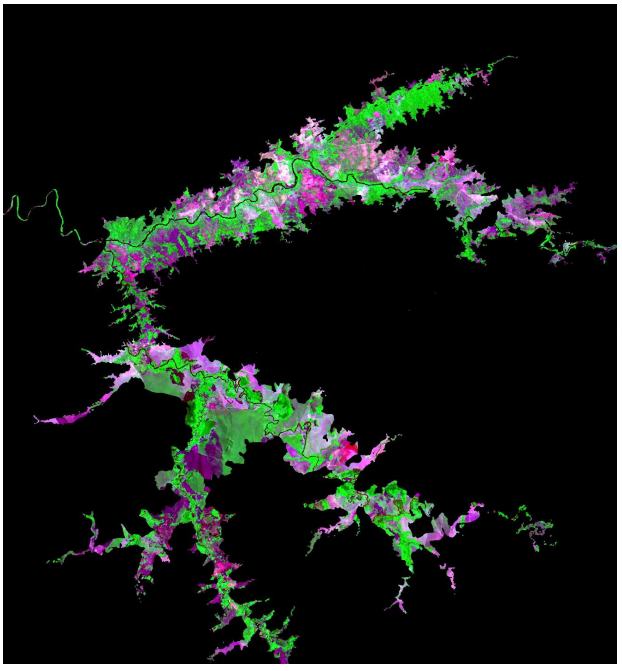
todos como *mínimo resíduo generalizado* (ou *GMRES*, isto é, *Generalized Minimum Residual Method* (SAAD; SCHULTZ, 1986)) e *gradiente biconjugado estabilizado* (ou *BI-CGSTAB*, isto é, *Bi-conjugate Gradient Stabilized* (VORST, 1992)).



(a) Distribuição vegetal antes do alagamento.



(b) Distribuição vegetal depois do alagamento (área inundada: 427 Km^2.)



(c) Vegetação Submersa.

Figura 1: Reservatório de Manso (Mato Grosso).

Neste contexto, é notória a necessidade de um estudo mais aprofundado acerca desses métodos iterativos e das técnicas utilizadas para otimizá-los, assim como, também é notória a necessidade de se ampliar conhecimentos na área da computação científica, a fim de se implementar a teoria e se obter resultados.

Sendo assim, o objetivo deste trabalho é estudar ambientes e métodos de resolução iterativa desses sistemas lineares, cuja representação matricial é dada pela equação (1) da página 8, buscando-se o isolamento e o tratamento de seus pontos críticos (multiplicação matriz-vetor e matriz-matriz), a fim de se otimiza-los do ponto de vista computacional.

Para tal, os sistemas lineares, ao longo deste texto, serão representados na sua forma matricial, a saber

$$\mathbf{Ax} = \mathbf{b}, \tag{1}$$

onde **x** é o vetor de incógnitas, **b** é o vetor dos termos independentes e **A** é a matriz dos coeficientes, esparsa e de grandes dimensões.

Algumas propriedades da matriz **A** da equação (1) na página 8, tais como número de condicionamento, dominância da diagonal, simetria, ser positiva-definida e raio espectral, são fatores de grande relevância na análise da convergência do método iterativo. Considerando esse fato, é desejável o uso de *pré-condicionadores* a fim de melhorar estas propriedades que interferem na convergência do método, transformando o problema descrito pela equação (1) em,

$$\mathbf{M}^{-1}\mathbf{Ax} = \mathbf{M}^{-1}b, \tag{2}$$

onde **M** é uma matriz que, neste caso, representa o pré-condicionador que está sendo aplicado pela esquerda. Outra possibilidade é a aplicação de **M** pela direita, assim,

$$\mathbf{AM}^{-1}\mathbf{Mx} = \mathbf{b}, \tag{3}$$

onde se tem dois sistemas a serem resolvidos, isto é,

$$\begin{aligned} \mathbf{AM}^{-1}\mathbf{u} &= \mathbf{b}, \\ \mathbf{Mx} &= \mathbf{u}, \end{aligned}$$

ou ainda, quando o pré-condicionador **M** está disponível em sua forma fatorada, como

$$\mathbf{M} = \mathbf{M}_L\mathbf{M}_R,$$

onde, na maioria das vezes, $\mathbf{M}_L$ e $\mathbf{M}_R$ são matrizes triangulares. Nesta situação, o pré-condicionador pode ser separado (*split*), assim

$$\mathbf{M}_L^{-1}\mathbf{AM}_R^{-1}\mathbf{u} = \mathbf{M}_R^{-1}\mathbf{b},\ \mathbf{x} = \mathbf{M}_R^{-1}\mathbf{u}, \tag{4}$$

onde é extremamente necessário preservar a simetria quando a matriz **A** for simétrica. A matriz $\mathbf{M}^{-1}\mathbf{A}$ da equação (2) na página 8, assim como nas demais formas de aplicação do pré-condicionados, é a matriz pré-condicionada e, agora, são suas propriedades que determinam o comportamento do método em termos de convergência. Se $\mathbf{M} = \mathbf{A}$, então a matriz pré-condicionada é a identidade, o que seria o caso ideal, pois representa a resolução exata do sistema linear. Porém, encontrar a inversa da matriz **A** em simulações de fenômenos físicos, em geral, tem um alto custo computacional. Logo, o pré-condicionador deve ser tal que a operação $\mathbf{M}^{-1}v$ para um vetor **v** qualquer seja de baixo custo em relação a operação $\mathbf{A}^{-1}\mathbf{v}$ e que seu uso em termos de melhoramento de convergência do método iterativo compense o custo de sua construção.

Os capítulos a seguir explicarão com mais detalhes a combinação de três itens importantes no que se refere a resolução de sistemas lineares de grande porte: métodos de resolução iterativa, uso de pré-condicionadores e, principalmente, técnicas computacionais para a implementação eficiente dos métodos e de seus pré-condicionadores.

Ainda, neste capítulo, serão vistas as descrições de cada capítulo da dissertação.

Organização da Dissertação

- No capítulo 1, será dado um exemplo de como se pode obter sistemas lineares a partir de estudos de fenômenos físicos;
- No capítulo 2, serão feitas alguns comentários sobre matrizes esparsas;
- No capítulo 3, serão vistos os principais métodos iterativos de resolução de sistemas lineares baseados em subespaços de Krylov;
- No capítulo 4, serão discutidas algumas técnicas de pré-condicionamento;
- No capítulo 5, serão vistas implementações dos métodos iterativos e pré-condicionadores e bibliotecas básicas de Álgebra Linear;
- No capítulo 6, serão vistos alguns resultados obtidos a partir do uso dos pacotes mencionados no capítulo 5.

1 ORIGEM DOS SISTEMAS

O objetivo deste capítulo é exemplificar, através da discretização das equações de *Navier-Stokes* descritas dentro da formulação Lagrangeana-Euleriana Arbitrária, como é possível se obter sistemas de equações no contexto da Dinâmica de Fluidos. Porém, os métodos que serão vistos neste trabalho podem ser aplicados em problemas resultantes de outras áreas de conhecimento.

Na seção 1.1, encontra-se um resumo da descrição Lagrangeana-Euleriana Arbitrária; na seção 1.2, encontram-se as equações de conservação de massa e quantidade de movimento linear; na seção 1.3, podem ser vistos conceitos básicos de elementos finitos e a discretização das equações de *Navier-Stokes*; e , na seção 1.4, será visto o método de projeção baseado em decomposição LU para desacoplar velocidade e pressão no sistema linear obtido na seção 1.3.

1.1 Descrição Lagrangeana-Eureliana Arbitrária

Para se especificar o movimento de um fluido em uma dada região do espaço, existem três principais tipos de abordagens: a descrição *Euleriana*, a descrição *Lagrangeana* e a descrição *Lagrangeana-Euleriana Arbitrária*.

Na formulação Euleriana, define-se uma região fixa no espaço, que não se deforma com relação ao tempo, onde o comportamento do fluido será estudado. Neste caso, há um fluxo de massa pelas faces desse volume de controle, e as equações para o escoamento são determinadas a partir do balanço de fluxo neste volume de controle.

Na formulação Lagrangeana, define-se uma região material formada por um conjunto de partículas de fluido, denominado de volume de controle Lagrangeano, ou ainda região material. Conforme as partículas se movimentam no escoamento, esta região se deforma, e não há fluxo de massa através de suas fronteiras. Nesta formulação, as grandezas do escoamento são especificadas como função do tempo e da partícula de fluido.

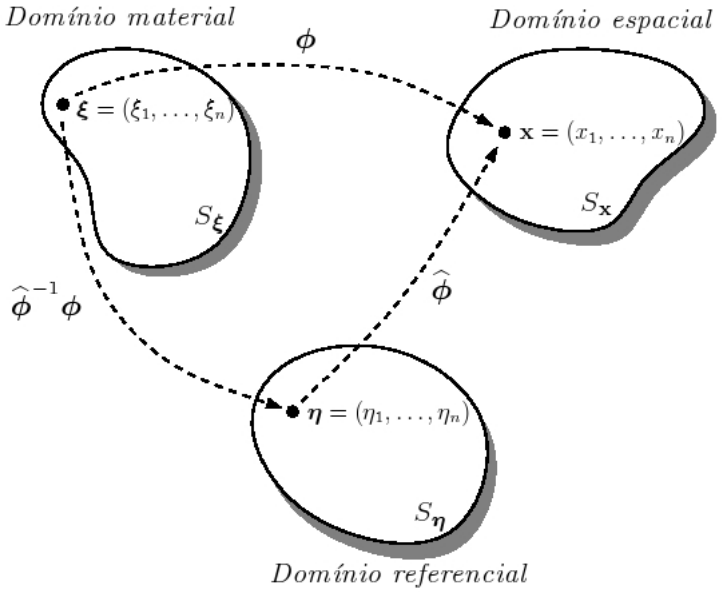


Figura 2: Domínio utilizados na formulação ALE.

A terceira abordagem, a formulação Lagrangeana-Euleriana Arbitrária (ou ALE, isto é, *Arbitrary Lagrangian-Eulerian*), é uma forma mais geral de se descrever o movimento de um fluido. Nesta, as equações são desenvolvidas em um referencial se movendo no espaço com velocidade arbitrária. Quando este referencial se move com a mesma velocidade do fluido, obtém-se a descrição Lagrangeana, e quando o referencial está parado, obtém-se a descrição Euleriana. Segundo (SIMEONI, 2005), essa descrição é muito útil para o caso de métodos com malhas dinâmicas, onde as equações são escritas no referencial da malha que se move com velocidade arbitrária. Assim, baseando-se em (SIMEONI, 2005), será descrita a variação temporal das propriedades macroscópicas do fluido em um referencial se movendo com velocidade arbitrária, a fim de se determinar, posteriormente, as equações de conservação utilizando-se a descrição Lagrangeana-Euleriana arbitrária. Sendo assim, considerando-se os domínios material ($S_{\boldsymbol{\xi}}$), referencial ou computacional ($S_{\boldsymbol{\eta}}$) e espacial ($S_{\mathbf{x}}$), mostrados na Figura 2, na descrição Lagrangeana-Euleriana arbitrária, os domínios $S_{\boldsymbol{\xi}}$ e $S_{\boldsymbol{\eta}}$ estão se movendo enquanto $S_{\mathbf{x}}$ está parado. Considerando-se que uma partícula $\boldsymbol{\xi}$ de fluido que ocupa o domínio $S_{\boldsymbol{\xi}}$ se move para um ponto $\mathbf{x}$ em um dado instante t no domínio espacial, o movimento de um fluido pode ser descrito por uma transformação ϕ tal que $\mathbf{x} = \boldsymbol{\phi}(\boldsymbol{\xi}, t)$. Assim, define-se *velocidade* da partícula $\boldsymbol{\xi}$ no domínio espacial como

$$\mathbf{u} = \left.\frac{\partial \mathbf{x}}{\partial t}\right|_{\boldsymbol{\xi}} = \frac{\partial}{\partial t}\,\boldsymbol{\phi}(\boldsymbol{\xi}, t).$$

Da mesma forma, seja $\widehat{\phi}$ a transformação que descreve o movimento de um ponto $\boldsymbol{\eta}$ no domínio $S_{\boldsymbol{\eta}}$ para um ponto $\mathbf{x}$ no domínio $S_{\mathbf{x}}$, então define-se velocidade do

ponto $\boldsymbol{\eta}$ no domínio espacial como

$$\hat{\mathbf{u}} = \left.\frac{\partial \mathbf{x}}{\partial t}\right|_{\boldsymbol{\eta}} = \frac{\partial}{\partial t}\,\widehat{\boldsymbol{\phi}}(\boldsymbol{\eta}, t).$$

Seja f uma propriedade expressa como função de $(\boldsymbol{\xi}, t)$ ou $(\mathbf{x}, t)$, ou seja, expressa através de uma descrição material (Lagrangeana) ou espacial (Euleriana). Em uma formulação Lagrangeana-Euleriana arbitrária, a expressão de f como função de $(\boldsymbol{\eta}, t)$ é chamada de descrição referencial. Considerando-se as seguintes derivadas temporais

$$\left.\frac{\partial}{\partial t} f(\mathbf{x}, t)\right|_{\boldsymbol{\xi}} = \left.\frac{\partial}{\partial t} f(\boldsymbol{\phi}(\boldsymbol{\xi}, t), t)\right|_{\boldsymbol{\xi}} = \left.\frac{\partial f}{\partial t}\right|_{\mathbf{x}} + \frac{\partial}{\partial t}\,\boldsymbol{\phi}(\boldsymbol{\xi}, t) \cdot \nabla f = \left.\frac{\partial f}{\partial t}\right|_{\mathbf{x}} + (\mathbf{u} \cdot \nabla) f, \tag{1.1}$$

$$\left.\frac{\partial}{\partial t} f(\mathbf{x}, t)\right|_{\boldsymbol{\eta}} = \left.\frac{\partial}{\partial t} f(\widehat{\boldsymbol{\phi}}(\boldsymbol{\eta}, t), t)\right|_{\boldsymbol{\eta}} = \left.\frac{\partial f}{\partial t}\right|_{\mathbf{x}} + \frac{\partial}{\partial t}\,\widehat{\boldsymbol{\phi}}(\boldsymbol{\eta}, t) \cdot \nabla f = \left.\frac{\partial f}{\partial t}\right|_{\mathbf{x}} + (\hat{\mathbf{u}} \cdot \nabla) f, \tag{1.2}$$

subtraindo-se (1.2) de (1.1), tem-se

$$\frac{Df}{Dt} = \left.\frac{\partial f}{\partial t}\right|_{\boldsymbol{\xi}} = \left.\frac{\partial f}{\partial t}\right|_{\boldsymbol{\eta}} + ((\mathbf{u} - \hat{\mathbf{u}}) \cdot \nabla)\, f$$

que é a *derivada material* ou *substantiva* da propriedade f em um referencial $\boldsymbol{\eta}$ que se move com velocidade $\hat{\mathbf{u}}$.

Note que, em uma descrição Lagrangeana, o domínio referencial se desloca com a mesma velocidade do domínio material, ou seja, $\mathbf{u} = \hat{\mathbf{u}}$, e portanto a variação temporal da propriedade f nos domínios material e referencial é a mesma.

Já em uma descrição Euleriana, o domínio referencial está parado em relação ao domínio espacial, ou seja, $\hat{\mathbf{u}} = \mathbf{0}$, e portanto a expresão da variação temporal de f no domínio material é dada por

$$\frac{Df}{Dt} = \left.\frac{\partial f}{\partial t}\right|_{\boldsymbol{\xi}} = \left.\frac{\partial f}{\partial t}\right|_{\boldsymbol{\eta}} + (\mathbf{u} \cdot \nabla)\, f$$

que corresponde à derivada material de f utilizando-se uma descrição Euleriana.

1.2 Equações de Conservação

A partir da seção 1.1, pode-se alcançar as equações de conservação (de *massa* e *quantidade de movimento linear*), utilizando-se a formulação Lagrangeana-Euleriana arbitrária. Porém, é necessário, antes de tudo, enunciar dois teoremas importantes para o desenvolvimento desta seção.

Teorema 1.1 (Divergência de Gauss) *Seja $\Omega \subset \mathbb{R}^m$, $\Gamma = \partial\Omega$ o bordo de Ω e $\boldsymbol{u} : U \subset \mathbb{R}^m \longrightarrow \mathbb{R}^m$ um campo vetorial. Então*

$$\int_\Omega \nabla \cdot \boldsymbol{u}\, d\Omega = \int_\Gamma \boldsymbol{u} \cdot \boldsymbol{n}\, d\Gamma$$

onde $\boldsymbol{n}$ é o vetor normal à Γ.

Teorema 1.2 (Teorema de Leibnitz) *Sejam $R(t) \subset \mathbb{R}^m$ uma região movendo-se no espaço com velocidade $\boldsymbol{w}$, $S(t) = \partial R(t)$ o bordo de $R(t)$ e $\boldsymbol{T}$ uma grandeza de dimensão $k < \infty$. Então*

$$\frac{d}{dt}\int_{R(t)} \boldsymbol{T}(\boldsymbol{x},t)\, dV = \int_R \frac{\partial}{\partial t}\boldsymbol{T}(\boldsymbol{x},t)\bigg|_x dV + \int_S (\boldsymbol{n}\cdot\boldsymbol{w})\boldsymbol{T}\, dS.$$

1.2.1 Equação de Conservação de Massa

Sabendo-se que a massa total numa região material no espaço $V(t) \subset \mathbb{R}^m$ é dada por

$$m(V(t),t) = \int_{V(t)} \rho(\mathbf{x},t)\, dV,$$

onde $\rho(\mathbf{x},t)$ é a massa por unidade de volume (*massa específica*).

Sabendo-se que, pelo princípio da conservação de massa, na ausência de fontes ou sorvedouros, a variação temporal de massa na região $V(t)$ é nula, ou seja,

$$\frac{d}{dt}\, m(V(t),t) = \frac{d}{dt}\int_{V(t)} \rho(\mathbf{x},t)\, dV = 0. \tag{1.3}$$

Aplicando-se o Teorema 1.2 em 1.3, encontra-se

$$\int_V \frac{\partial}{\partial t}\rho(\mathbf{x},t)\bigg|_{\mathbf{x}} dV + \int_S (\mathbf{n}\cdot\mathbf{u})\rho\, dS = 0,$$

onde $S(t) = \partial V(t)$ é o bordo e $\mathbf{u}$ a velocidade da região material $V(t)$, e $\mathbf{n}$ é um vetor unitário normal a $S(t)$. Aplicando-se o Teorema 1.1 na equação anterior, tem-se

$$\int_V \left(\frac{\partial}{\partial t}\,\rho(\mathbf{x},t)\bigg|_{\mathbf{x}} + \nabla\cdot\ (\rho\mathbf{u})\right) dV = 0,$$

e, uma vez que essa equação deve ser verificada para qualquer região fixa de V, é necessário que a expressão entre parênteses seja identicamente nula (GAMA, 2005). Sendo assim,

$$\frac{\partial}{\partial t}\rho(\mathbf{x},t)\bigg|_{\mathbf{x}} + \nabla\cdot(\rho\mathbf{u}) = 0$$

ou,

$$\underbrace{\left.\frac{\partial \rho}{\partial t}\right|_{\boldsymbol{\eta}} + ((\mathbf{u} - \hat{\mathbf{u}}) \cdot \nabla)\rho}_{\text{Derivada Material, seção 1.1, com } f = \rho.} + \rho \nabla \cdot \mathbf{u} = 0,$$

substiuindo-se por (1.2) e utilizando-se a identidade

$$\nabla \cdot (\rho \mathbf{u}) = \rho \nabla \cdot \mathbf{u} + (\mathbf{u} \cdot \nabla)\rho,$$

ou, ainda,

$$\frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{u} = 0, \tag{1.4}$$

utilizando-se a notação de derivada material expressa na seção 1.1. No caso deste exemplo, ou seja, simulação de escoamentos multifásicos, cada fluido envolvido determina uma fase. Assim, foi considerado, no trabalho desenvolvido em (SIMEONI, 2005), o fluido em todas as fases como um contínuo, onde as propriedades mudam de acordo com a posição de partículas marcadoras de interface em cada ponto do escoamento como um todo. Desta forma, as equações de conservação de massa e de quantidade de movimento foram desenvolvidas com propriedades, como massa específica (ρ) e viscosidade (μ), variáveis para o escoamento como um todo, mas constantes no interior de cada uma das fases separadamente. Devido ao fato de este trabalho se basear nesta referência, considera-se a mesma abordagem com relação as fases do fluido.

Sendo assim, seja $\boldsymbol{\Omega} = \cup_i \boldsymbol{\Omega}_i$, onde $\boldsymbol{\Omega}_i$ é a região ocupada pelo fluido i, tem-se

$$\rho(\mathbf{x}) = \rho_i \,, \text{ se } \mathbf{x} \in \boldsymbol{\Omega}_i \,,$$
$$\mu(\mathbf{x}) = \mu_i \,, \text{ se } \mathbf{x} \in \boldsymbol{\Omega}_i \,.$$

Desta forma, tem-se que ρ e μ variam de acordo com a posição, mas são constantes no interior de cada subdomínio $\boldsymbol{\Omega}_i$ e não se alteram em relação ao tempo, o que permite escrever que

$$\frac{D\rho}{Dt} = 0,$$

e, substituindo em (1.4), tem-se

$$\rho \nabla \cdot \mathbf{u} = 0.$$

Como $\rho \neq 0$ para todas as fases, pode-se afirmar que

$$\nabla \cdot \mathbf{u} = 0. \tag{1.5}$$

A equação (1.5) é conhecida como condição de incompressibilidade, e será

mantida em todo o domínio. Os fluidos que obedecem esta condição são chamados *incompressíveis*.

1.2.2 Equação de Conservação de Quantidade de Movimento Linear

A equação de conservação da quantidade de movimento linear é obtida aplicando-se a segunda lei de Newton, a qual afirma que a taxa de variação temporal da quantidade de movimento de uma partícula é igual a resultante das forças que agem sobre essa partícula. Considere novamente uma região material arbitrária $V(t) \in \Omega$ movendo-se no espaço, limitada por uma superfície $S(t) = \partial V(t)$.

O balanço integral da quantidade de movimento pode ser expresso pela equação

$$\frac{d}{dt}\int_{V(t)} \rho\mathbf{u}(\mathbf{x},t)\,dV = \int_S \boldsymbol{\sigma}\cdot\mathbf{n}\,dS + \int_{V(t)} \rho\mathbf{b}(\mathbf{x},t)\,dV$$

Na equação acima, o lado esquerdo representa a taxa de variação de quantidade de movimento linear (produto da massa pela aceleração), a primeira integral do lado direito representa a resultante das tensões agindo sobre a superfície S, e a segunda integral do lado direito representa a resultante das forças de campo.

Aplicando-se o Teorema 1.2 na equação acima tem-se

$$\int_V \left.\frac{\partial(\rho\mathbf{u})}{\partial t}\right|_{\mathbf{x}} dV + \int_S (\mathbf{n}\cdot\mathbf{u})\rho\mathbf{u}\,dS = \int_S \boldsymbol{\sigma}\cdot\mathbf{n}\,dS + \int_V \rho\mathbf{b}\,dV$$

e utilizando o Teorema 1.1 para transformar as integrais de superfície em integrais de volume, tem-se

$$\int_V \left.\frac{\partial(\rho\mathbf{u})}{\partial t}\right|_{\mathbf{x}} dV + \int_V \nabla\cdot(\rho\mathbf{u}\otimes\mathbf{u})\,dV = \int_V \nabla\cdot\boldsymbol{\sigma}\,dV + \int_V \rho\mathbf{b}\,dV,$$

onde $\otimes$ representa o produto diádico entre dois vetores (ver (SIMEONI, 2005)). Assim

$$\int_V \left(\left.\frac{\partial(\rho\mathbf{u})}{\partial t}\right|_{\mathbf{x}} + \nabla\cdot(\rho\mathbf{u}\otimes\mathbf{u})\nabla\cdot\boldsymbol{\sigma} - \rho\mathbf{b}\right) dV = 0,$$

de onde vem

$$\left.\frac{\partial(\rho\mathbf{u})}{\partial t}\right|_{\mathbf{x}} + \nabla\cdot(\rho\mathbf{u}\otimes\mathbf{u}) - \nabla\cdot\boldsymbol{\sigma} - \rho\mathbf{b} = 0.$$

Utilizando-se a Equação (1.2) para substituir a derivada temporal, e distribuindo o operador divergente do segundo termo, obtém-se

$$\left.\frac{\partial(\rho\mathbf{u})}{\partial t}\right|_{\boldsymbol{\eta}} - (\hat{\mathbf{u}}\cdot\nabla)(\rho\mathbf{u}) + (\mathbf{u}\cdot\nabla)(\rho\mathbf{u}) + \rho\mathbf{u}(\nabla\cdot\mathbf{u}) - \nabla\cdot\boldsymbol{\sigma} - \rho\mathbf{b} = 0,$$

que, pela Equação (1.5), resulta em

$$\left.\frac{\partial(\rho\mathbf{u})}{\partial t}\right|_{\boldsymbol{\eta}} + ((\mathbf{u}-\hat{\mathbf{u}})\cdot\nabla)\rho\mathbf{u} \;-\; \nabla\cdot\boldsymbol{\sigma} \;-\; \rho\mathbf{b} = 0,$$

e finalmente, com a notação de derivada material (seção 1.1) tem-se

$$\frac{D(\rho\mathbf{u})}{Dt} = \nabla\cdot\boldsymbol{\sigma} \;+\; \rho\mathbf{b}. \tag{1.6}$$

Na equação (1.6), $\boldsymbol{\sigma}$ representa o tensor de tensões totais do escoamento. Utilizando um modelo *Newtoniano* isotrópico, este tensor pode ser escrito como

$$\boldsymbol{\sigma} = -p\,\mathbf{I} + \boldsymbol{\tau}$$

onde $p = p(\mathbf{x}, t)$ representa o campo de pressão do escoamento, e $\boldsymbol{\tau}$ é o tensor de tensões viscosas dado por

$$\boldsymbol{\tau} = 2\mu\mathbf{D} \;+\; \left(\lambda - \frac{2}{3}\mu\right)(\nabla\cdot\mathbf{u})\;\mathbf{I}, \tag{1.7}$$

sendo μ o coeficiente de viscosidade dinâmica e λ o coeficiente de viscosidade volumétrica do fluido. Devido a condição de incompressibilidade (1.5), pode-se escrever simplesmente

$$\boldsymbol{\tau} = 2\mu\mathbf{D}. \tag{1.8}$$

O tensor $\mathbf{D}$ que aparece nas equações (1.7) e (1.8), é denominado *tensor deformação*, e é dado por

$$\mathbf{D} = \frac{1}{2}\dot{\boldsymbol{\gamma}}(\mathbf{u}) = \frac{1}{2}\left(\nabla\mathbf{u} + \nabla\mathbf{u}^T\right),$$

onde $\dot{\boldsymbol{\gamma}}(\mathbf{u})$ é denominado de *tensor taxa de deformação*.

Deste modo, utilizando o tensor de tensões *Newtoniano* isotrópico, pode-se calcular o divergente do tensor de tensões totais:

$$\nabla\cdot\boldsymbol{\sigma} = \nabla\cdot\left[-p\mathbf{I} + \mu\left(\nabla\mathbf{u} \;+\; \nabla\mathbf{u}^T\right)\right] = -\nabla p + \nabla\cdot\left[\mu\left(\nabla\mathbf{u} + \nabla\mathbf{u}^T\right)\right].$$

O vetor $\mathbf{b}$ é uma força por unidade de massa que age sobre o fluido na região Ω. Para o tipo de escoamento que é objeto de interesse neste trabalho, somente o campo gravitacional é considerado, ou seja, $\mathbf{b} = \mathbf{g}$. Finalmente, a equação de

conservação de quantidade de movimento pode ser expressa como

$$\frac{D(\rho\mathbf{u})}{Dt} = -\nabla p + \nabla \cdot \left[\mu\left(\nabla\mathbf{u} + \nabla\mathbf{u}^T\right)\right] + \rho\mathbf{g}. \tag{1.9}$$

As equações da continuidade (1.5) e da conservação de quantidade de movimento linear (1.9) são as equações de conservação que modelam escoamentos incompressíveis, e são também chamadas de equações de *Navier-Stokes*, que, na forma adimensional (veja (SIMEONI, 2005), para maiores detalhes), podem ser escritas como

$$\frac{\partial(\rho\mathbf{u})}{\partial t} + ((\mathbf{u}-\hat{\mathbf{u}})\cdot\nabla)\rho\mathbf{u} = -\nabla p + \frac{1}{N^{\frac{1}{2}}}\nabla \cdot \left[\mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\right] + \rho\mathbf{g} + \frac{1}{E_0}\mathbf{f} \tag{1.10}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{1.11}$$

onde

$$N = \frac{\rho_0^2 D^3 g}{\mu_0^2} \quad \text{e} \quad E_0 = \frac{\rho_0 g D^2}{\sigma_0},$$

são o *Número de Galileu* e o *Número de Eötvös* respectivamente, e onde ρ_0, μ_0 e σ_0 são, respectivamente, valores de referência para massa específica, viscosidade e coeficiente de tensão superficial, D é o diâmetro (aparece em escoamento com bolhas) e g é a força gravitacional. A força $\mathbf{f}$ definida por

$$\mathbf{f} = -\sigma\kappa\nabla H,$$

onde $\sigma = \sigma(\mathbf{x}, t) \in \mathbb{R}$ é o coeficiente de tensão interfacial entre os fluidos, $\kappa = \kappa(\mathbf{x}, t) \in \mathbb{R}$ denota a curvatura local da superfície que define a interface e $H : \mathbb{R}^m \longrightarrow \{0, 1\}$ é uma função de *Heaviside* que pode ser definida como 1 onde existe um determinado fluido ou 0 caso contrário, representa, no caso de escoamentos multifásicos, uma força campo que está ligada à tensão interfacial, e portanto, deve agir diretamente na interface entre dois fluidos

1.3 Método de Elementos Finitos

Nesta seção, será feita uma breve descrição do *Método de Elementos Finitos*, a fim de se gerar uma base teórica para a obtenção dos sistemas de equações deste capítulo. Para maiores detalhes sobre o assunto, veja (SIMEONI, 2005), (ANJOS, 2007), (ZIENKIEWICZ; TAYLOR, 2000a) e (ZIENKIEWICZ; TAYLOR, 2000b).

1.3.1 Conceitos Básicos

O método de elementos finitos é uma eficiente ferramenta numérica de resolução de problemas de meio contínuo que consiste na divisão deste meio, isto é, deste domínio Ω, em subdomínios Ω_i conhecidos como elementos que, por sua vez, podem ser obtidos através de técnicas de geração de malha como, por exemplo, a triangulação de Delaynay para malhas 2D ou, a tetraedralização Delaynay para malhas 3D.

Para se ter uma idéia básica de como é este método, considere uma função $\phi : \mathbb{R}^m \longrightarrow \mathbb{R}$, válida no domínio Ω com condições de contorno estabelecidas como sendo a solução exata do problema

$$\mathcal{L}\phi = f, \tag{1.12}$$

onde $\mathcal{L}$ é um operador diferencial. O método de elementos finitos busca uma aproximação $\Phi(\mathbf{x})$ da solução exata $\phi(\mathbf{x})$ de maneira particionada em cada elemento Ω^e do domínio, ou seja, a aproximação de $\phi(\mathbf{x})$ restrita a cada elemento Ω^e é tal que $\phi^e(\mathbf{x}) = 0$ para qualquer ponto $\mathbf{x}$ fora de Ω^e. Assim, a aproximação de $\phi(\mathbf{x})$ pode ser escrita como

$$\Phi(\mathbf{x}) = \sum_e \phi^e(\mathbf{x})\,.$$

O valor de ϕ avaliado nos vértices da malha que discretiza Ω é chamado de valor nodal de ϕ, denotado por ϕ_i, onde i é o número do nó correspondente, isto é,

$$\phi_i = \phi(\mathbf{x}_i)\,.$$

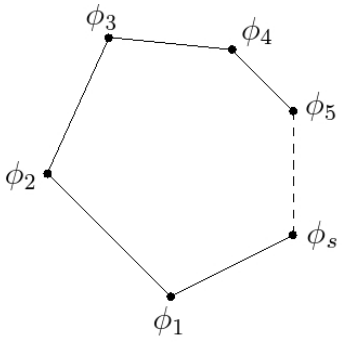


Figura 3: Elemento arbitrário com s lados.

Definem-se graus de liberdade de um vértice (ou nó) como sendo a quantidade de variáveis associadas com este mesmo nó, e graus de liberdade de um elemento como sendo o somatório dos graus de liberdade dos nós que definem tal elemento.

Considere um elemento poligonal com s lados, como mostrado na figura 3, e considere que, associado a cada vértice do elemento está o valor de ϕ avaliado naquele nó, digamos ϕ_i, $i = 1, \ldots, s$. A cada elemento pode ser associada uma família de funções de interpolação, também chamadas de funções de forma, representada genericamente pela matriz

$$\mathbf{N}^e(\mathbf{x}) = \begin{bmatrix} N^e_{1,1}(\mathbf{x}) & \cdots & N^e_{1,s}(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ N^e_{r,1}(\mathbf{x}) & \cdots & N^e_{r,s}(\mathbf{x}) \end{bmatrix}$$

onde r representa o grau de liberdade de cada nó, e s representa a quantidade de nós em cada elemento. Portanto, o número de graus de liberdade no elemento é dado por rs. Assumindo que cada nó tenha somente um grau de liberdade, tem-se

$$\mathbf{N}^e(\mathbf{x}) = [N^e_1(\mathbf{x}), \ldots, N^e_s(\mathbf{x})] \ .$$

Assim, sendo

$$\boldsymbol{\delta}^e = [\phi_1, \ldots, \phi_s]^T \ ,$$

o que é chamado de vetor deslocamento na literatura de elementos finitos, tem-se que a aproximação para ϕ em cada elemento é dada por

$$\phi^e(\mathbf{x}) = \mathbf{N}^e \boldsymbol{\delta}^e \tag{1.13}$$

e a solução aproximada de (1.12) pode ser escrita como

$$\Phi(\mathbf{x}) = \sum_{e=1}^{NE} \phi^e(\mathbf{x}) = \sum_{e=1}^{NE} \mathbf{N}^e \boldsymbol{\delta}^e \ , \tag{1.14}$$

onde NE é o número total de elementos. Tendo sido decidido o tipo de discretização do domínio, o próximo passo é escolher como são as funções de interpolação. A escolha mais comum é optar por funções polinomiais, pois além de serem mais fáceis de manipular, tanto algebricamente quanto computacionalmente, aproximações polinomiais possuem propriedades garantidas pelo teorema da aproximação de *Weierstrass*, o qual especifica que qualquer função contínua pode ser aproximada por um polinômio, tão próximo quanto se deseje.

A escolha de tais aproximações polinomiais é feita seguindo-se algumas observações:

1. O número total de termos no polinômio deve ser igual ao número total de graus de liberdade do elemento. Isto garante a unicidade da aproximação;

2. A aproximação não deve ter uma direção preferencial, ou seja, deve ter invariân-

cia geométrica;

3. Para garantir convergência, as incógnitas devem ser contínuas, e a aproximação polinomial tal que permita com que elas assumam qualquer forma linear arbitrária.

Apesar disso, é difícil estabelecer regras gerais que possam ser aplicadas a todos os casos. Geralmente não é ideal acrescentar na aproximação polinomial termos de alta ordem ao custo da exclusão de outros de baixa ordem, o que pode causar oscilações na interpolação. Assim, o uso de polinômios completos é mais recomendável.

O problema de se determinar as funções de interpolação através do uso de aproximações polinomiais leva à inversão de matrizes mal-condicionadas. Por isso, a melhor forma de se obter tais funções é através de interpolação direta no elemento. Isto é feito escolhendo-se convenientemente um conjunto de polinômios de interpolação $N_i^e(\mathbf{x})$ de maneira que, se $\mathbf{x}_j$ representa as coordenadas do nó j, então

$$N_i^e(\mathbf{x}_j) = \delta_{ij}, \quad i, j = 1, \ldots, s \tag{1.15}$$

onde δ_{ij} representa o delta de *Kroneck*, dado por

$$\delta_{ij} = \begin{cases} 1, & \text{se } i = j, \\ 0, & \text{se } i \neq j. \end{cases} \tag{1.16}$$

É claro que as funções de forma, obtidas desta maneira, devem satisfazer as condições 1, 2 e 3 vistas anteriormente. Em particular, se as funções de forma satisfazem a condição 3, então

$$\sum_{i \in e} N_i^e = 1, \quad \sum_{i \in e} N_i^e \mathbf{x}_i = \mathbf{x} \ .$$

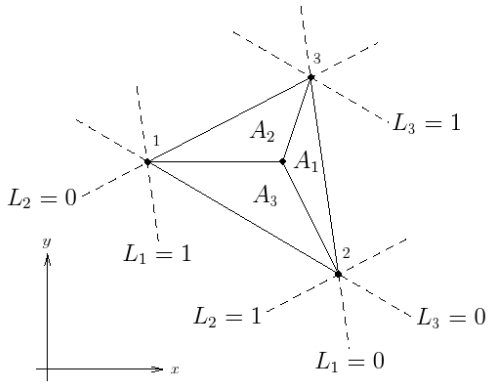


Figura 4: Determinação de funções de interpolação para um elemento triangular.

Para exemplificar o uso dessas funções, considera-se as funções de interpolção para o elemento triangular. Apesar de ser possível a determinação de funções de interpolação em coordenadas globais para o triângulo, o uso de coordenadas locais simplifica e padroniza os cálculos na hora de se resolver as integrais do modelo variacional. Considere então um sistema de coordenadas locais no triângulo (L_1, L_2, L_3), que na Figura 4 é dado por

$$L_1 = \frac{A_1}{A}, \quad L_2 = \frac{A_2}{A}, \quad L_3 = \frac{A_3}{A}, \tag{1.17}$$

onde $A = A_1 + A_2 + A_3$ é a área do triângulo e A_1, A_2 e A_3 são as áreas mostradas na figura 4. Note que, por exemplo, qualquer ponto sobre a aresta entre os vértices 1 e 2 tem coordenada $L_3 = 0$, e se o ponto coincide com o vértice 3, então $L_3 = 1$, uma vez que $A_3 = A$. Desta forma, a posição de qualquer ponto dentro do triângulo pode ser representada pelas coordenadas (L_1, L_2, L_3), também chamadas de coordenadas baricêntricas. Note ainda que

$$L_1 + L_2 + L_3 = 1 \ . \tag{1.18}$$

A relação entre o sistema de coordenadas global (x, y) e o sistema de coordenadas local (L_1, L_2, L_3) é dada pelas equações

$$x = L_1 x_1 + L_2 x_2 + L_3 x_3 \ , \tag{1.19}$$

$$y = L_1 y_1 + L_2 y_2 + L_3 y_3 \ , \tag{1.20}$$

que podem ser resolvidas para obter L_i em termos de x e y como

$$L_i(x, y) = \frac{a_i + b_i x + c_i y}{2A} \ , \tag{1.21}$$

onde a área A é dada por

$$A = \frac{1}{2}\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} . \tag{1.22}$$

As constantes a_i, b_i e c_i são dadas em termos das coordenadas nodais

$$a_i = x_j y_k - x_k y_j , \tag{1.23}$$

$$b_i = y_j - y_k , \tag{1.24}$$

$$c_i = x_k - x_j , \tag{1.25}$$

onde j e k são permutações cíclicas de i. De (1.21) as seguintes relações para as derivadas podem ser obtidas

$$\frac{\partial L_i}{\partial x} = \frac{b_i}{2A}, \quad \frac{\partial L_i}{\partial y} = \frac{c_i}{2A} . \tag{1.26}$$

Finalmente, um resultado sobre integração em coordenadas de área, importante para a discretização por elementos triangulares, é dado por

$$\int_A L_1^m L_2^n L_3^p \, dxdy = \frac{2Am!n!p!}{(m+n+p+2)!} , \tag{1.27}$$

cuja prova pode ser encontrada em (DAVIES, 1980).

Considerando um elemento triangular com um grau de liberdade em cada nó, a aproximação da variável ϕ no elemento segue a seguinte forma

$$\phi^e(x, y) = \alpha_0 + \alpha_1 x + \alpha_2 y , \tag{1.28}$$

e utilizando as coordenadas baricêntricas do triângulo e como funções de interpolação, isto é,

$$\mathbf{N}^e = \begin{bmatrix} L_1 & L_2 & L_3 \end{bmatrix} , \tag{1.29}$$

pode-se escrever

$$\phi^e(x, y) = \mathbf{N}^e(x, y)\boldsymbol{\delta}^e = \begin{bmatrix} L_1(x, y) & L_2(x, y) & L_3(x, y) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix} . \tag{1.30}$$

Funções de interpolação para elementos de mais alta ordem podem ser determinadas, mas são facilmente encontradas em qualquer livro de elementos finitos, como (DAVIES, 1980) ou para mais detalhes (ZIENKIEWICZ; TAYLOR, 2000a).

Utilizando-se a teoria descrita acima e discretizando-se as equações de *Navier-Stokes*, em sua formulação variacional, no espaço (através do método de Taylor-Galerkin) e em relação ao tempo por um sistema semi-implícito (veja (SIMEONI, 2005) para maiores detalhes), têm-se

$$\mathbf{M}_\rho\left(\frac{\mathbf{u}^{n+1}-\boldsymbol{u}^n}{\Delta t}\right)+\mathbf{A}\boldsymbol{u}^n+\frac{1}{Re}\mathbf{K}\boldsymbol{u}^{n+1}-\mathbf{G}\boldsymbol{p}^{n+1}-\frac{1}{Fr^2}\mathbf{M}_\rho\boldsymbol{g}-\frac{1}{We}\mathbf{M}\boldsymbol{f}=\mathbf{0} \tag{1.31}$$

$$\mathbf{D}\boldsymbol{u}^{n+1}=\mathbf{0} \tag{1.32}$$

Os números Re, Fr e We são, respectivamente, os números de *Reynold*, *Froude* e *Weber*, dados por

$$Re=\frac{\rho_0 LU}{\mu_0}=\frac{LU}{\nu_0},$$
$$Fr=\frac{U}{\sqrt{gL}} \text{ e}$$
$$We=\frac{\rho_0 LU^2}{\sigma_0}$$

onde ρ_0, L, U, μ_0, g e σ_0 são os valores de referência para massa específica, comprimento, velocidade, viscosidade, força gravitacional e coeficiente de tensão superficial. O valor ν_0 representa a viscosidade cinemática e é a razão entre a viscosidade aparente ou dinâmica μ_0 e a massa específica. As matrizes de (1.31) são dadas por

$$\mathbf{M}_\rho=\begin{bmatrix}\mathbf{M}_{\rho,x} & \mathbf{0}\\ \mathbf{0} & \mathbf{M}_{\rho,y}\end{bmatrix}_{2NV\times 2NV} \qquad \mathbf{M}=\begin{bmatrix}\mathbf{M}_x & \mathbf{0}\\ \mathbf{0} & \mathbf{M}_y\end{bmatrix}_{2NV\times 2NV}$$

$$\mathbf{A}=\begin{bmatrix}\mathbf{A}_x & \mathbf{0}\\ \mathbf{0} & \mathbf{A}_y\end{bmatrix}_{2NV\times 2NV} \qquad \mathbf{K}=\begin{bmatrix}\mathbf{K}_{xx} & \mathbf{K}_{xy}\\ \mathbf{K}_{yx} & \mathbf{K}_{yy}\end{bmatrix}_{2NV\times 2NV}$$

$$\mathbf{G}=\begin{bmatrix}\mathbf{G}_x\\ \mathbf{G}_y\end{bmatrix}_{2NV\times NP} \qquad \mathbf{D}=\begin{bmatrix}\mathbf{D}_x & \mathbf{D}_y\end{bmatrix}_{NP\times 2NV}.$$

onde

$$\begin{array}{llll}\mathbf{M}_{\rho,x}=\mathcal{A}_x(\mathbf{m}^e_\rho), & \mathbf{M}_{\rho,y}=\mathcal{A}_y(\mathbf{m}^e_\rho), & \mathbf{M}_x=\mathcal{A}_x(\mathbf{m}^e), & \mathbf{M}_y=\mathcal{A}_y(\mathbf{m}^e),\\ \mathbf{K}_{xx}=\mathcal{A}_x(\mathbf{k}^e_{xx}), & \mathbf{K}_{xy}=\mathcal{A}_x(\mathbf{k}^e_{xy}), & \mathbf{K}_{yx}=\mathcal{A}_y(\mathbf{k}^e_{yx}), & \mathbf{K}_{yy}=\mathcal{A}_y(\mathbf{k}^e_{yy}),\\ \mathbf{A}_x=\mathcal{A}_x(\mathbf{a}^e), & \mathbf{A}_y=\mathcal{A}_y(\mathbf{a}^e), & \mathbf{G}_x=\mathcal{A}_x(\mathbf{g}^e_x), & \mathbf{G}_y=\mathcal{A}_y(\mathbf{g}^e_y),\\ & \mathbf{D}_x=\mathcal{A}_x(\mathbf{d}^e_x), & \mathbf{D}_y=\mathcal{A}_y(\mathbf{d}^e_y), & \end{array}$$

tal que as submatrizes $\mathbf{m}^e_\rho$, $\mathbf{m}^e$, $\mathbf{a}^e$, $\mathbf{k}^e_{xx}$, $\mathbf{k}^e_{xy}$, $\mathbf{k}^e_{yx}$, $\mathbf{k}^e_{yy}$, $\mathbf{g}^e_x$, $\mathbf{g}^e_y$, $\mathbf{d}^e_x$ e $\mathbf{d}^e_y$, que são matrizes

definidas localmente para cada elemento, são dadas por

$$m^e_{\rho,ij} = \int_{\Omega^e} \rho^e N^e_i N^e_j \, d\Omega \tag{1.33}$$

$$m^e_{ij} = \int_{\Omega^e} N^e_i N^e_j \, d\Omega \tag{1.34}$$

$$a^e_{ij} = \int_{\Omega^e} \rho^e N^e_i \left((u^e - \hat{u}^e) \frac{\partial N^e_j}{\partial x} + (v^e - \hat{v}^e) \frac{\partial N^e_j}{\partial y} \right) \, d\Omega \tag{1.35}$$

$$k^e_{xx,ij} = \int_{\Omega^e} \mu^e \left(\frac{\partial N^e_i}{\partial x} \frac{\partial N^e_j}{\partial x} \right) d\Omega \tag{1.36}$$

$$k^e_{xy,ij} = \int_{\Omega^e} \mu^e \left(\frac{\partial N^e_i}{\partial y} \frac{\partial N^e_j}{\partial x} \right) d\Omega \tag{1.37}$$

$$k^e_{yx,ij} = \int_{\Omega^e} \mu^e \left(\frac{\partial N^e_i}{\partial x} \frac{\partial N^e_j}{\partial y} \right) d\Omega \tag{1.38}$$

$$k^e_{yy,ij} = \int_{\Omega^e} \mu^e \left(\frac{\partial N^e_i}{\partial y} \frac{\partial N^e_j}{\partial y} \right) d\Omega \tag{1.39}$$

$$g^e_{x,ik} = \int_{\Omega^e} \frac{\partial N^e_i}{\partial x} P^e_k \, d\Omega \tag{1.40}$$

$$g^e_{y,ik} = \int_{\Omega^e} \frac{\partial N^e_i}{\partial y} P^e_k \, d\Omega \tag{1.41}$$

$$d^e_{x,kj} = \int_{\Omega^e} \frac{\partial N^e_j}{\partial x} P^e_k \, d\Omega \tag{1.42}$$

$$d^e_{y,kj} = \int_{\Omega^e} \frac{\partial N^e_j}{\partial y} P^e_k \, d\Omega \tag{1.43}$$

O operador $\mathcal{A}$ que aparece acima é um operador que monta as submatrizes de elemento nas submatrizes das matrizes de (1.31).

A equação (1.31) pode ainda ser reescrita como

$$\left(\mathbf{M}_\rho + \frac{\Delta t}{Re} \mathbf{K} \right) \boldsymbol{u}^{n+1} = -\Delta t \left(\mathbf{A}\boldsymbol{u}^n - \mathbf{G}\boldsymbol{p}^{n+1} - \frac{1}{Fr^2} \mathbf{M}_\rho \boldsymbol{g} - \frac{1}{We} \mathbf{M} \boldsymbol{f} \right) + \mathbf{M}_\rho \boldsymbol{u}^n \tag{1.44}$$

que juntamente com (1.32), formam um sistema de equações que pode ser representado da seguinte forma

$$\begin{bmatrix} \mathbf{B} & -\Delta t \mathbf{G} \\ \mathbf{D} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}^{n+1} \\ \boldsymbol{p}^{n+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}^n \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \boldsymbol{bc}_1 \\ \boldsymbol{bc}_2 \end{bmatrix} \tag{1.45}$$

onde agora o sistema é escrito apenas para as incógnitas do problema, ou seja, $\boldsymbol{u}^{n+1} = [u_1^{n+1}, \ldots, u_{Nu}^{n+1}, v_1^{n+1}, \ldots, v_{Nv}^{n+1}]^T$, $p^{n+1} = [p_1^{n+1}, \ldots, p_{Np}^{n+1}]^T$, sendo Nu, Nv e Np o número de incógnitas (nós livres) para velocidade na direção x, velocidade na direção y e pressão respectivamente. A notação para as matrizes e vetores foi mantida a mesma

por simplicidade. A matriz $\mathbf{B}$ é dada por

$$\mathbf{B} = \mathbf{M}_\rho + \frac{\Delta t}{Re}\mathbf{K}$$

que é simétrica e positiva-definida, e o lado direito representa as grandezas conhecidas no tempo n,

$$\boldsymbol{r}^n = -\Delta t \left(\mathbf{A}\boldsymbol{u}^n - \frac{1}{Fr^2}\mathbf{M}_\rho \boldsymbol{g} - \frac{1}{We}\mathbf{M}\boldsymbol{f} \right) + \mathbf{M}_\rho \boldsymbol{u}^n \ , \tag{1.46}$$

mais as condições de contorno que nada mais são do que as contribuições dos valores conhecidos de velocidade e pressão no lado direito do sistema.

Agora, para se obter um conjunto de sistemas onde velocidade e pressão aparecem de forma desacoplada, será aplicado, na seção a seguir, o método da projeção baseado em decomposição LU.

1.4 Método da Projeção baseado em decomposição *LU*

Os métodos para solução das equações de *Navier-Stokes* para escoamentos de fluidos podem ser, de maneira geral, classificados em *métodos acoplados* e *segregados*. Os métodos acoplados (ZIENKIEWICZ; TAYLOR, 2000a; FORTUNA, 2000) buscam resolver o sistema completo de equações a cada ciclo computacional, enquanto que os métodos segregados (CHORIN, 1968) buscam o desacoplamento entre as equações, separando o sistema não-linear multidimensional em problemas mais simples, que podem ser resolvidos sequencialmente. Os métodos da projeção, podem ainda ser classificados em métodos contínuos, métodos semi-discretos, ou de passo fracionário (GRESHO, 1990; GRESHO; CHAN, 1990), e métodos discretos, baseados em decomposição LU ((PEROT, 1993), (NI; KOMORI; MORLEY, 2003) e (LEE; OH; KIM, 2001)). Aqui, será usado este último.

No método baseado na decomposição LU a separação (ou *split*) entre velocidade e pressão é feita somente após a discretização espacial das mesmas. Portanto, considerando o sistema (1.45), o método consiste em decompor a matriz do sistema através de uma fatoração por blocos. Utilizando-se uma fatoração LU por blocos canônica, tem-se o seguinte sistema

$$\begin{bmatrix} \mathbf{B} & \mathbf{0} \\ \mathbf{D} & \Delta t \mathbf{D}\mathbf{B}^{-1}\mathbf{G} \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\Delta t \mathbf{B}^{-1}\mathbf{G} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}^{n+1} \\ \boldsymbol{p}^{n+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}^n \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \boldsymbol{bc}_1 \\ \boldsymbol{bc}_2 \end{bmatrix} \tag{1.47}$$

Uma aproximação comum para a inversa $\mathbf{B}^{-1}$ para este problema é considerar

$\mathbf{B}^{-1} = \mathbf{M}_{\rho}^{-1}$ (CHANG; GIRALDO; PEROT, 2002), e portanto, o sistema (1.45) fica

$$\mathbf{B}\tilde{\boldsymbol{u}} = \boldsymbol{r}^{n} + \boldsymbol{bc}_1 \tag{1.48}$$

$$\Delta t \mathbf{D}\mathbf{M}_{\rho}^{-1}\mathbf{G}\boldsymbol{p}^{n+1} = -\mathbf{D}\tilde{\boldsymbol{u}} + \boldsymbol{bc}_2 \tag{1.49}$$

$$\boldsymbol{u}^{n+1} = \tilde{\boldsymbol{u}} + \Delta t \mathbf{M}_{\rho}^{-1}\mathbf{G}\boldsymbol{p}^{n+1}, \tag{1.50}$$

onde procedimento para a solução das equações é dado na seguinte ordem:

1. Resolve-se $\tilde{\boldsymbol{u}}$ de (1.48);
2. Resolve-se $\boldsymbol{p}^{n+1}$ de (1.49);
3. Encontra-se a velocidade final $\boldsymbol{u}^{n+1}$ usando (1.50);

Nos capítulos a seguir, serão discutidas algumas técnicas computacionais que podem ser utilizadas nos sistemas das equações (1.48), (1.49) e (1.50).

2 CONSIDERAÇÕES SOBRE MATRIZES ESPARSAS

Neste capítulo, serão feitas algumas considerações sobre matrizes esparsas, comuns em sistemas resultantes de grandes simulações. Serão discutidas aqui, técnicas como *armazenamento sparso de matrizes* (seção 2.1), *representação em grafos* (seção 2.2) e *reodenamento* (seção 2.3).

2.1 Armazenamento

Apesar de as matrizes, neste texto consideradas, apresentarem um número elevado de elementos, são poucos os que são diferentes de zero (matrizes *esparsas*). Portanto, por limitações físicas (memória), é fundamental otimizar o armazenamento dessas matrizes, possibilitando, assim, maior velocidade de acesso aos elementos. Nesta seção, serão vistas algumas formas de se armazenar matrizes esparsas. Para isso, considera-se como exemplo a matriz esparsa **X** (equação (2.1)) de ordem $n = 5$,

$$\mathbf{X} = \begin{bmatrix} 1 & 3 & 0 & 0 & 0 \\ 2 & 4 & -2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 5 & 4 \\ 4 & 0 & 0 & 0 & 9 \end{bmatrix}. \tag{2.1}$$

Note que a matriz **X** não é simétrica e, se fosse o caso, poderia ser armazenada apenas a sua parte triangular inferior ou superior, cabendo ao algoritmo de leitura o tratamento dos dados considerando a simetria. Também é importante notar que, apesar de **X** em (2.1) ser quadrada, os tipos de armazenamento aqui mencionados também se aplicam a matrizes retangulares.

2.1.1 Armazenamento em Coordenadas

O modo mais direto de se armazenar uma matriz esparsa é através do *armazenamento em coordenadas* (ou COORD, isto é, *coordinate storage*), onde se guarda os elementos não nulos, juntamente com os índices de suas linhas e de suas colunas. Três estruturas de dados são utilizadas para este tipo de armazenamento:

- Um vetor $val(i)$ de tamanho nz, para se armazenar os elementos não nulos da matriz que, neste texto, são valores de ponto flutuante;
- Um vetor de inteiros $row\_ind(i)$ de tamanho nz, para se armazenar os índices das linhas;

- Um vetor de inteiros $col\_ind(i)$ de tamanho nz, para se armazenar os índices das colunas.

Onde nz é o número total de elementos não nulos da matriz e $i \in \{0, 1, \cdots, nz-1\}$[1].

Neste caso, a matriz **X** de (2.1), ficaria representada da seguinte forma:

$val(i)$	1	3	2	4	-2	1	5	4	4	9
$row\_ind(i)$	0	0	1	1	1	2	3	3	4	4
$col\_ind(i)$	0	1	0	1	2	2	3	4	0	4

2.1.2 Armazenamento Compactado por Linha

Este formato, *compactado por linha* (ou CRS, isto é, *compressed row storage*) se diferencia do anterior no vetor $row\_ind(i)$ que é substituído pelo vetor $row\_ptr(numrow)$ de tamanho $m+1$ (ou seja, o número de linhas da matriz mais um), que armazena a posição do primeiro elemento não-nulo de cada linha, contado de forma contínua, e reserva sua última posição para o número total elementos não nulos (nz).

Sendo assim, a matriz **X** ficaria representada da seguinte forma:

$col\_ind(i)$	0	1	0	1	2	2	3	4	0	4
$val(i)$	1	3	2	4	-2	1	5	4	4	9
$row\_ptr(numrow)$	0	2	5	6	8	10				

Onde $numrow \in \{0, 1, 2, \ldots, m\}$.

2.1.3 Armazenamento Compactado por Coluna

Parecido com o formato CRS, no *armazenamento compactado por coluna* (CCS, isto é, *compressed column storage*), é armazenada a posição do primeiro elemento não nulo de cada coluna, contado de forma contínua. Estas posições estão contidas no vetor $col\_ptr(numcol)$ de tamanho $n+1$ (isto é, número de colunas da matriz mais um). Desta forma, a matriz (2.1) ficaria:

$val(i)$	1	3	2	4	-2	1	5	4	4	9
$row\_ind(i)$	0	0	1	1	1	2	3	3	4	4
$col\_ptr(numcol)$	0	3	5	7	8	10				

Onde $numcol \in \{0, 1, 2, \ldots, n\}$.

[1] Quando se trata da Linguagem C/C++. Para linguagens como MATLAB e Fortran, tem-se $i \in \{1, 2, 3, \ldots, nz\}$.

Observação 2.1 *Arquivos no formato Harwell Boeing (HB), utilizados nos pacotes que implementam os métodos iterativos e pré-condicionadores, utilizam o formato CCS.*

Note que, nos formatos CRS e CCS tem-se, respectivamente, $row\_ptr(numrow + 1) - row\_ptr(numrow)$ igual ao número de elementos não nulos da linha $numrow$ e $col\_ptr(numcol + 1) - col\_ptr(numcol)$ igual ao número de elementos não nulos da coluna $numcol$ da matriz.

2.1.4 Armazenamento em Blocos Compactado por Linha

O *armazenamento em blocos compactado por linha* (ou BCSR, isto é, *block compressed sparse row*) é similar ao CRS, exceto pelo fato de que, aqui, os elementos são substituídos por blocos densos de dimensão $r \times c$. Ou seja, o formato BCSR com os parâmetros $r = 1$ e $c = 1$ é equivalente ao CRS. Uma matriz no formato BCRS é armazenada em três vetores: um contendo os blocos densos, outro contendo o índice da coluna do primeiro elemento de cada bloco e outro contendo o indice do elemento de início de cada *block-row* (linha formada por blocos) contado continuamente. Como exemplo, considerando a matriz,

$$\mathbf{Y} = \begin{bmatrix} 18 & 19 & 20 & 21 & 0 & 0 \\ 0 & 29 & 30 & 0 & 0 & 0 \\ 0 & 0 & 40 & 41 & 42 & 43 \\ 0 & 0 & 0 & 50 & 51 & 0 \\ 0 & 0 & 0 & 0 & 0 & 63 \\ 0 & 0 & 0 & 0 & 0 & 73 \end{bmatrix}, \tag{2.2}$$

onde $r = c = 2$. No formato BCRS a matriz $\mathbf{Y}_{m \times n}$ (onde $m = n = 6$) pode ser representada como mostra a Tabela 1, onde, sendo $nnzb$ o número de blocos densos, $k \in \{0, \cdots, \frac{m}{r} + 1\}$, $j \in \{0, \cdots, nnzb\}$ e $i \in \{0, \cdots, nz\}$. Note, ainda na Tabela 1, que o vetor row_start_ind reserva sua última posição para $nnzb$ (que, neste caso, é 5). Assim, $row\_start\_ind(k+1) - row\_start\_ind(k)$ retorna o número de blocos não nulos por *block-row*.

Tabela 1: Representação da matriz **Y** em (2.2) no formato BCRS.

$row\_start\_ind(k)$	0								2								4				5
$col\_ind(j)$	0				2				2				4				4				
$val(i)$	18	19	0	29	20	21	30	0	40	41	0	50	42	43	51	0	0	63	0	73	

Existem outros tipos de armazenamento, tais como *compressed diagonal storage*, *jagged diagonal storage*, *skyline storage*, etc, que por não serem utilizados neste trabalho, não serão abordados.

2.2 Representação em Grafos

Considere o sistema expresso pela equação (1). Sendo **A** uma matriz esparsa, deve-se observar que, quando se aplica um método de resolução a este sistema linear, corre-se o risco de haver um preenchimento da matriz, ocorrendo, assim, o aumento do número de elementos não nulos. Para que o processo de resolução seja eficiente, deve-se evitar este preenchimento.

Neste contexto, a Teoria de Grafos pode ser aplicada a fim de se acompanhar os efeitos do método de resolução sobre o sistema e auxiliar em possíveis reordenamentos (veja seção 2.3). Assim, a estrutura de elementos não nulos de **A** pode ser modelada utilizando-se *grafos direcionais* e os efeitos do método de resolução de sistema de equações aplicado podem ser acompanhados por este grafo.

A subseção 2.2.1 fará um breve resumo sobre a notação da Teoria de Grafos e sua aplicação no estudo de matrizes esparsas.

2.2.1 Notação

Definição 2.1 *Um* grafo direcional $G = (V, E)$ *é formado por um conjunto finito* $V = \{v_1, v_2, \cdots, v_n\}$ *de* $n = |V|$ *(onde* $|V|$ *é o número de elementos do conjunto* V*) elementos chamados* $\mathrm{vértices}$ *e um conjunto finito* $E \subset V \times V$ *de* $m = |E|$ *elementos chamados* $\mathrm{arestas}$*.*

Definição 2.2 *Um grafo* G *é dito* não direcional *ou* simétrico *quando a aresta* $(v_i, v_j) \in E$ *se e somente se* $(v_j, v_i) \in E$*.*

No que diz respeito às matrizes esparsas de sistemas **Ax** $=$ **b**, um *grafo de adjacência* é o grafo $G = (V, E)$ onde n vértices em V representa n variáveis do sistema linear. Suas arestas representam a relação existente entre as equações e as variáveis da seguinte forma: quando $a_{ij} \neq 0$, existirá uma aresta do nó i até o nó j. Esta aresta significa que a equação i tem a variável j. Note que o grafo de adjacência será não direcional quando a matrix for simétrica, isto é, quando $a_{ij} \neq 0$ se e somente se $a_{ij} \neq 0$, $\forall i, j$ tal que $1 \leq i, j \leq n$.

2.3 Reordenamento

A permutação de linhas e colunas de uma matriz esparsa é uma operação bastante comum. Na verdade, reordenar linhas e colunas é um dos tópicos mais importantes, usado em implementações paralelas tanto em métodos diretos quanto em métodos iterativos (SAAD, 2003).

Como foi dito anteriormente, o processo de decomposição, além de computacionalmente caro, pode gerar fatores menos esparsos que a matriz original, e para não perder a esparsidade existem várias técnicas de fatoração incompleta que podem ser utilizados para este fim. No entanto, decomposições incompletas geram fatores aproximados, devido ao descarte de entradas, que podem ter baixa qualidade, então apresentaremos nesta seção um meio que, dentre outras funções, pode minimizar tal problema.

O *ordenamento* ou *reordenamento* das matrizes pode ser, em alguns casos, um procedimento muito simples e computacionalmente barato, que visa diminuir os preenchimentos dos fatores através de matrizes de permutação.

Permutar linhas ou colunas de matrizes trata-se de uma operação simples, do ponto de vista matemático e computacional, porém importantíssima para o uso de implementações em paralelo bem como para técnicas de resolução de sistemas lineares, tanto diretas quanto iterativas. Além disso, representam um ganho significativo no momento que são usadas para garantir a esparsidade das matrizes.

Definição 2.3 *Seja* ***A*** *uma matriz e* $\pi = \{i_1, i_2, \ldots, i_n\}$ *uma permutação de um conjunto* $\{1, 2, \ldots, n\}$*. Então as matrizes*

$$\boldsymbol{A}_{\pi,*} = \{a_{\pi(i),j}\}_{i=1,\ldots,n;j=1,\ldots,m},$$
$$\boldsymbol{A}_{*,\pi} = \{a_{i,\pi(j)}\}_{i=1,\ldots,n;j=1,\ldots,m}$$

são chamadas permutação-π *linha e permutação-*π *coluna de* ***A****, respectivamente.*

Uma matriz de *permutação simples* é a matriz identidade com duas de suas linhas permutadas. Denota-se estas matrizes por $\mathbf{X}_{i,j}$ sendo i e j o número das linhas trocadas. Para trocar linhas de uma matriz **A**, basta que se multiplique **A** pela esquerda por $\mathbf{X}_{i,j}$. Seja π uma permutação arbitrária. Esta permutação é o produto de uma sequência de n trocas consecutivas $\sigma(i_k, j_k)$,$k = 1, \ldots, n$. Cada uma dessas trocas é feita através de uma multiplicação à esquerda por $\mathbf{X}_{i_k,j_k}$. O mesmo pode ser feito pra trocas de coluna ao multiplicar a matriz **A** pela direta por $\mathbf{X}_{i_k,j_k}$. Claramente pode-se ver que $\mathbf{X}_{i,j}^2 = \mathbf{I}$ que é equivalente a $\mathbf{X}_{i,j}\mathbf{X}_{i,j}^{-1}$.

A seguinte proposição apresenta estes fatos de forma mais clara

Proposição 2.1 *Seja π uma permutação resultante de um produto de permutações simples $\sigma(i_k, j_k)$,$k = 1, \ldots, n$. Então*

$$\boldsymbol{A}_{\pi,*} = \boldsymbol{P}_\pi \boldsymbol{A}, \quad \boldsymbol{A}_{*,\pi} = \boldsymbol{A}\boldsymbol{Q}_\pi,$$

onde

$$\boldsymbol{P}_\pi = \boldsymbol{X}_{i_n,j_n}\boldsymbol{X}_{i_{n-1},j_{n-1}} \ldots \boldsymbol{X}_{i_1,j_1},$$
$$\boldsymbol{Q}_\pi = \boldsymbol{X}_{i_1,j_1}\boldsymbol{X}_{i_2,j_2} \ldots \boldsymbol{X}_{i_n,j_n}.$$

Produtos de permutação simples de linhas ou colunas de matrizes são chamados de *matrizes permutação*. É importante observar que $\mathbf{P}_\pi$ e $\mathbf{Q}_\pi$ são inversas uma da outra. Desde que as matrizes elementares $\mathbf{X}_{i_k,j_k}$ são simétricas, pode-se concluir que

$$\mathbf{Q}_\pi = \mathbf{P}_\pi^T = \mathbf{P}_\pi^{-1}.$$

No contexto de resolução de sistemas lineares, quando as linhas de uma matriz são permutadas, a ordem em que as equações são escritas é alterada. Enquanto que quando as colunas são permutadas, as incógnitas são reordenadas.

Tais sistemas, quando obtidos de equações diferenciais parciais, muitas vezes podem gerar matrizes *diagonal-dominantes* (veja definição A.3). Com o intuito de manter esta característica é comum utilizarem-se permutações simétricas para preservar os elementos da diagonal, da seguinte maneira

$$\mathbf{A}_{\pi,\pi} = \mathbf{P}_\pi^T \mathbf{A} \mathbf{P}_\pi.$$

2.3.1 Grau mínimo

Um dos algoritmos mais eficientes e utilizados, para a diminuição de preenchimento nas fatorações LU, é o *algoritmo do grau mínimo* (GEORGE; LIU, 1989), que apesar de suas várias versões não teve seu nome muito alterado. Este algoritmo não utiliza os valores dos elementos da matriz como critério, levando em consideração apenas sua estrutura, pois, ao considerar a matriz diagonal dominante devido à sua origem e fazer sempre uma permutação simétrica não se tem problemas com instabilidade numérica na eliminação gaussiana. Assim, pode-se descrevê-lo de maneira geral como segue.

Ao trabalhar somente com a estrutura de **A**, são simulados de alguma forma, os n passos da eliminação simétrica gaussiana. A cada passo, uma linha, e sua coluna correspondente, da submatriz a ser fatorada, são deslocadas de sua posição original de forma que o número de elementos não nulos da linha ou coluna pivô seja mínimo. Após os n passos desta simulação o novo posicionamento dos pivôs determina o novo

ordenamento.

Diversas vezes, o processo de determinar qual linha deve ser movida de forma a minimizar o número de elementos não nulos encontra empates segundo o critério original. Então, algum tipo de critério de desempate deve ser utilizado, tornando-se fundamental para garantir qualidade à este ordenamento. Diversas técnicas para melhoria deste processo bem como critérios de desempate foi desenvolvida e uma grande variedade pode ser encontrada, juntamente com exemplos numéricos, em (GEORGE; LIU, 1989).

A simulação da eliminação gaussiana é feita, é baseando-se na teoria de grafos (BONDY; MURTY, 1976). Sendo assim, considerando-se $\mathcal{G}(\mathbf{A})$, ou simplesmente $\mathcal{G}$ quando não restar dúvidas, um grafo não direcionado, pois este representa a estrutura de uma matriz simétrica por estrutura, considerando-se também v como um nó de $\mathcal{G}$. Denotaremos por $\text{Adj}_{\mathcal{G}}(v)$ o conjunto de nós adjacentes ao nó v, e o grau de v por $\text{grau}_{\mathcal{G}}(v)$ que representa o número de lados conectados ao nó v.

Quando se permutam as linhas e colunas de **A** usando uma matriz permutação **P** gera-se uma matriz $\mathbf{PAP}^T$, cujo grafo é $\mathcal{G}(\mathbf{PAP}^T)$. Este grafo tem estrutura identica à do grafo $\mathcal{G}(\mathbf{A})$ porém com os nós renomeados ou reordenados, de acordo com a matriz permutação **P**.

Após o primeiro passo da eliminação gaussiana, onde sem perda de generalidade foi eliminado o nó v, surge um novo grafo $\mathcal{G}_v$ que é obtido pela eliminação do nó v bem como os lados conectados a ele e, adicionando novos lados que passarão a conectar os nós adjacentes a v, isto é, todo o conjunto $\text{Adj}_{\mathcal{G}}(v)$. O grau de um nó pode variar ao longo do processo de eliminação pois se v é adjacente a y então

$$grau_{\mathcal{G}_y}(v) \geq grau\mathcal{G}(v) - 1 \geq grau_{\mathcal{G}}(y) - 1.$$

Assim elimina-se apenas um nó de cada vez, o que pode tornar o algoritmo lento. Formas de melhoria deste algoritmo são também apresentadas na referência (BONDY; MURTY, 1976) já citada anteriormente.

Este é um procedimento que reduz preenchimente de forma local, com isto o nó de grau mínimo não é necessariamente o que provoca o menor número de enchimentos. Para melhorar este desenvolvimento é inevitável o aumento do custo computacional.

2.3.2 Symmetric reverse Cuthill-McKee

É fato conhecido que matrizes com banda ou envoltória pequenas mantêm esta propriedade ao se calcular seus fatores triangulares e, portanto, não sofrem muitos preenchimentos no processo de eliminação gaussiana. Reduzir a banda de uma matriz é um critério para redução de preenchimento de um ponto de vista global.

Originalmente, esta idéia surgiu em (CUTHILL; MCKEE, 1969) e ficou conhecido como *algoritmo de Cuthill-McKee (CM)*, cuja idéia é reduzir a banda de uma matriz de estrutura simétrica. Esta estratégia visa posicionar os elementos da matriz o mais próximo possível da diagonal, assim a largura de banda é definida como sendo o maior valor de $|i - j|$ para $a_{ij} \neq 0$.

Primeiro escolhe-se um nó qualquer e o numera com 1. Depois, numeram-se os nós adjacentes ao nó 1 em ordem crescente de seus graus. Este conjunto de nós é dito ser de primeiro nível. Agora, este procedimento deve continuar de acordo com a ordem em que foram nomeados os nós deste nível. Os novos nós são de segundo nível. Repete-se este procedimento até esgotarem-se os nós. Assim, esta forma de ordenamento gera uma estrutura de níveis.

Obviamente é possível fazer algumas melhorias como determinação do nó inicial segundo algum critério e a utilização de critérios de desempate, por exemplo. Um processo iterativo para melhorar este algoritmo utilizando características da estrutura de níveis é:

Procurar nós iniciais que gerem estruturas de níveis de largura pequena, isto é, pequena quantidade de nós por nível. Como observado em (GEORGE; LIU, 1989), aumentando-se a profundidade, ou quantidade de níveis, da estrutura L, deve-se esperar uma diminuição da largura de seus níveis. Em (GEORGE; LIU, 1979), é proposto um algoritmo heurístico que procura determinar nós que gerem estrutura de níveis de profundidade elevada da seguinte maneira. Construa uma estrutura de níveis gerada por um nó r, escolha um nó de grau mínimo x do último nível da estrutura e gere uma nova estrutura a partir dele. Se sua estrutura é mais profunda que a gerada por r então repita o processo.

Apesar de o algoritmo CM ser eficiente, o de maior sucesso na redução do tamanho da envoltória de uma matriz esparsa é construído por uma pequena modificação do CM proposta por George, em sua tese de doutorado ((GEORGE, 1971)), e que consiste em inverter a numeração obtida pelo ordenamento CM, assim este algoritmo ficou conhecido como *algoritmo de Cuthill-McKee reverso* ou *CMR*.

Em termos matriciais temos:

$$\mathbf{A}_r = \mathbf{P}\mathbf{A}_c\mathbf{P}^T$$

onde

$$\mathbf{P} = \begin{bmatrix} & & & 1 \\ & & 1 & \\ & \iddots & & \\ & 1 & & \\ 1 & & & \end{bmatrix},$$

onde $\mathbf{A}_c$ representa a permutação da matriz **A** pelo algoritmo CM e $\mathbf{A}_r$ representa a permutação da matriz **A** pelo algoritmo CMR.

Em (LIU; SHERMAN, 1976) é demonstrado que CMR necessita, no máximo, do mesmo número de operações e espaço de memória utilizado para realização da eliminação gaussiana.

3 MÉTODOS ITERATIVOS DE RESOLUÇÃO DE SISTEMAS LINEARES BASEADOS EM SUBESPAÇOS DE KRYLOV

3.1 Operadores de Projeção

A maioria das técnicas iterativas existentes para resolução de sistemas lineares de grande porte utilizam, de alguma forma, uma projeção. A *projeção* representa uma forma canônica de se extrair uma aproximação da solução de um sistema a partir de um subespaço (neste caso, o *subespaço de Krylov*). Neste capítulo, serão vistos alguns métodos baseados em subespaços de Krylov e seus respectivos algoritmos. Para tal, considera-se como referência as seguintes fontes: (GOLUB; LOAN, 1996), (SAAD, 2003),(VORST, Jan/Feb 2000) e (GREENBAUM, 1997).

3.1.1 Imagem e Núcleo de uma Projeção

Definição 3.1 *Considere P uma transformação linear, P é dita uma projeção, $P : \mathbb{C}^n \to \mathbb{C}^n$ se $P^2 = P$, P é também chamada* idempotente*.*

Definição 3.2 *Seja I uma transformação linear,$I : \mathbb{C}^n \to \mathbb{C}^n$, com $I(x) = x$, I é chamada* transformação identidade*.*

Definição 3.3 *Considere T uma transformação linear, $T : \mathbb{C}^n \to \mathbb{C}^n$, tem-se*

1. *O núcleo de T, $N(T)$, é dado por:*

$$N(T) = \{x \in \mathbb{C}^n \backslash T(x) = 0\}.$$

2. *O domínio de T, $D(T)$, é dado por:*

$$D(T) = \{x \in \mathbb{C}^n \backslash \exists y \in \mathbb{C}^n, T(x) = y\}.$$

3. *A imagem de T, $Im(T)$, é dada por:*

$$Im(T) = \{y \in \mathbb{C}^n \backslash \exists x \in \mathbb{C}^n, T(x) = y\}.$$

Teorema 3.1 *Seja P uma projeção, tem-se então que*

1. $N(P) = Im(I - P)$;
2. $N(P) \cap Im(P) = \{0\}$;
3. *Se* $x \in Im(P) \Rightarrow P(x) = x$.

Teorema 3.2 *O espaço $\mathbb{C}^n$ pode ser decomposto na seguinte soma direta:*

$$\mathbb{C}^n = N(P) \oplus Im(P), \tag{3.1}$$

ou seja,

$$\forall x \in \mathbb{C}^n, \exists x_1 \in Im(P) \wedge \exists x_2 \in N(P) \quad tais \quad que \quad x = x_1 + x_2, \tag{3.2}$$

onde P é uma projeção, $P : \mathbb{C}^n \to \mathbb{C}^n$.

Teorema 3.3 *Para todo par de subespaços M e S de $\mathbb{C}^n$, tais que,a soma direta desses dois subespaços resulta em $\mathbb{C}^n$, tem-se que existe uma única projeção, P, onde $Im(P) = M$ e $N(P) = S$.*

Corolário 3.3.1 *Considerando o Teorema (3.3) pode-se escrever*

$$\mathbb{C}^n = N(P) \oplus Im(P). \tag{3.3}$$

Corolário 3.3.2 *Considerando o Teorema (3.3), seja $L = S^{\perp}$. Então*

$$\begin{cases} u \in M \\ x - u \perp L \end{cases} \tag{3.4}$$

Essa transformação linear P é dita projeção sobre M, ao longo ou paralela ao subespaço linear S, ou, uma projeção sobre M e ortogonal ao subespaço L.

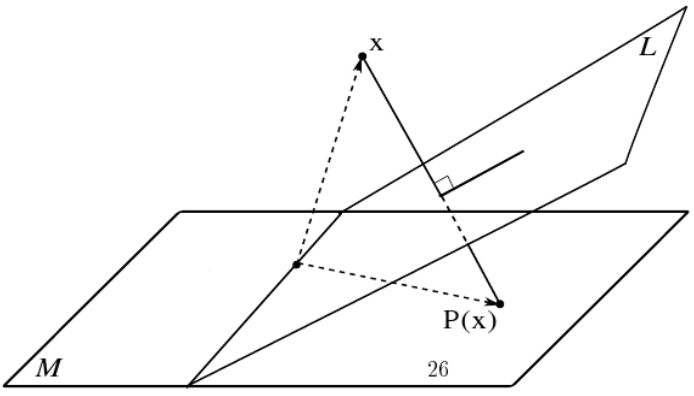


Figura 5: Projeção de x sobre M e ortogonal a L.

Teorema 3.4 *Dados dois subespaços $M, L \in \mathbb{C}^n$, de mesma dimensão m, as duas condições abaixo são equivalentes:*

1. *Nenhum vetor de M diferente de zero é ortogonal a L.*

2. *Para todo $x \in \mathbb{C}^n$ existe um único vetor u que satisfaz às condições dadas em (3.4).*

Uma interpretação geométrica desse teorema seria a de que dados $M, L \in \mathbb{C}^n$, sob as hipóteses do Teorema 3.4, existe um único subespaço $L^\perp$ de $\mathbb{C}^n$ ortogonal a M.

Resumindo, dados em $\mathbb{C}^n$ os subespaços de mesma dimensão, M e L, satisfazendo à condição, $M \cap L^\perp = \{0\}$, existe uma projeção P sobre M e ortogonal a L que define o vetor projeção u para todo $x \in \mathbb{C}^n$ atendendo às condições de (3.4). Essa projeção é tal que:

$$Im(P) = M \quad \text{e} \quad N(P) = L^\perp \tag{3.5}$$

Se $P(x) = 0$, $x \in N(P)$, ou seja, $x \in L^\perp$.

3.1.2 Representação Matricial

Definição 3.4 *Seja a matriz $A \in \mathbb{C}^{n\times n}$,a matriz transposta conjugada de A, notada por A^H é*

$$A^H = \bar{A}^T,$$

onde barra representa a operação de conjugação.

Teorema 3.5 *Considere $M, L \in \mathbb{C}^n$, com $dim(M) = dim(L) = m$, $M \cap L^\perp = \{0\}$, V e W, bases de M e L, respectivamente. Então $W^H V$ é não-singular.*

Definição 3.5 *Dois conjuntos de vetores $A = \{a_1, \ldots, a_m\}$ e $B = \{b_1, \ldots, b_m\}$ são ditos bi-ortogonais se:*

$$(a_i, b_j) = \delta_{ij}, \quad \textit{onde} \quad \delta_{ij} = \begin{cases} 1, \textit{se} & i = j, \\ 0, \textit{se} & i \neq j. \end{cases} \tag{3.6}$$

Na forma matricial tem-se $B^H A = I$.

Teorema 3.6 *Considere $M, L \in \mathbb{C}^n$, com $dim(M) = dim(L) = m$, $M \cap L^\perp = \{0\}$, V e W, bases de M e L, respectivamente. Então, existe uma projeção P, sobre M e com direção ortogonal a L, dada por:*

$$P = V(W^H V)^{-1} W^H. \tag{3.7}$$

Corolário 3.6.1 *Considerando as hipóteses do Teorema (3.6) e, sejam V e W bi-ortogonais, então*

$$P(x) = V W^H x,$$

ou seja,

$$P = VW^H. \tag{3.8}$$

3.1.3 Projeções Ortogonais

Definição 3.6 *Seja o subespaço vetorial, $S \subset \mathbb{C}^n$, sobre um corpo $K \subset \mathbb{C}$. A operação* produto interno, *$(,) : S \times S \to K$, está bem definida se os seguintes axiomas são atendidos:*
Sejam $u, v, w \in S$ e $\alpha, \beta \in K$

1. $(u, v) = \overline{(v, u)}$;
2. $(u, v + w) = (u, v) + (u, w)$;
3. $(\alpha u, v) = \alpha(u, v)$ e $(u, \beta v) = \bar{\beta}(u, v)$;
4. $(u, u) \geq 0$, e, $(u, u) = 0 \Leftrightarrow u = 0$.

Definição 3.7 *Uma projeção P é dita* ortogonal *se*

$$P^H P = P P^H = I, \tag{3.9}$$

Se o espaço vetorial considerado, $S \subset \mathbb{C}^n$, admite um produto interno, tem-se que

$$(u, v) = (Pu, Pv),\ \forall x, \forall y \in S. \tag{3.10}$$

Se a igualdade (3.9) não se verifica, P é dita oblíqua.

Teorema 3.7 *Seja P uma projeção ortogonal de $\mathbb{C}^n$, então o núcleo de P é igual ao complemento ortogonal da imagem de P. Ou seja,*

$$N(P) = Im(P)^{\perp} \tag{3.11}$$

A Figura 6 ilustra a condição acima.

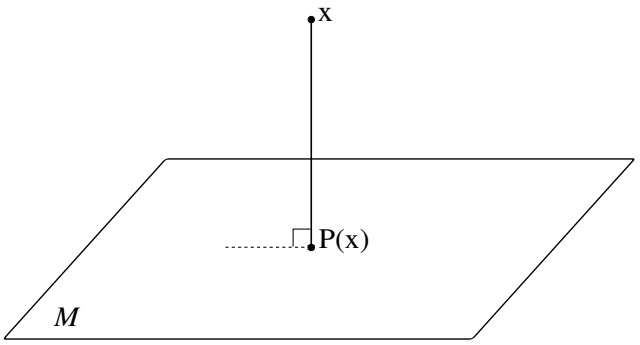


Figura 6: Projeção ortogonal de x sobre M.

Teorema 3.8 *Seja P uma projeção , então P^H, a adjunta de P é, tal que,*

$$(P^H x, y) = (x, Py) \qquad \forall x, \forall y \in \mathbb{C}^n, \tag{3.12}$$

além disso, P^H é uma projeção.

Corolário 3.8.1 *Como conseqüência do Teorema (3.8) tem-se:*

$$N(P^H) = Im(P)^{\perp}, \tag{3.13}$$

$$N(P) = Im(P^H)^{\perp}. \tag{3.14}$$

Teorema 3.9 *Uma projeção é ortogonal se, e somente se, é Hermitiana.*

Teorema 3.10 *Seja P uma projeção ortogonal sobre o subespaço vetorial $M \subset \mathbb{C}^n$, $M = Im(P)$. Considere uma matriz V, de ordem $n \times m$, cujas colunas formam uma base ortonormal de M. Então, podemos representar P pela matriz $P = VV^H$, além disso,*

$$VV^H x \in M \qquad \textbf{e} \qquad (I - VV^H)x \in M^{\perp}. \tag{3.15}$$

Teorema 3.11 *A representação de uma projeção ortogonal, P, não é única, e, para quaisquer duas bases ortonormais, $V_1, V_2 \in M$, com $Im(P) = M$, tem-se que $V_1 V_1^H = V_2 V_2^H$.*

Teorema 3.12 *Seja P uma projeção ortogonal, então*

$$\|x\|_2^2 = \|Px\|_2^2 + \|(I - P)x\|_2^2. \tag{3.16}$$

Corolário 3.12.1

$$\frac{\|Px\|_2}{\|x\|_2} \leq 1, \qquad \forall x \in \mathbb{C}^n. \tag{3.17}$$

Temos ainda que, se tomarmos $x \in Im(P)$*,*

$$P(x) = x \quad \Rightarrow \quad \|P(x)\|_2 = \|x\|_2 \quad \Rightarrow \quad \frac{\|P(x)\|_2}{\|x\|_2} = 1, \tag{3.18}$$

logo o $\sup_{\|x\|_2 \neq 0} \left\{ \frac{\|P(x)\|_2}{\|x\|_2} \right\} = 1$ *é atingido. Portanto:*

$$\|P\|_2 = 1, \tag{3.19}$$

para toda projeção ortogonal P*.*

Uma projeção ortogonal tem apenas dois autovalores, zero com multiplicidade igual à $dim(N(P))$ e, um com multiplicidade igual à $dim(Im(P))$. Qualquer vetor da imagem de P é um autovetor associado ao autovalor um. Qualquer vetor do núcleo de P é um autovetor associado ao autovalor zero.

No próximo teorema uma importante condição de otimalidade é estabelecida.

Teorema 3.13 *Seja* P *uma projeção ortogonal sobre um subespaço* M *de* $\mathbb{C}^n$*. Então, para todo* x *em* $\mathbb{C}^n$*, é verdade que:*

$$\min_{y \in M} \|x - y\|_2 = \|x - Px\|_2. \tag{3.20}$$

Seja $\hat{y} = Px$ para a projeção ortogonal P sobre o subespaço M, tem-se:

Corolário 3.13.1 *Sejam* M *um subespaço de* $\mathbb{C}^n$ *e* $x \in \mathbb{C}^n$*. Então:*

$$\min_{y \in M} \|x - y\|_2 = \|x - \hat{y}\|_2 \tag{3.21}$$

se satisfaz às duas condições abaixo:

$$\begin{cases} y^* \in M, \\ x - y^* \perp M. \end{cases} \tag{3.22}$$

3.2 Subespaço de Krylov

Definição 3.8 *Um subespaço de Krylov é gerado por uma matriz* A*, de ordem* n*, e por um vetor* $v \in \mathbb{C}^n$ *da seguinte forma*

$$[v, Av, \ldots, A^{m-1}v].$$

Notamos tal subespaço por $K_m(A, v)$(SAAD, 2003), (VORST, 2003).

Definição 3.9 *Seja A uma matriz de ordem n. O* polinômio mínimo *de um vetor v é o polinômio mônico não-nulo, p, de menor grau, tal que, $p(A)v = 0$.*

Definição 3.10 *Seja A uma matriz de ordem n, o grau do polinômio mínimo de v com relação a A, é chamado* grau de v com relação a A.

Teorema 3.14 *O subespaço de Krylov $\mathcal{K}_m$ é de dimensão m se o grau μ de v com relação a A é maior ou igual a m, ou seja,*

$$dim(\mathcal{K}_m) = m \quad \leftrightarrow \quad grau(v) \geq m. \tag{3.23}$$

Portanto,

$$dim(\mathcal{K}_m) = \min\{m, grau(v)\}. \tag{3.24}$$

Em vários métodos para resolução de sistemas lineares, o subespaço de Krylov é utilizado como espaço de busca para a melhor solução aproximada de $Ax = b$. Vejamos porque tal subespaço é conveniente.

Dado um sistema linear, $Ax = b$, e x^* sua solução exata. Vamos considerar A não-singular.

Definição 3.11 *Seja A uma matriz de ordem n. O* polinômio característico *de A e dado por*

$$p(t) = \sum_{j=1}^{d} (t - \lambda_j)^{n_j},\ n_j \leq n.$$

onde $\lambda_1, \ldots, \lambda_d$ são autovalores distintos de A.

Definição 3.12 *Seja A uma matriz de ordem n e $p(t) = \sum_{j=1}^{d}(t - \lambda_j)^{n_j}$ seu polinômio característico. O* polinômio mínimo *associado a A, q, é tal que $q(A) = 0$, ou seja,*

$$q(t) = \sum_{j=1}^{d} (t - \lambda_j)^{m_j}, \quad onde \quad 1 \leq m_j \leq n_j.$$

Podemos escrever:

$$q(A) = \sum_{j=0}^{m} \alpha_j A^j,$$

como A é não-singular tem-se que

$$\prod_{j=1}^{d} \lambda_j \neq 0 \quad \rightarrow \quad \alpha_0 \neq 0$$

assim, $0 = q(A) = \alpha_0 I + \alpha_1 A + \ldots + \alpha_m A^m$, como $\alpha_0 \neq 0$

$$A^{-1} = -\frac{1}{\alpha_0} \sum_{j=0}^{m-1} \alpha_{j+1} A^j.$$

Por outro lado, a solução exata do sistema é dada por:

$$\begin{aligned} x^* &= A^{-1}b \\ &= \big(-\frac{1}{\alpha_0} \sum_{j=0}^{m-1} \alpha_{j+1} A^j\big) b \\ &= -\frac{1}{\alpha_0} \sum_{j=0}^{m-1} \alpha_{j+1} A^j b. \end{aligned}$$

Então x^* é escrito como combinação linear de $(b, Ab, \ldots, A^{m-1}b)$, ou seja, $x^* \in K_m(A, b)$ (IPSEN; MEYER, 1998).

Se $x_0 = 0$ nota-se, facilmente, que $K_m(A, b) = K_m(A, r_0)$ e podemos usar $K_m(A, r_0)$ como espaço de busca para a solução aproximada. Suponhamos que $x_0 \neq 0$, então

$$\begin{aligned} x^* &= A^{-1}b \\ &= A^{-1}(Ax_0 + r_0) \\ &= x_0 + A^{-1}r_0, \end{aligned}$$

logo $x^* \in x_0 + K_m(A, r_0)$.

Definição 3.13 *Seja a matriz A, seu autovalor zero tem índice i, então a matriz inversa de Drazin de A é a única matriz A^D, que satisfaz:*

$$A^D A A^D = A^D, \quad A^D A = A A^D, \quad A^{i+1} A^D = A^i.$$

Se A é não-singular, $i = 0$ e $A^D = A^{-1}$.

Teorema 3.15 (Unicidade da solução de Krylov) *Seja m o grau do polinômio mínimo de A, e i o ínidice do autovalor zero de A. Se $b \in Im(A^i)$, então o sistema linear $Ax = b$ tem uma única solução de Krylov, $x = A^D b \in K_{m-i}(A, b)$. Se $b \notin Im(A^i)$ então $Ax = b$ não tem solução no subespaço de Krylov $K_n(A, b)$.*

Veja demonstração em (IPSEN; MEYER, 1998).

3.3 Gradiente Conjugado Pré-condicionado

Nesta seção, será considerado como referência o seguinte trabalho (SHEWCHUK, 1994).

3.3.1 Método do Passo Máximo Descendente

Observação 3.1 *Quando A é simétrica, tem-se*

$$f(x) = \frac{1}{2}x^T Ax - b^T x + c \Rightarrow f'(x) = Ax - b$$

Assim, $f'(x) = 0 \Rightarrow Ax = b$. Se A é positiva-definida (veja (SHEWCHUK, 1994)), a solução de $Ax = b$ minimiza a função f.

Considerando-se o sistema linear da equação 1, onde A é uma matriz simétrica, positiva-definida. Considerando-se um ponto inicial x_0, no método do *Passo Máximo Descendente*, determina-se um melhor caminho de forma a se chegar ao vértice do parabolóide elíptico (ponto de mínimo da forma quadrática, caso A seja positiva-definida), isto é, onde $f'(x) = Ax - b = 0$, através de uma série de pontos x_i até que se encontre pontos suficientemente próximos da solução x do sistema mencionado.

A cada passo, escolhe-se a direção onde f decresce mais rapidamente, que é a direção oposta a $f'(x_i)$, dada por $f'(x_i) = b - Ax$. Assim tem-se que, $r_{(i)} = -f'(x_{(i)}$ e pode-se pensar o resíduo como a direção do passo de descida máxima.

A partir de $x_{(0)}$ escolhe-se o próximo ponto como:

$$x_{(1)} = x_{(0)} + \alpha r_{(0)}. \tag{3.25}$$

Observando-se a equação acima, percebe-se que α minimiza f ao longo de uma reta quando

$$\frac{d}{d\alpha} f(x_{(1)}) = 0,$$

ou seja, quando a *derivada direcional* de f no ponto x_1 é igual a zero. Pela regra da cadeia,

$$\begin{aligned} \frac{d}{d\alpha} f(x_{(1)}) = \\ f'(x_{(1)})^T \frac{d}{d\alpha} x_{(1)} = \\ f'(x_{(1)})^T r_{(0)}. \end{aligned}$$

Igualando essa expressão a zero, conclui-se que α deverá ser tomado tal que $r_{(0)}$ e $f'(x_{(1))}$ sejam ortogonais.

Observando-se que

$$f'(x_{(1))} = -r_{(1)},$$
$$r_{(0)}^T = (b - Ax_{(0)})^T$$

e que, sendo $A^T = A$, tem-se

$$(Ar_{(0)})^T r_{(0)} = (r_{(0)}^T A^T) r_{(0)} = r_{(0)}^T (Ar_{(0)}).$$

Logo, para se determinar α, conclui-se que

$$\begin{aligned}
r_{(1)}^T r_{(0)} &= 0 \\
(b - Ax_{(1)})^T r_{(0)} &= 0 \\
(b - A(x_{(0)} + \alpha r_{(0)}))^T r_{(0)} &= 0 \\
(b - Ax_{(0)})^T r_{(0)} + \alpha (Ar_{(0)})^T r_{(0)} &= 0 \\
(b - Ax_{(0)})^T r_{(0)} &= \alpha (Ar_{(0)})^T r_{(0)} \\
r_{(0)}^T r_{(0)} &= \alpha r_{(0)}^T (Ar_{(0)}) \\
\alpha &= \frac{r_{(0)}^T r_{(0)}}{r_{(0)}^T A r_{(0)}}
\end{aligned}$$

Agrupando-se as equações, o método do Passo Máximo Descendente é resumido por

$$r_{(i)} = b - Ax_{(i)} \tag{3.26}$$
$$\alpha_{(i)} = \frac{r_{(i)}^T r_{()}}{r_{(i)}^T A r_{(i)}} \tag{3.27}$$
$$x_{(i+1)} = x_{(i)} + \alpha_{(i)} r_{(i)} \tag{3.28}$$

Nas equações de (3.26) a (3.28), observa-se duas multiplicações matriz-vetor por iteração. O custo computacional do Passo Máximo Descendente é dominado por estes produtos, porém, pode-se eliminar um deles multiplicando-se a equação (3.28) previamente por $-A$ e adicionando b, assim

$$-Ax_{(i+1)} = -Ax_{(i)} - A\alpha_{(i)} r_{(i)}$$

e, desta forma, obtendo-se

$$r_{(i+1)} = r_{(i)} - \alpha_{(i)} A r_{(i)}. \tag{3.29}$$

A desvantagem de usarmos a recorrência (3.29) é o fato dessa seqüência ser gerada sem o uso dos valores de $x_{(i)}$, assim, a acumulação de erros de arredondamento pode ter conseqüências na convergência de $x_{(i)}$.

3.3.2 Método das Direções Conjugadas

3.3.2.1 Conjugação

Considerando o que foi visto na subseção 3.3.1, é possível estabelecer critérios para uma escolha mais eficiente da direção a ser seguida a cada passo. Sendo assim, pode-se tomar um conjunto ortogonal de direções $d_{(0)}, d_{(1)}, \ldots, d_{(n-1)}$ e, dessa forma, cada direção estaria associada a um passo. Assim, a convergência do método se daria em exatamente n passos.

O passo genérico seria dado por

$$x_{(i+1)} = x_{(i)} + \alpha_{(i)} d_{(i)}. \tag{3.30}$$

Seja $e_{(i+1)}$ o erro associado a cada iteração i, determinando-se o valor de $\alpha_{(i)}$, considerando-se que $e_{(i+1)}$ deve ser ortogonal a $d_{(i)}$ e que nenhum passo seria dado na direção $d_{(i)}$ novamente, tem-se

$$\begin{aligned} d_{(i)}^T e_{i+1)} &= 0 \\ d_{(i)}^T (e_{(i)} + \alpha_{(i)} d_{(i)} &= 0 \\ \alpha_{(i)} &= -\frac{d_{(i)}^T e_{(i)}}{d_{(i)}^T d_{(i)}} \end{aligned} \tag{3.31}$$

Da equação (3.31), observa-se que não se teria avançado nada, pois para se obter o valor de $\alpha_{(i)}$ seria necessário ter calculado o valor de $e_{(i)}$.

Uma forma de solucionar esse problema seria, no lugar de direções ortogonais, utilizar direções que sejam *A-ortoganais*.

Definição 3.14 *Seja A uma matriz $n \times n$, dois vetores $d_{(i)}$ e $d_{(j)}$ são* A-ortogonais*, ou* conjugados*, se:*

$$d_{(i)}^T A d_{(j)} = 0. \tag{3.32}$$

A nova hipótese é que $e_{(i+1)}$ e $d_{(i)}$ sejam A-ortogonais. Essa condição de ortogonalidade é equivalente a encontrar o ponto de mínimo ao longo da direção $d_{(i)}$,

como no método do Passo Máximo Descendente. Desenvolvendo a derivada direcional e igualando-a a zero, tem-se

$$\begin{aligned}
\frac{d}{d\alpha} f(x_{(i+1)}) &= 0 \\
f'(x_{(i+1)})^T \frac{d}{d\alpha} x_{(i+1)} &= 0 \\
(Ax_{(i+1)} - b)^T \frac{d}{d\alpha}(x_{(i)} + \alpha_{(i)} d_{(i)}) &= 0 \\
-r_{(i+1)}^T d_{(i)} &= 0 \\
d_{(i)}^T A e_{(i+1)} &= 0.
\end{aligned}$$

Análoga à derivação da equação (3.31), segue que

$$\alpha_{(i)} = -\frac{d_{(i)}^T A e_{(i)}}{d_{(i)}^T A d_{(i)}} \tag{3.33}$$

$$= -\frac{d_{(i)}^T r_{(i)}}{d_{(i)}^T A d_{(i)}}. \tag{3.34}$$

Agora é possível calcular o valor de $\alpha_{(i)}$ usando a expressão (3.34). É importante notar que, se a direção procurada for o resíduo, a fórmula fica idêntica à usada no método do Passo Máximo Descendente. É possível provar que o Método das Direções conjugadas converge em n passos.

3.3.2.2 Conjugação de Gram-Schmidt

Deseja-se, nesta subseção, encontrar um conjunto A-ortogonal de direções $\{d_{(i)}\}$. Para tal, será descrito aqui um processo simples chamado *processo de conjugação de Gram-Schmidt*.

O processo é semelhante ao processo de ortogonalização de Gram-Schmidt. Para dar início ao processo considera-se n vetores linearmente independentes $u_{(0)}, u_{(1)}, \ldots, u_{(i-1)}$. Para construir uma direção $d_{(i)}$ a partir de cada $u_{(i)}$ é preciso subtrair todas as componentes que não são A-ortogonais aos vetores d anteriores. Ou seja, considerando-se $d_{(0)} = u_{(0)}$, para $i > 0$, tem-se

$$d_{(i)} = u_{(i)} + \sum_{k=0}^{i-1} \beta_{ik} d_{(k)}, \tag{3.35}$$

onde β_{ik} é definido para $i > k$. Para determinar o valor de β_{ik}, usa-se que

$d_{(i)}^T A d_{(j)} = 0$ para $i \neq j$, assim,

$$\begin{aligned} d_{(i)}^T A d_{(j)} &= u_i^T A d_{(j)} + \sum_{k=0}^{i-1} \beta_{ik} d_{(k)}^T A d_{(j)} \\ 0 &= u_i^T A d_{(j)} + \beta_{ij} d_{(j)}^T A d_{(j)}, \qquad i > j \\ \beta_{ij} &= -\frac{u_i^T A d_{(j)}}{d_{(j)}^T A d_{(j)}} \end{aligned} \tag{3.36}$$

A dificuldade resultante de se usar a conjugação de Gram-Schmidt no método das Direções Conjugadas é que todas as direções anteriores devem ser guardadas na memória para se construir a próxima nova direção, e portanto $O(n^3)$ operações são necessárias para se gerar o conjunto todo.

3.3.3 Método do Gradiente Conjugado

Para uma explicação simples do que é o método do Gradiente Conjugado, deve-se ter como base o método das Direções Conjugadas. Para isso, basta que as direções conjugadas sejam construídas pela conjugação dos resíduos.

Uma propriedade útil do resíduo que se deve ter em mente é o fato dele ser ortogonal a cada direção conjugada de algum passo anterior ao passo atual do resíduo considerado, ou seja, $d_{(i)}^T r_{(j)} = 0$ para $i < j$, com isso pode-se garantir que as novas direções produzidas são linearmente independentes, considerando que o resíduo seja não-nulo.

Como cada direção conjugada é construída a partir dos resíduo, tem-se que

$$r_{(i)}^T r_{(j)} = 0 \qquad i \neq j. \tag{3.37}$$

Conclui-se também que o subespaço D_i gerado por $\{r_{(0)}, r_{(1)}, \ldots, r_{(i-1)}\}$ é também gerado pelas direções $\{d_{(0)}, d_{(1)}, \ldots, d_{(i-1)}\}$, e que

$$\begin{aligned} r_{(i+1)} &= -A e_{(i+1)} \\ &= -A(e_{(i)} + \alpha_{(i)} d_{(i)}) \\ &= r_{(i)} - \alpha_{(i)} A d_{(i)} \end{aligned} \tag{3.38}$$

Da equação (3.38), nota-se que cada novo resíduo $r_{(i)}$ é uma combinação linear do resíduo anterior e $Ad_{(i-1)}$. Como $d_{(i-1)} \in D_i$, cada novo subespaço D_{i+1} é

formado pela união do subespaço anterior D_i e o subespaço AD_i.

$$D_i = [d_{(0)}, Ad_{(0)}, A^2 d_{(2)}, \ldots, A^{i-1} d_{(i-1)}]$$

Esse subespaço é o *subespaço de Krylov* (veja seção 3.2), que é criado aplicando-se a matriz A repetidas vezes sobre um vetor. Tem-se, assim, uma propriedade interessante, como $AD_i \in D_{i+1}$, o fato de $r_{(i+1)}$ ser ortogonal a D_{i+1} implica que $r_{(i+1)}$ é A-ortogonal a D_i.

Devido ao fato de ter se partido do resíduo, e segundo a equação (3.36), as constantes de Gram-Schmidt são dadas por $\beta_{ij} = -r_{(i)}^T Ad_{(j)} / d_{(j)}^T Ad_{(j)}$. Simplificando-se essa expressão através do o produto interno de $r_{(i)}$ pela equação (3.38), tem-se

$$\begin{aligned}
r_{(i)}^T r_{(j+1)} &= r_{(i)}^T r_{(j)} - \alpha_{(j)} r_{(i)}^T Ad_{(j)} \\
\alpha_{(j)} r_{(i)}^T Ad_{(j)} &= r_{(i)}^T r_{(j)} - r_{(i)}^T r_{(j+1)} \\
r_{(i)}^T Ad_{(j)} &= \begin{cases} \frac{1}{\alpha_{(i)}} r_{(i)}^T r_{(i)}, & i = j \\ -\frac{1}{\alpha_{(i-1)}} r_{(i)}^T r_{(i)}, & i = j+1, \\ 0, & \text{outros casos} \end{cases} \\
\therefore \beta_{ij} &= \begin{cases} \frac{1}{\alpha_{(i-1)}} \frac{r_{(i)}^T r_{(i)}}{d_{(i-1)}^T Ad_{(i-1)}}, & i = j+1, \\ 0, & i > j+1. \end{cases}
\end{aligned}$$

O que torna o método do Gradiente Conjugado tão usado é o fato desse algoritmo reduzir, tanto o espaço de complexidade como o tempo de complexidade, de $O(n^2)$ para $O(m)$, onde m é o número de entradas não-nulas da matriz A. Pode-se, ainda, fazer a seguinte simplificação na notação, $\beta_{(i)} = \beta_{i,i-1}$ e escrever

$$\begin{aligned}
\beta_{(i)} &= \frac{r_{(i)}^T r_{(i)}}{d_{(i-1)}^T r_{(i-1)}} \\
&= \frac{r_{(i)}^T r_{(i)}}{r_{(i-1)}^T r_{(i-1)}}
\end{aligned}$$

Reunindo as equações usadas até aqui, o método do Gradiente Conjugado é dado

por

$$
\begin{aligned}
d_{(0)} &= r_{(0)} = b - Ax_{(0)}, && (3.39)\\
\alpha_{(i)} &= \frac{r_{(i)}^T r_{(i)}}{d_{(i)}^T A d_{(i)}}, && (3.40)\\
x_{(i+1)} &= x_{(i)} + \alpha_{(i)} d_{(i)}, &&\\
r_{(i+1)} &= r_{(i)} + \alpha_{(i)} A d_{(i)}, && (3.41)\\
\beta_{(i+1)} &= \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}}, && (3.42)\\
d_{(i+1)} &= r_{(i+1)} + \beta_{(i+1)} d_{(i)}. && (3.43)
\end{aligned}
$$

3.3.4 Gradiente Conjugado Pré-condicionado

Para pré-condicionar o gradiente conjugado, isto é, pré-condicionar o sistema que será resolvido utilizando-se o gradiente conjugado, é necessário manter sua matriz simétrica após a aplicação do pré-condicionador. Mesmo que **M** seja simétrica, $\mathbf{M}^{-1}\mathbf{A}$ ou $\mathbf{A}\mathbf{M}^{-1}$ não são necessariamente simétricas.

Para contornar essa dificuldade, pode-se fatorar **M** simétrica e positiva definida como $\mathbf{M} = \mathbf{E}\mathbf{E}^T$ (onde **E** pode ser o fator de Cholesky por exemplo).

Assim transformarmos o sistema $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ em

$$\mathbf{E}^{-1}\mathbf{A}\mathbf{E}^{-T}\hat{\boldsymbol{x}} = \mathbf{E}^{-1}\boldsymbol{b}, \quad \hat{\boldsymbol{x}} = \mathbf{E}^T\boldsymbol{x}, \tag{3.44}$$

onde resolve-se primeiro o sistema para $\hat{x}$ e depois para x.

Como $\mathbf{M}^{-1} = \mathbf{E}^{-1}\mathbf{A}\mathbf{E}^{-T}$ é simétrica e positiva-definida então pode-se utilizar o gradiente conjugado para resolver este sistema, porém agora este passa a se chamar *gradiente conjugado pré-condicionado*. No entanto, não é necessário modificar todo algoritmo do gradiente conjugado para resolver o problema pré-condicionado. Toda transformação que simplifica a aplicação do pré-condicionador no algoritmo de gradiente conjugado pode ser encontrada em detalhes em (SHEWCHUK, 1994). Abaixo, encontra-se, o algoritmo do gradiente conjugado pré-condicionado onde o pré-condionador somente precisa ser aplicado no resíduo onde, de forma equivalente, pode ser aplicado pela esquerda ou pela direita.

$$
\begin{aligned}
\boldsymbol{r}_{(0)} &= \boldsymbol{b} - \mathbf{A}\boldsymbol{x}_{(0)}, \\
\boldsymbol{d}_{(0)} &= \mathbf{M}^{-1}\boldsymbol{r}_{(0)}, \\
\alpha_{(i)} &= \frac{\boldsymbol{r}_{(i)}^T \mathbf{M}^{-1}\boldsymbol{r}_{(i)}}{\boldsymbol{d}_{(i)}^T \mathbf{A}\boldsymbol{d}_{(i)}}, \\
\boldsymbol{x}_{(i+1)} &= \boldsymbol{x}_{(i)} + \alpha_{(i)}\boldsymbol{d}_{(i)}, \\
\boldsymbol{r}_{(i+1)} &= \boldsymbol{r}_{(i)} - \alpha_{(i)}\mathbf{A}\boldsymbol{d}_{(i)}, \\
\beta_{(i+1)} &= \frac{\boldsymbol{r}_{(i+1)}^T \mathbf{M}^{-1}\boldsymbol{r}_{(i+1)}}{\boldsymbol{r}_{(i)}^T \mathbf{M}^{-1}\boldsymbol{r}_{(i)}}, \\
\boldsymbol{d}_{(i+1)} &= \mathbf{M}^{-1}\boldsymbol{r}_{(i+1)} + \beta_{(i+1)}\boldsymbol{d}_{(i)},
\end{aligned}
$$

isto é,

Algoritmo 3.1 Gradiente Conjugado Pré-condicionado

1. Entrar com $x^{(0)}$
2. Calcular $r^{(0)} = b - Ax^{(0)}$
3. **Enquanto** critério de convergência não for atingido **faça**
4. resolva $Mz^{(i-1)} = r^{(i-1}$
5. $\rho_{i-1} = {r^{(i-1)}}^T z^{(i-1)}$
6. **Se** $i = 1$
7. $p^1 = z^{(0)}$
8. **senão**
9. $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
10. $p^(i) = z^{(i-1)} + \beta_{(i-1)}p^{(i-1)}$
11. **fim-se**
12. $q^{(i)} = Ap^{(i)}$
13. $\alpha_i = \rho_{i-1}/{p^{(i)}}^T q^{(i)}$
14. $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
15. $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
16. Atualizar critério de convergência
17. **fim-enquanto**

Apenas é necessário que se resolva um sistema $\mathbf{M}^{-1}\boldsymbol{x} = \boldsymbol{r}$ para o resíduo $\boldsymbol{r}$ como lado direito a cada iteração. Aliás, ainda é possível armazenar o vetor obtido de $\mathbf{M}^{-1}\boldsymbol{r}_{(i)}$ pois é utilizado várias vezes. Assim, é necessário resolver apenas um sistema linear para o pré-condicionador por iteração, garantindo uma melhoria significativa em quantidade de operações e consequentemente, em tempo.

É claro que, se tiver sido necessário fatorar $\mathbf{M}$, como no caso de utilizar-se a fatoração de Cholesky, então utiliza-se os fatores triangulares $\mathbf{E}$ e $\mathbf{E}^T$ resolvendo dois sistemas retângulares, para resolver este novo sistema.

Nem sempre é necessário calcular $\mathbf{M}$ ou $\mathbf{M}^{-1}$ explicitamente, somente é necessário que se calcule o efeito da aplicação de $\mathbf{M}^{-1}$ em um vetor. Isto quer dizer que não é preciso que se tenha um pré-condicionador, é possível ter uma forma ou método

de pré-condicionamento.

3.4 Método do Resíduo Mínimo Generalizado

O método do Resíduo Mínimo Generalizado (ou GMRES, isto é, *Generalized Minimal Residual*) foi proposto por Saad (SAAD; SCHULTZ, 1986), no ano de 1986. É um método iterativo, para resolução de sistemas lineares, onde a cada passo a norma do resíduo é minimizada sobre um subespaço de Krylov escolhido. Tal método é derivado do método de Arnoldi (1952), no qual se obtem uma base ortogonal para um subespaço de Krylov. Esta seção apresentará o GMRES na sua forma clássica (SAAD; SCHULTZ, 1986), mas antes os métodos do Resíduo Conjugado Generalizado (GCR)(EISENSTAT; ELMAN; SCHULTZ, 1983) e da Ortogonalização Completa (FOM)(SAAD, 2003) serão apresentados

3.4.1 Método do Resíduo Conjugado Generalizado

O método do Resíduo Conjugado Generalizado (GCR) é um método iterativo para se resolver sistemas lineares da forma $Ax = b$, onde A é uma matriz de ordem $n \times n$. Neste método, tem-se como hipótese que a parte simétrica de A seja positiva-definida . O GCR é matematicamente equivalente ao GMRES, ou seja, os dois métodos produzem a mesma aproximação, x_k, em aritmética exata.
Segue o algoritmo do GCR.

Algoritmo 3.2 Algoritmo do Resíduo Conjugado Generalizado

1. Escolha x_0 e calcule $r_0 = b - Ax_0$, $p_0 = r_0$
2. **Para** $j = 1, \ldots$ até convergir **faça**
3. $\quad \alpha_j = \frac{(r_j, Ap_j)}{(Ap_j, Ap_j)}$
4. $\quad x_{j+1} = x_j + \alpha_j p_j$
5. $\quad r_{j+1} = r_j - \alpha_j Ap_j$
6. $\quad$ Calcule $\beta_{ij} = -\frac{(Ar_{j+1}, Ap_i)}{(Ap_i, Ap_i)}$ para $i = 1, 2, \ldots, j$
7. $\quad p_{j+1} = r_{j+1} + \sum_{i=0}^{j} \beta_{ij} p_i$
8. **fim-para**

Teorema 3.16 *Considerando o Algoritmo 3.2, as seguintes relações ocorrem:*

1. $(Ap_i, Ap_j) = 0$, *se* $i \neq j$;
2. $(r_i, Ap_j) = 0$, *se* $i > j$;
3. $(r_i, Ap_i) = (r_i, Ar_i)$;

4. $(r_i, Ar_j) = 0$, *se* $i > j$;

5. $(r_i, Ap_i) = (r_0, Ap_i)$, *se* $i \geq j$;

6. $\langle p_0, \ldots, p_i \rangle = \langle p_0, Ap_0, \ldots, A^i p_0 \rangle = \langle r_0, \ldots, r_i \rangle$;

7. *Se* $r_i \neq 0$, *então* $p_i \neq 0$;

8. x_{i+1} *minimiza* $E(\omega) = \|b - A\omega\|_2$ *sobre o espaço afim* $x_0 + \langle p_0, \ldots, p_i \rangle$.

Observação 3.2 *As bases geradas pelo Algoritmo 3.2 não são ortonormais. Então, conclui-se, apenas, que* $W = \{Ap_0, Ap_1, \ldots, Ap_{m-1}\}$ *é uma base ortogonal de* $\mathcal{L}$ *e* $V = \{p_0, p_1, \ldots, p_{m-1}\}$ *é uma base* $A^H A$*-ortogonal de* $\mathcal{K}$.

3.4.2 Método da Ortogonalização Completa

O método da Ortogonalização Completa (FOM) é um método iterativo para se solucionar sistemas lineares, foi apresentado em 1951 em (ARNOLDI, 1951). Considerando-se um sistema linear, $Ax = b$, sendo A uma matriz de ordem m, o método consiste em determinar a solução x_m do sistema.

Inicialmente determina-se uma base ortonormal partindo-se das colunas da matriz A e utilizamos o método de Arnoldi (ARNOLDI, 1951), segue daí que a matriz H é também obtida após todas as iterações.

Considerando-se uma solução inicial x_0 e o resíduo $r_0 = b - Ax_0$, toma-se o subespaço de Krylov $\mathcal{K} = K_m(A, r_0)$ e adota-se a hipótese de *Galerkin*

$$b - Ax_i \perp \mathcal{K}_i.$$

Sendo $v_1 = \frac{r_0}{\|r_0\|}$ e aplicando-se o método de Arnoldi(ARNOLDI, 1951), tem-se

$$V_m^T A V_{m-1} = H_{m-1,m}.$$

A solução aproximada do sistema é dada por

$$\begin{aligned} x_m &= x_0 + V_m y_m, \quad \text{com} \\ y_m &= H_m^{-1}(\|r_0\| e_1). \end{aligned}$$

A seguir,o algoritmo do FOM será escrito, porém é importante que a característica principal desse algoritmo fique clara. No FOM executa-se os m passos da iteração e ao final obtem-se a matriz H_m. Pode-se então calcular y_m, resolvendo o

sistema com a matriz H_m, e determinar a solução x_m do sistema $Ax = b$.
Segue, então, o algoritmo da Ortogonalização Completa.

Algoritmo 3.3 Algoritmo da Ortogonalização Completa

1. Escolha um vetor x_0, compute $r_0 = b - Ax_0$ e $v_1 = r_0/\|r_0\|$.
2. **Para** $j = 1, \ldots, m-1$, **faça**
3. $\quad h_{i,j} = (Av_j, v_i), i = 1, 2, \ldots, j$
4. $\quad \hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i$
5. $\quad h_{j+1,j} = \|\hat{v}_{j+1}\|$
6. $\quad v_{j+1} = \frac{\hat{v}_{j+1}}{h_{j+1,j}}$
7. Forme a solução: $x_k = x_0 + V_k y_k$, onde $y_k = H_k^{-1}\|r_0\|e_1$.

Nesse algoritmo, obtem-se o subespaço de Krylov $\mathcal{K}$ a partir de r_0 e da matriz A. Uma base ortonormal é obtida e, então, determina-se uma solução para o sistema linear considerado. Nota-se que a solução não é obtida a cada nova direção, o que se faz é escolher a ordem do subespaço e gerar uma solução.

3.4.3 Resíduo Mínimo Generalizado

O método do Resíduo Mínimo Generalizado é um método iterativo que vai determinar a solução aproximada, x_i, de um sistema linear da forma $Ax = b$.
Serão considerados os subespaços de Krylov $\mathcal{K}_i = K_i(A, r_0)$ e as soluções aproximadas serão da forma $x_0 + \delta_i$, onde $\delta_i \in \mathcal{K}_i$.
Tal algoritmo, usa o algoritmo de Arnoldi (ARNOLDI, 1951), determinando, assim, uma base ortonormal V_m e a matriz de Hessenberg H_m. Assim, tem-se que a igualdade abaixo

$$V_{i+1}^H A V_i = H_{i+1,i}$$

é valida e pode ser reescrita como

$$AV_i = V_{i+1} H_{i+1,i}.$$

O GMRES tem como ponto principal minimizar a norma do resíduo sobre $\mathcal{K}_i$, ou seja, o que se quer é resolver o seguinte problema

$$\min_{\delta_i \in \mathcal{K}_i} \|b - A(x_0 + \delta_i)\| = \min_{\delta_i \in \mathcal{K}_i} \|r_0 - A\delta_i\|. \tag{3.45}$$

Considerando-se a igualdade $\delta_i = V_i y_i$, defini-se a função J, a qual se deseja minimizar

$$J(y_i) = \|r_0 - A\delta_i\|. \tag{3.46}$$

Seja $\beta = \|r_0\|$, segue que:

$$\begin{aligned} J(y_i) &= \|\beta v_1 - AV_i y_i\|; \\ J(y_i) &= \|V_i(\beta e_1 - H_i y_i)\|, \end{aligned}$$

onde e_1 é o primeiro vetor da base canônica de $\mathbb{R}^m$.
Como V é uma base ortonormal, a função J fica sendo dada por

$$J(y_i) = \|\beta e_1 - H_i y_i\|. \tag{3.47}$$

Assim, tem-se a seguinte solução para o problema de mínimos quadrados dado pela equação (3.45):

$$x_m = x_0 + V_m y_m,$$

onde y_m minimiza a função J em $\mathbb{R}^m$. Segue o algoritmo do GMRES.

Algoritmo 3.4 Algoritmo do Resíduo Mínimo Generalizado

1. Escolha x_0 e calcule $r_0 = b - Ax_0$ e $v_1 = \frac{r_0}{\|r_0\|}$.
2. **Para** $j = 1, \ldots, m$ **faça**
3. $h_{ij} = (Av_j, v_i), i = 1, \ldots, j$;
4. $\hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i$;
5. $h_{j+1,j} = \|\hat{v}_{j+1}\|$; e
6. $v_{j+1} = \frac{\hat{v}_{j+1}}{h_{j+1,j}}$;
7. **fim-para**
8. Forme a solução: $x_m = x_0 + V_m y_m$ onde y_m minimiza $J(y_i) = \|\beta e_1 - H_m y_m\|$, onde $\beta = \|r_0\|$.

É natural que seja levantada a hipótese de o algoritmo falhar. Portanto, é importante observa os resultados a seguir.

Teorema 3.17 *A solução x_j produzida pelo GMRES no passo (j) é exata se e somente se as seguintes condições equivalentes ocorrem:*

1. *O algoritmo falha no passo (j).*
2. $\hat{v}_{j+1} = 0$.
3. $h_{j+1,j} = 0$.
4. *O grau do polinômio mínimo do resíduo inicial r_0 é igual a j.*

Corolário 3.17.1 *Para uma matriz de ondem $n \times n$ o GMRES termina em no máximo n passos.*

Teorema 3.18 *Seja A uma matriz não-singular. Então o algoritmo do GMRES falha no passo j, isto é, $h_{j+1,j} = 0$, se a solução aproximada, x_j, é a solução exata.*

3.5 Método do Resíduo Mínimo Generalizado Pré-condicionado

No caso de se pré-condicionar o GMRES, podem ser utilizadas as mesmas três opções de aplicação de pré-condicionamento que estão disponíveis para o gradiente conjugado (pré-condicionamento pela esquerda, direita e split). No entanto, para o GMRES a forma de pré-condicionamento modifica a forma de medida do resíduo proporcionando alguma diferença na velocidade de convergência do algoritmo.

3.5.1 Pré-condicionamento pela esquerda

Exatamente como apresentado anteriormente, o pré-condicionamento do GMRES pela esquerda é feito através da seguinte alteração do sistema

$$\mathbf{M}^{-1}\mathbf{A}\boldsymbol{x} = \mathbf{M}^{-1}\boldsymbol{b}$$

o que implica na seguinte alteração no algoritmo

Algoritmo 3.5 Algoritmo do GMRES pré-condicionado pela esquerda

1. Calcule $r_0 = M^{-1}(b - Ax_0)$, $\beta = ||r_0||_2$, e $v_1 = r_0/\beta$
2. **Para** $j = 1, \cdots, m$, **Faça**
3. Calcule $w = M^{-1}Av_j$
4. **Para** $i = 1, \cdots, j$, **Faça**
5. $h_{i,j} = (w, v_i)$
6. $w = w - h_{i,j}v_i$
7. **fim-para**
8. Calcule $h_{j+1,j} = ||w||_2$ e $v_{j+i} = w/h_{j+i,j}$
9. **fim-para**
10. Defina $V_m = [v_1, \cdots, v_m]$,
11. Defina $\hat{H}_m = h_{i,j}{}_{1 \leqslant i \leqslant j+1, 1 \leqslant j \leqslant m}$
12. Calcule $y_m = argmin_y ||\beta_{e_1} - \hat{H}_m y||_2$
13. Calcule $x_m = x_0 + V_m y_m$
14. **Se** o critério de convergência for satisfeito Pare
15. **Senão** calcule $x_0 = x_m$ e volte para a linha 1

Todos os resíduos e normas dos resíduos que são calculadas neste algoritmo correspondem ao resíduo pré-condicinado, apresentado como $\boldsymbol{z}_m = \mathbf{M}^{-1}(\boldsymbol{b} - \mathbf{A}\boldsymbol{x}_m)$, ao invés do original (não pré-condicionado) resíduo $\boldsymbol{b} - \mathbf{A}\boldsymbol{x}_m$. Não há como medir o

resíduo não pré-condicionado a menos que o calcule explicitamente através de uma multiplicação por **M**. O que proprciona alguma modificação nos critérios de parada.

3.5.2 Pré-condicionamento pela direita

Como também já foi visto, o pré-condicionamento do algoritmo do GMRES pela diretia é baseado na seguinte equação

$$\mathbf{AM}^{-1}\boldsymbol{u} = \boldsymbol{b}, \text{ com } \boldsymbol{u} = \mathbf{M}\boldsymbol{x}.$$

No entanto a variável $\boldsymbol{u}$ nunca é utilizada explicitamente. E o resíduo sempre pode ser calculado atrvés da utilização do vetor $\boldsymbol{x}$ em $\boldsymbol{r} = \boldsymbol{b} - \mathbf{A}\boldsymbol{x} = \boldsymbol{b} - \mathbf{AM}^{-1}\boldsymbol{u}$. No final o vetor $\boldsymbol{u}$ que aproxima a solução tem a seguinte forma

$$\boldsymbol{u}_m = \boldsymbol{u}_0 + \sum_{i=1}^{m} v_i \eta_i,$$

com $\boldsymbol{u}_0 = \mathbf{M}\boldsymbol{x}_0$.

Em termos da variável x tem-se

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + \mathbf{M}^{-1} \sum_{i=1}^{m} v_i \eta_i.$$

Assim, a única modificação é o momento em que o pré-condicionamento é feito, atentando, é claro, para a possibilidade de um novo critério de parada.

Algoritmo 3.6 Algoritmo do GMRES pré-condicionado pela direita

1. Calcule $r_0 = b - Ax_0$, $\beta = ||r_0||_2$, e $v_1 = r_0/\beta$
2. **Para** $j = 1, \cdots, m$, **Faça**
3. Calcule $w = AM^{-1}v_j$
4. **Para** $i = 1, \cdots, j$, **Faça**
5. $h_{i,j} = (w, v_i)$
6. $w = w - h_{i,j}v_i$
7. **fim-para**
8. Calcule $h_{j+1,j} = ||w||_2$ e $v_{j+i} = w/h_{j+i,j}$
9. **fim-para**
10. Defina $V_m = [v_1, \cdots, v_m]$,
11. Defina $\hat{H}_m = h_{i,j 1 \leqslant i \leqslant j+1, 1 \leqslant j \leqslant m}$
12. Calcule $y_m = argmin_y ||\beta_{e_1} - \hat{H}_m y||_2$
13. Calcule $x_m = x_0 + M^{-1}V_m y_m$
14. **Se** o critério de convergência for satisfeito Pare
15. **Senão** calcule $x_0 = x_m$ e volte para a linha 1

Neste caso, o subespaço de Krylov gerado é

$$\mathcal{K}^m(\mathbf{AM}^{-1}) = span\{\boldsymbol{r}_0, \mathbf{AM}^{-1}\boldsymbol{r}_0, \ldots, (\mathbf{AM}^{-1})^{m-1}\boldsymbol{r}_0\}.$$

3.6 Método do Gradiente Bi-conjugado Estabilizado (Bi-CGSTAB)

O método do Gradiente Bi-Conjugado Estabilizado foi desenvolvido por (VORST, 1992), é um método iterativo para resolução de sistemas lineares, onde a matriz principal do sistema é qualquer.

Nesta seção, serão estudados os algoritmos do Gradiente Bi-Conjugado e do Gradiente Conjugado Quadrado, para que se possa abordar o algoritmo do Gradiente Bi-Conjugado Estabilizado.

3.6.1 Método do Gradiente Bi-Conjugado

Dado um sistema linear, $Ax = b$, onde A é uma matriz de $\mathbb{C}^{n\times n}$. Sejam x_0 a aproximação inicial da solução do sistema e $r_0 = b - Ax_0$ o resíduo inicial.

Considerando dois subespaços de Krylov,

$$\mathcal{K}_i = K_i(A, r_0) \quad \text{e} \quad \mathcal{L}_i = K_i(A^H, \hat{r}_0),$$

$\hat{r}_0$ é escolhido tal que $(r_0, \hat{r}_0) \neq 0$.

As soluções aproximadas, $x_i \in x_0 + K_i(A, r_0)$, serão obtidas de tal forma que o resíduo associado, r_i, seja ortogonal ao subespaço $\mathcal{L}_i$ e, $\hat{r}_i$, ortogonal a $\mathcal{K}_i$.

O método do Bi-CG ((FLETCHER, 1975)) termina em n passos, mas não há nenhuma propriedade de minimização nos passos intermediários como no método do Gradiente Conjugado.

Abaixo, segue o algoritmo do Gradiente Bi-Conjugado.

Algoritmo 3.7 Algoritmo do Gradiente Bi-Conjugado

1. Seja x_0 uma solução inicial. Compute $r_0 = b - Ax_0$
2. Escolha $\hat{r}_0$ tal que $(r_0, \hat{r}_0) \neq 0$, i.e., $r_0 = \hat{r}_0$
3. $\rho_0 = 1$
4. $\hat{p}_0 = p_0 = 0$
5. **Para** $i = 1, 2, 3, \ldots$
6. $\rho_i = (\hat{r}_{i-1}, r_{i-1}); \beta_i = (\frac{\rho_i}{\rho_{i-1}})$
7. $p_i = r_{i-1} + \beta_i p_{i-1}$
8. $\hat{p}_i = \hat{r}_{i-1} + \beta_i \hat{p}_{i-1}$
9. $v_i = Ap_i$
10. $\alpha_i = \frac{\rho_i}{(\hat{p}_i, v_i)}$
11. $x_i = x_{i-1} + \alpha_i p_i$
12. if x_i is accurate enough then quit
13. $r_i = r_{i-1} - \alpha_i v_i$
14. $\hat{r}_i = \hat{r}_{i-1} - \alpha_i A^H \hat{p}_i$
15. **fim-para**

Teorema 3.19 *Os vetores produzidos pelo algoritmo do Gradiente Bi-Conjugado satisfazem às seguintes propriedades de ortogonalidade:*

$$\begin{aligned} (r_j, \hat{r}_i) &= 0 \qquad para \qquad i \neq j \\ (Ap_j, \hat{p}_i) &= 0 \qquad para \qquad i \neq j \end{aligned}$$

Do passo (2) ao (9), e, mais os passos (13) e (14), tem-se a geração das bases bi-ortogonais para os subespaços $\mathcal{K}$ e $\mathcal{L}$, respectivamente.

Pode-se identificar as recorrências de três termos nos passos (7) e (8), e , também, em (13) e (14).

3.6.2 Método do Gradiente Conjugado Quadrado

O método do Gradiente Conjugado Quadrado (CGS) foi apresentado por Sonneveld (SONNEVELD, 1989), é aplicado na resolução de sistemas lineares, onde a matriz principal do sistema é não-simérica e esparsa.

Pode ser classificado como método de Lanczos, uma de suas diferenças em relação ao Bi-CG é que não utiliza a matriz adjunta na multiplicação matriz-vetor.

Vejamos o algoritmo de tal método.

Algoritmo 3.8 Algoritmo do Gradiente Conjugado Quadrado

1. Seja x_0 uma solução inicial. Compute $r_0 = b - Ax_0$
2. Escolha $\hat{r}_0$ tal que $(r_0, \hat{r}_0) \neq 0$, i.e., $r_0 = \hat{r}_0$
3. $\rho_0 = 1$; $p_0 = q_0 = 0$
4. **Para** $i = 1, 2, 3, \ldots$
5. $\rho_i = (\hat{r}_0, r_{i-1})$
6. $\beta = (\frac{\rho_i}{\rho_{i-1}})$
7. $u = r_{i-1} + \beta q_{i-1}$
8. $p_i = u + \beta(q_{i-1} + \beta p_{i-1})$
9. $v = Ap_i$
10. $\alpha = \frac{\rho_i}{(\hat{r}_0, v)}$
11. $q_i = u - \alpha v$
12. $\hat{u} = u + q_i$
13. $x_i = x_{i-1} + \alpha\hat{u}$
14. se x_i "is accurate enough" então End
15. $r_i = r_{i-1} - \alpha A\hat{u}$
16. **fim-para**

No Algoritmo 3.7, foi visto que, no método do Bi-CG, o resíduo r_i pode ser obtido como

$$r_{i(Bi-CG)} = Q_i(A)r_0.$$

A recursividade para se obter r_i no CGS é desenvolvida de modo que

$$r_i = Q_i^2(A)r_0. \tag{3.48}$$

Podemos ver a prova desta afirmação em (SONNEVELD, 1989) ou (SAAD, 2003).

3.6.3 Método do Gradiente Bi-Conjugado Estabilizado

Desenvolvido por (VORST, 1992), o método do Gradiente Bi-Conjugado Estabilizado aplica-se para resolução de sistemas lineares com matrizes não-simétricas.

O Bi-CGSTAB pode ser obtido a partir do Bi-CG e, como o CGS, não usa a matriz conjugada para operações do tipo matriz-vetor.

Pode-se observar o que muda do Bi-CG para o Bi-CGSTAB. Considerando, o resíduo dado pelo médoto do Bi-CG, e o resíduo, $\hat{r}_i$ associado a A^H, tem-se que

$$\left(Q_i(A)r_0, Q_j(A^H)\hat{r}_0\right) = 0 \quad \text{para} \quad j < i, \tag{3.49}$$

pois, por hipótese no método do Bi-CG, sabemos que r_i é orotogonal a $\mathcal{L}_j$, com $j < i$,

e, $\hat{r}_j \in \mathcal{L}_j$.

A equação (3.49) pode ser reescrita da seguinte forma:

$$\left(\tilde{Q}_j(A^H)Q_i(A)r_0, \hat{r}_0\right) = 0.$$

O objetivo é, então, encontrar uma classe de polinômios que gere x_i, tais que, $r_i = \tilde{Q}_i(A)Q_i(A)r_0$.

Uma possibilidade é tomar

$$\tilde{Q}_i(x) = T_i(x) = (1-\omega_1 x)(1-\omega_2 x)\ldots(1-\omega_i x)$$

e selecionar constantes apropriadas.

No Bi-CGSTAB tem-se a seguinte relação de recorrência:

$$r_i = T_i(A)Q_i(A)r_0 \tag{3.50}$$

(para ver a prova desse fato, consulte (VORST, 1992)).

Com isso, para o algoritmo do Bi-CGSTAB não é preciso calcular A^H, nem gerar os resíduos $\hat{r}_i$.

Abaixo, encontra-se o algoritmo do método do Gradiente Bi-Conjugado Estabilizado.

Algoritmo 3.9 Algoritmo do Gradiente Bi-Conjugado Estabilizado

1. Seja x_0 uma solução inicial. Compute $r_0 = b - Ax_0$
2. Escolha $\hat{r}_0$ tal que $(r_0, \hat{r}_0) \neq 0$, i.e., $r_0 = \hat{r}_0$
3. $\rho_0 = \alpha = \omega_0 = 1$
4. $v_0 = p_0 = 0$
5. **Para** $i = 1, 2, 3 \ldots .$ **faça**
6. $\quad \rho_i = (r_0, \hat{r}_0)$; $\beta = (\frac{\rho_i}{\rho_{i-1}})(\frac{\alpha}{\omega_{i-1}})$
7. $\quad p_i = r_{i-1} + \beta(p_{i-1} - \omega_{i-1}v_{i-1})$
8. $\quad v_i = Ap_i$
9. $\quad \alpha = \frac{\rho_i}{(\hat{r}_0, v_i)}$
10. $\quad s = r_{i-1} - \alpha v_i$
11. $\quad t = As$
12. $\quad \omega_i = \frac{(t,s)}{(t,t)}$
13. $\quad x_i = x_{i-1} + \alpha p_i + \omega_i s$
14. $\quad$ se x_i "is accurate enough"**EndDo**
15. $\quad r_i = s - \omega_i t$
16. **Fim-para**

E, no algoritmo a seguir, encontra-se a sua versão pré-condicionada.

Algoritmo 3.10 Algoritmo BI-CGSTAB pré-condicionado

1. Calcule $r_0 = b - Ax^{(0)}$ a partir de um $x^{(0)}$ escolhido
2. Escolha $\tilde{r}$ ($\tilde{r} = r^{(0)}$ por exemplo)
3. **Para** $i = 1, \cdots$ **Faça**
4. $\rho_{i-1} = \tilde{r}^T r^{(i-1)}$
5. **Se** $\rho_{i-1} = 0$ o método falha
6. **Se** $i = 1$
7. $p^{(i)} = r^{(i-1)}$
8. **Senão** 9. $\beta_{i-1} = (\rho_{i-1}/\rho_{i-2})(\alpha_{i-1}/\omega_{i-1})$
10 $p^{(i)} = r^{(i-1)} + \beta_{i-1}(p^{(i-1)} - \omega_{i-1} v^{(i-1)})$
11. **fim-se**
12. Resolva $M\hat{p} = p^{(i)}$
13. $v^{(i)} = A\hat{p}$
14. $\alpha_i = \rho_{i-1}/\tilde{r}^T v^{(i)}$
15. $s = r^{(i-1)} - \alpha_i v^{(i)}$
16. Verifique a norma de s
17. **Se** $||s||$ for pequena o suficiente
18. $x^{(i)} = x^{(i-1)} + \alpha_i \hat{p}$ e Pare
21. Resolva $M\hat{s} = s$
22. $t = A\hat{s}$
23. $\omega_i = t^T s / t^T t$
24. $x^{(i)} = x^{(i-1)} + \alpha_i \hat{p} + \omega_i \hat{s}$
25. $r^{(i)} = s - \omega_i t$
26. Verifique a convergência; continue se necessário
27. Para continuar é necessário que $\omega_i \neq 0$
28. **Fim-para**

4 PRÉ-CONDICIONAMENTO

Os métodos baseados em espaço de Krylov, descritos no capítulo anterior, convergem rapidamente se a matriz dos coeficientes **A** se aproxima, de alguma forma, da matriz identidade ***I***, porém isto não é comum. A taxa de convergência de métodos iterativos depende, em muitos casos importantes, de propriedades espectrais da matriz de coeficientes, que pode até mesmo não permitir que as iterações convirjam.

Então, ao invés de resolver este problema, que é difícil, pode-se transformar tal sistema em um outro sistema que é equivalente ao original, isto é, com a mesma solução, porém com melhores propriedades espectrais ou, pensando de forma mais simples, transformando a matriz original em uma nova matriz que de alguma maneira se aproxima da matriz identidade.

O pré-condiconamento, que é o processo de transformação do sistema original em outro equivalente porém mais simples, é essencial para a grande maioria das aplicações de métodos iterativos, devendo ser utilizado para garantir a viabilidade dos métodos iterativos de Krylov aplicados aos problemas reais.

O objetivo deste capítulo é discutir alguns tipos de pré-condicionamento, como mostrado a seguir.

4.1 Pré-condicionando A*x*=*b*

A idéia do processo de pré-condicionamento é elementar. Supondo-se que se queira resolver o seguinte sistema não singular de n equações e n incógnitas

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b} \tag{4.1}$$

Para qualquer matriz não singular **M** de dimensões $n \times n$, os sistemas

$$\mathbf{M}^{-1}\mathbf{A}x = \mathbf{M}^{-1}\boldsymbol{b} \tag{4.2}$$

e

$$\mathbf{A}\mathbf{M}^{-1}\hat{\boldsymbol{x}} = \boldsymbol{b} \quad \text{com} \quad \boldsymbol{x} = \mathbf{M}^{-1}\hat{\boldsymbol{x}}. \tag{4.3}$$

têm a mesma solução que o original (4.1).

Diz-se que as equações (4.2) e (4.3) estão pré-condicionadas pela esquerda e pela direita, respectivamente.

Ao aplicar o pré-condicionamento, espera-se que a matriz **M** chamada *pré-condicionador*, seja uma boa aproximação para **A** e que obtenha-se, desta forma, as

propriedades favoráveis desejadas para a matriz deste novo sistema $\mathbf{M}^{-1}\mathbf{A}$ que é chamada de matriz pré-condicionada pela esquerda, neste caso, fazendo assim refêrencia à matriz original do sistema e à forma com que ela foi pré-condicionada.

Em alguns casos, como no Gradiente Conjugado, o pré-condicionamento deve ser feito de forma que a matriz pré-condicionada seja simétrica, então deve-se utilizar a seguinte forma de aplicação do pré-condicionador conhecida como *split*

$$\mathbf{M}_1^{-1}\mathbf{A}\mathbf{M}_2^{-1}\hat{\boldsymbol{x}} = \mathbf{M}_1^{-1}\boldsymbol{b} \quad \text{com} \quad \boldsymbol{x} = \mathbf{M}_2^{-1}\hat{\boldsymbol{x}}. \tag{4.4}$$

onde $\mathbf{M} = \mathbf{M}_1\mathbf{M}_2$. A matriz $\mathbf{M}_1$ pode ser o fator triangular inferior da decomposição de Cholesky de $\mathbf{M}$, por exemplo.

As três formas de pré-condicionamento apresentadas podem ser consideradas equivalentes, sob algum ponto de vista, pois garantem as mesmas propriedades espectrais à matriz pré-condicionada.

Definição 4.1 (HORN; JOHNSON, 1985) *Uma matriz $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ é dita ser similar a uma matriz $\boldsymbol{B} \in \mathbb{R}^{n\times n}$ se existe uma matriz $\boldsymbol{S} \in \mathbb{R}^{n\times n}$ tal que*

$$\boldsymbol{B} = \boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{S}.$$

Teorema 4.1 (HORN; JOHNSON, 1985) *Sejam $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$. Se $\boldsymbol{A}$ e $\boldsymbol{B}$ são similares, então seus polinômios característicos são os mesmos. E com isso possuem os mesmos autovalores com suas respectivas multiplicidades.*

As matrizes $\mathbf{A}\mathbf{M}^{-1}$, $\mathbf{M}^{-1}\mathbf{A}$ e $\mathbf{M}_1^{-1}\mathbf{A}\mathbf{M}_2^{-1}$ são similares e assim têm os mesmos autovalores.

Para que a idéia de pré-condicionamento seja viável deve-se ser capaz de efetuar a multiplicação $\mathbf{M}^{-1}\mathbf{A}$, $\mathbf{A}\mathbf{M}^{-1}$ ou $\mathbf{M}_1^{-1}\mathbf{A}\mathbf{M}_2^{-1}$ mesmo que de forma implícita. Estes fatores não são construídos explicitamente, pois a matriz pré-condicionada não seria esparsa e além disso, inverter $\mathbf{M}$ não costuma ser viável. Mais vantajoso que uma inversão é resolver o sistema $\mathbf{M}\boldsymbol{y} = \boldsymbol{c}$.

Assim a escolha do pré-condicionador deve ser feita de forma que

1. O sistema pré-condicionado seja fácil de ser resolvido.
2. O pré-condicionador seja fácil de se construir e aplicar.

A primeira propriedade quer dizer que, após o sistema ser pré-condicionado, os métodos devem convergir rapidamente, enquanto a segunda afirma que cada iteração

não deve ser muito cara. Na realidade, deve-se conseguir que o processo de construção e aplicação do pré-condicionador seja compensado na resolução do sistema pré-condicionado em relação ao original. Repare que os dois requisitos acima competem um com o outro.

Vejamos os casos extremos. Se $\mathbf{M} = \mathbf{A}$ então resolver $\mathbf{M}\boldsymbol{y} = \boldsymbol{c}$ é tão difícil quanto resolver $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ tornando a aplicação do pré-condicionador tão difícil quanto resolver o problema original. Se $\mathbf{M} = \mathbf{I}$ então aplicar o pré-condicionador é trivial porém a dificuldade original permanece.

Existem diversas características que podem dizer se determinado pré-condicionador é suficientemente bom, mas uma boa e simples resposta para esta pergunta é: Um pré-condicionador é bom se $\mathbf{M}^{-1}\mathbf{A}$ não está muito distante de ser normal[1] e seus autovalores são agrupados ((TREFETHEN; Bau, III, 1997)).

Abaixo, encontram-se mais algumas definições de álgebra linear que permitam avançar na análise dos pré-condicionadores. Nesse contexto estamos considerando que o corpo de escalares sobre o qual as matrizes estão operando é o corpo dos complexos.

Teorema 4.2 *Seja $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ dada. Existe um único polinômio mônico $q_{\boldsymbol{A}}(t)$ de grau mínimo tal que $q_{\boldsymbol{A}}(\boldsymbol{A}) = \boldsymbol{0}$. O grau deste polinômio é de no máximo n. Se $p(t)$ é qualquer polinômio tal que $p(\boldsymbol{A}) = 0$, então $q_{\boldsymbol{A}}(t)$ divide $p(t)$.*

Corolário 4.2.1 *Matrizes similares têm o mesmo polinômio minimal.*

Corolário 4.2.2 *Para toda matriz $\boldsymbol{A} \in \mathbb{C}^{n\times n}$, o polinômio minimal $p_{\boldsymbol{A}}(t)$ divide o polinômio característico $p_{\boldsymbol{A}}(t)$. Além disso, $q_{\boldsymbol{A}}(\lambda) = 0$ se e somente se λ é um autovalor de $\boldsymbol{A}$. Então, toda raiz de $p_{\boldsymbol{A}}(t) = 0$ é raiz de $q_{\boldsymbol{A}}(t) = 0$.*

Uma outra alternativa encontrada na literatura (BENZI; GOLUB; LIESEN, 2005) para se obter um bom pré-condicionador $\mathbf{M}$ que aproxima $\mathbf{A}$ seria obter diretamente uma matriz $\mathbf{M}$ que aproxime $\mathbf{A}^{-1}$ pois, neste caso, precisaria apenas acrescentar mais uma multiplicação ao algoritmo e não uma inversão ou fatoração. Os pré-condicionadores dessa forma são denominados “aproximantes da inversa” ou “explícitos”, que não serão tratados neste trabalho.

4.2 Diagonal

Talvez um dos mais simples seja o pré-condicionador *diagonal* ou *pré-condicionador de Jacobi* dado por $\mathbf{M} = diag(\mathbf{A})$, o que significa que, $\mathbf{M}$ é uma matriz diagonal que é

[1] Uma matriz normal comuta com a sua hermitiana, ou seja: $\mathbf{A}^H\mathbf{A} = \mathbf{A}\mathbf{A}^H$, onde $\mathbf{A}^H$ é a conjugada transposta de $\mathbf{A}$.

composta exatamente da diagonal da matriz **A**, para **A** não singular, assim

$$m_{ij} = \begin{cases} a_{ij} & \text{se } i = j \\ 0 & \text{caso contrário} \end{cases}$$

representando um simples escalamento onde $a_{i,i} \neq 0,\ \forall i$.

Para alguns problemas esta transformação é suficiente para tornar rápida uma convergência lenta.

Pré-condicionadores de Jacobi exigem muito pouco armazenamento e são fáceis de implementar mesmo em paralelo. Por outro lado, em geral, pré-condicionadores mais sofisticados e caros apresentam melhores resultados.

4.3 Fatorações incompletas

O processo de eliminação Gaussiana é equivalente a fatoração da matriz **A** na forma **PA** $=$ **LU**, onde **L** é triangular inferior com diagonal unitária e **U** é triangular superior e **P** é uma matriz de permutação. Para efeito de simplificação, neste texto, **P**, só será usada em casos necessários. Na construção desses fatores, ao trabalhar-se com matrizes esparsas, eles podem se tornar menos esparsos do que **A**, o que torna este processo computacionalmente muito caro. Em muitos casos não é necessária a fatoração completa da matriz, apenas uma fatoração aproximada é suficiente; nesse quadro surgem as fatorações incompletas (GOLUB; LOAN, 1996; SAAD, 2003). Diz-se que uma fatoração é incompleta se durante o processo de fatoração alguns elementos não nulos dos fatores são anulados.

Mesmo assim, uma fatoração incompleta pode ser computacionalmente muito cara. Uma das utilizações dessa fatoração ocorre quando devem-se resolver muitos sistemas com a mesma matriz **A** e vetor do lado direito variável, assim aplica-se os fatores a muitos sistemas com custo de apenas uma fatoração.

4.3.1 ILU(0), ILUT, MILU

Ao aproximar a fatoração de uma matriz na forma citada acima, isto é, **A** $=$ **LU** , utiliza-se como forma de construção da fatoração *ILU* (*Incomplete LU*) uma modificação na eliminação Gaussiana permitindo preenchimento em apenas um conjunto de posições nos fatores LU.

Suas posições, em que são permitidos preenchimentos são dadas em função de um conjunto de índices **S** pré-definido:

$$l_{ij} = 0 \quad \text{se} \quad j > i \quad \text{ou} \quad (i,j) \notin \mathbf{S}$$
$$u_{ij} = 0 \quad \text{se} \quad i > j \quad \text{ou} \quad (i,j) \notin \mathbf{S}.$$

Ao construir uma fatoração LU incompleta, estamos construindo fatores $\tilde{\mathbf{L}}$ e $\tilde{\mathbf{U}}$ que aproximam os fatores **L** e **U**, isto é, $\tilde{\mathbf{L}} \approx \mathbf{L}$ e $\tilde{\mathbf{U}} \approx \mathbf{U}$, ou $\mathbf{A} = \mathbf{LU} - \mathbf{R}$ onde **R** é o resíduo ou erro da fatoração. Esta fatoração incompleta de **A** representa uma fatoração completa de $\tilde{\mathbf{A}}$, quer dizer, $\tilde{\mathbf{A}} = \tilde{\mathbf{L}}\tilde{\mathbf{U}}$ onde o pré-condicionador **M** é dado por $\mathbf{M} = \tilde{\mathbf{L}}\tilde{\mathbf{U}}$. De forma que não reste dúvidas, os fatores **L** e **U** representarão os fatores aproximados sempre que necessário.

Para que **M** seja um bom pré-condicionador, é necessário que se aproxime de **A** através de alguma medida. Uma estratégia típica é que seja exigido que as entradas **M** coincidam com as de **A** no conjunto **S**:

$$m_{ij} = a_{ij} \quad \text{se} \quad (i,j) \in \mathbf{S}. \tag{4.5}$$

Uma estratégia comumente utilizada para definir o conjunto **S** é:

$$\mathbf{S} = \{(i,j) | a_{ij} \neq 0\}.$$

Isto quer dizer que os únicos elementos diferentes de zero permitidos nos fatores LU são aqueles que correspondem aos elementos diferentes de zero de **A**. Este tipo de fatoração incompleta é chamada de ILU(0): fatoração LU incompleta de nível zero. Esta fatoração é simples e tem baixo custo computacional, mas pode acarretar em muitos casos em baixa efetividade na aceleração da convergência dos métodos iterativos.

Para superar este problema, diversas outras opções de fatorações incompletas foram desenvolvidas para permitir maior quantidade de preenchimento e assim prover uma melhoria na aproximação de forma a tornar o pré-condicionador mais eficiente, porém tornando o processo de fatoração mais custoso computacionalmente.

O conjunto **S** apresentado anteriormente representa uma alternativa pertencente a um padrão estático, onde as posições que serão anuladas nos fatores são previamente definidas. Neste caso, é muito importante que não anule-se nenhum elemento da diagonal para não tornar a fatoração instável.

Fatorações que se baseiam na estrutura da matriz perdem muita informação ao não considerar os valores numéricos dos elementos. Existem alguns métodos disponíveis que descartam elementos baseados em suas magnitudes. Este tipo de decomposição incompleta baseia-se em um padrão dinâmico.

A fatoração *LU incompleta com limitante (threshold)* ou *ILUT* baseia-se em

um padrão dinâmico cuja característica é anular dos fatores os elementos que, em módulo, são menores que um determinado parâmetro limite τ pré-estabelecido, obtendo uma melhor aproximação com custo computacional mais elevado. Este critério pode funcionar mal caso a matriz apresente problemas de escalabilidade, neste caso é melhor utilizar um parâmetro τ relativo. Por exemplo, quando for eliminar a linha i, um novo preenchimento é aceitável somente se, em valor absoluto, é maior do que $\tau||\boldsymbol{a}_{i*}||_2$, onde $\boldsymbol{a}_{i*}$ denota a i-ésima linha de **A**. Outros critérios podem ser utilizados. O grande problema é conseguir determinar o valor ideal para parâmetro τ pois varia de um problema para outro.

Também é possível enfrentar problemas de armazenamento quando não podemos prever o quanto pode ser preenchido por este critério. Pode-se então utilizar uma estratégia de *limitante duplo* (SAAD, 2003; BENZI; GOLUB; LIESEN, 2005): fixa-se uma tolerância τ e um valor inteiro positivo *p* que representa quantas entradas não nulas são permitidas em cada linha, que serão aquelas de maior magnitude, dos fatores **L** e **U**.

Em todas as técnicas apresentadas, os elementos que devem ser retirados dos fatores são simplesmente descartados. Porém existem técnicas que tentam compensar de alguma forma os elementos eliminados. Como por exemplo, uma estratégia popular é retirar de cada elemento da diagonal de **U** a soma dos elementos de cada linha respectivamente. Esta técnica que visa uma compensação diagonal origina o chamado *Modified ILU* ou *MILU*. De forma mais simples, pode-se dizer que a condição (4.5) é removida para $m_{ii} = a_{ii}$ e uma nova condição para a soma das linhas é adicionada. Assim pode-se substituir (4.5) por

$$\sum_{j=1}^{n} m_{ij} = \sum_{j=1}^{n} a_{ij} \quad \forall i \quad \text{e} \quad m_{ij} = a_{ij} \quad \text{se} \quad i \neq j \quad \text{e} \quad (i,j) \in \mathbf{S}. \tag{4.6}$$

Denotemos a i-ésima linha de **R** por $\boldsymbol{r}_{i*}$. Assim, a modificação proposta apresenta

$$u_{ii} := u_{ii} - (\boldsymbol{r}_{i*}\boldsymbol{e})$$

onde $\boldsymbol{e} \equiv (1, 1, \ldots, 1)^T$. Note que $\boldsymbol{r}_{i*}\boldsymbol{e}$ é a soma dos elementos da linha $\boldsymbol{r}_{i*}$. A expressão acima pode ser reescrita como $\boldsymbol{u}_{i*} := \boldsymbol{u}_{i*} - (\boldsymbol{r}_{i*}\boldsymbol{e})\boldsymbol{e}_i^T$ e assim pode-se escrever

$$\boldsymbol{a}_{i*} = \sum_{k=1}^{i} l_{ik}\boldsymbol{u}_{k*} + (\boldsymbol{r}_{i*}\boldsymbol{e})\boldsymbol{e}_i^T - \boldsymbol{r}_{i*}$$

Se multiplicarmos à esquerda por e, obteremos

$$\boldsymbol{a}_{i*}\boldsymbol{e} = \sum_{k=1}^{i-1} l_{ik}\boldsymbol{u}_{k*}\boldsymbol{e} = \mathbf{LU}\boldsymbol{e}.$$

Isto garante que a soma das linhas de **A** é igual a soma das linhas de LU. Este processo pode ser utilizado para qualquer fatoração LU incompleta.

4.3.2 ICC

Matrizes simétricas e positivas-definidas (**spd**) podem ser decompostas em fatores triangulares mais rapidamente do que matrizes sem essas propriedades. O algoritmo padrão para esta operação é a *fatoração de Cholesky*, que é uma variante da eliminação gaussiana que opera em ambos os lados, esquerdo e direito, da matriz de uma vez, preservando e explorando sua simetria.

Relembrando, uma matriz $\mathbf{A} \in \mathbb{R}^{n\times n}$ é simétrica se $\mathbf{A} = \mathbf{A}^T$, e é positiva definida se $\boldsymbol{x}^T\mathbf{A}\boldsymbol{x} > 0$.

Suponha que **A** seja uma matriz **spd** e $\mathbf{X} \in \mathbb{R}^{m\times n}$ de posto completo com $m \geq n$, então a matriz $\mathbf{X}^T\mathbf{AX}$ também é simétrica e positiva definida. É simétrica porque $(\mathbf{X}^T\mathbf{AX})^T = \mathbf{X}^T\mathbf{A}^T\mathbf{X} = \mathbf{X}^T\mathbf{AX}$, e positiva definida porque, para qualquer vetor $x \neq 0$, temos $\mathbf{X}\boldsymbol{x} \neq 0$ e então $\boldsymbol{x}^T(\mathbf{X}^T\mathbf{AX})\boldsymbol{x} = (\mathbf{X}\boldsymbol{x})^T\mathbf{A}(\mathbf{X}\boldsymbol{x}) > 0$.

Além disso, todos os autovalores de uma matriz simétrica positiva definida são reais e positivos pois, se $\mathbf{A}\boldsymbol{x} = \lambda\boldsymbol{x}$ para $\boldsymbol{x} \neq 0$, então $\boldsymbol{x}^T\mathbf{A}\boldsymbol{x} = \lambda\boldsymbol{x}^T\boldsymbol{x} > 0$ de onde pode-se concluir que $\lambda > 0$. Por outro lado, se uma matriz simétrica e tem todos os autovalores positivos, podemos concluir que ela é positiva definida.

Ao aplicar uma simples transformação linear pela esquerda da matriz a ser fatorada **A**, visamos anular todos os elementos abaixo da primeira entrada da diagonal. Pela simetria da matriz, este mesmo procedimento pode ser feito pela direita para anular todos os elementos a direita da primeira entrada da diagonal, onde a matriz que representa a transformação linear pela direita é a transposta daquela aplica a esquerda. Ao proceder desta forma para o restante dos elementos, estamos transformando a matriz **A** em uma matriz identidade e obtemos

$$\mathbf{A} = \underbrace{\mathbf{C}_1^T\mathbf{C}_2^T\ldots\mathbf{C}_m^T}_{\mathbf{C}^T}\underbrace{\mathbf{C}_m^T\ldots\mathbf{C}_2^T\mathbf{C}_1^T}_{\mathbf{C}},$$

isto é

$$\mathbf{A} = \mathbf{C}^T\mathbf{C}.$$

Uma redução deste tipo é conhecida como *fatoração de Cholesky*.

Quando da implementação desta fatoração, somente metade da matriz a ser fatorada precisa ser representada explicitamente e assim somente metade das operações precisa ser feita. Ao final da fatoração só é necessário armazenar um fator, já que o outro é facilmente calculado. Esta fatoração, como era de se esperar, é realizada com apenas metade das operações necessárias para a decomposição LU. E sua forma incompleta é construída da mesma forma que a apresentada para a decomposição LU, passando a se chamar *decomposição de Cholesky incompleta*, representada por *ICC*.

5 IMPLEMENTAÇÃO DOS MÉTODOS ITERATIVOS PRÉ-CONDICIONADOS

Serão discutidas, neste capítulo, formas de implementação dos métodos iterativos vistos anteriormente, em conjunto com os pré-condicionadores. Assim, na seção 5.1, será comentada a importâcia da Programação Orientada a Objeto. Nas seções 5.2 e 5.3, serão vistos dois pacotes orientados a objeto que utilizam as técnicas aqui já vistas para resolução de sistemas lineares. E, finalmente, na seção 5.4, serão discutidas cinco abordagens para as subrotinas básicas de Álgebra Linear (BLAS, isto é, *Basic Linear Algebra Subprograms*).

5.1 Programação Orientada a Objeto (POO)

É muito comum, atualmente, se perguntar o porquê de se trabalhar utilizando-se a programação orientada a objeto em computação científica (SHAPIRA, 2006). Em geral, os conceitos de *classes* e de *objetos* (além de outros) são vistos como uma grande ferramenta que facilita a utilização de um mesmo código em vários programas diferentes.

Pode-se, de forma simples, definir uma *classe* como sendo um “tipo” (como `int` na linguagem C, por exemplo), porém mais elaborado, pois, dentro da classe, é possível encontrar variáveis de outros tipos (*atributos*) e funções que modificam seus atributos e os atributos de outras classes (*métodos*). Neste contexto, o *objeto* seria uma ocorrência (*instância*) da classe dentro de um programa que a utiliza. Como exemplo, assume-se que um programador queira implementar um códido onde se tenha a descrição de um ponto no plano cartesiano. Dentro de uma linguagem orientada a objeto (como o C++, por exemplo), este programador poderia escrever uma classe chamada *ponto*, com todas as características de um ponto no plano, que poderá ser usada por ele e por qualquer outro programador que necessite de algo similar. O algoritmo desta classe poderia ser escrito asim:

```
class ponto
{
private:
        /* Atributos */

        double x;
        double y;
public:
        /* Métodos */

        double X() const
        {
                return x
        }
        double Y() const
        {
                return y
        }
        void zero()
        {
                x = y = 0.;
        }
};
```

Já o programa que utilizaria esta classe, poderia ser:

```
 int main()
{
        /* declaração de um objeto da classe ponto */
        ponto a;

        /* atribuindo 0. às coordenadas do ponto */
        a.zero();

        return 0;
}
```

Desta forma, caso o programador da classe queira alterá-la, os programadores que a utilizam não precisam alterar os seus respectivos códigos. Ou seja, supondo que a classe `ponto` receba mais um método, `um()` por exemplo, o acréscimo deste método

não alterará a estrutura da função `main()`. Por estas e outras facilidades, a orientação a objeto vem sendo cada vez utilizada na computação científica. Nas seções a seguir, serão vistos alguns exemplos deste uso.

Existem outros conceitos ligados a orientação a objeto, tais como, *herança*, *encapsulamento*, *sobrecarga de operadores*, etc, que não serão abordados neste trabalho.

5.2 IML++ e Sparselib++

O IML++, isto é, *Iterative Methods Library* (POZO; REMINGTON, 1996), é uma coleção de algoritmos desenvolvidos em C++ no NIST (National Institute of Standards and Technology) para resolver sistemas lineares simétricos e não simétricos usando técnicas iterativas.

Neste pacote, é possível perceber a flexibilidade da programação orientada a objeto com o uso de *templates* na implementação dos métodos iterativos, como mostra o algoritmo abaixo:

```
template <class Matrix, class Vector,
 class Preconditioner, class Real>

int CG(const Matrix &A, Vector &x, const Vector &b,
const Preconditioner &M, int &max_iter, Real &tol)
{
        Real resid;
        Vector p, z, q;
        Vector alpha(1), beta(1), rho(1), rho_1(1);

        Real normb = norm(b);
        Vector r = b - A*x;

        if (normb == 0.0)
        normb = 1;

        if ((resid = norm(r) / normb) <= tol) {
                tol = resid;
                max_iter = 0;
                return 0;
         }
```

```
        for (int i = 1; i <= max_iter; i++) {
        z = M.solve(r);
        rho(0) = dot(r, z);

        if (i == 1)
                p = z;
        else {
                beta(0) = rho(0) / rho_1(0);
                p = z + beta(0) * p;
        }

         q = A*p;
        alpha(0) = rho(0) / dot(p, q);

        x += alpha(0) * p;
        r -= alpha(0) * q;

        if ((resid = norm(r) / normb) <= tol) {
                tol = resid;
                max_iter = i;
                return 0;
        }

        rho_1(0) = rho(0);
        }

        tol = resid;
        return 1;
}
```

É importante notar as semelhanças e diferenças do algoritmo acima com o algoritmo 3.1. Nota-se que, na primeira e segunda linha, existe uma declaração (*template*) que permite o uso de qualquer pré-condicionador do Sparselib, ou seja, não foi necessário aqui criar um algoritmo para cada tipo de pré-condicionador.

Além do método gradiente conjugado pré-condicionado (visto no algoritmo de exemplo acima), o IML++ apresenta os seguintes métodos:

- Método de Richardson;
- Método de Chebyshev;

- Gradiente Conjugado Quadrado;
- Gradiente Bi-Conjugado;
- Gradiente Bi-Conjugado Estabilizado;
- Resíduo Mínimo Generalizado
- Resíduo Quase-Mínimo.

O IML++ é geralmente utilizado com outro pacote, também desenvolvido pelo NIST, que implementa os pré-condicionadores: o *Sparselib++*. Neste pacote, podem ser encontrados os seguintes pré-condicionadores:

- Diagonal;
- LU Incompleto (Fill-in 0);
- Cholesky Incompleto (Fill-in 0).

Um ponto que deve ser observado no IML++ e no Sparselib++, e que talvez seja a dirença entre estes pacotes e os pacotes da biblioteca da seção 5.3, é que eles utilizam o BLAS esparso desenvolvido pelo mesmo grupo.

Um outro ponto é o fato do Sparselib++ não retornar a matriz de permutação no momento da aplicação dos pré-condicionadores, o que pode acarretar o aumento no número de iterações e, consequentemente, queda no desempenho do programa, quando se trata de matrizes mal-condicionadas e que precisam de reordenamento.

5.3 Trilinos

O Trilinos (HEROUX et al., 2003) é um projeto desenvolvido nos Laboratórios Sandia (Novo México e Califórnia), dirigidos pela Sandia Corporation e pela Lockheed Martin Company, para o Departamento de Energia dos Estados Unidos. O objetivo do projeto é facilitar o desenvolvimento de algoritmos voltados a problemas científicos, tais como: resolução de sistemas de equações lineares e não lineares, implementação paralela de algoritmos de álgebra linear, etc. Trata-se de um conjunto de pacotes escritos principalmente em C++ que utilizam técnica de orientação a objeto. Dentre seus pacotes, encontra-se o *Epetra* (HEROUX et al., 2003) que é um conjunto de classes escritas em C++ elaboradas para dar suporte a aplicações de Álgebra Linear e a outros pacotes do Trilinos. Neste pacote, podemos encontrar formatos de matrizes, vetores e operações básicas, como produto interno e produto de matrizes. Abaixo, serão vistas as principais classes do Epetra:

5.3.1 Principais Classes Seriais e Paralelas do Epetra

- *Epetra_Comm:* Contém informações específicas sobre as máquinas paralelas que estão sendo utilizadas. Atualmente suporta o modelos de programação serial (Epetra_SerialComm), MPI (Epetra_MPIComm) e protótipos híbridos de programação paralela com MPI e Threads.

- *Classes de Mapeamento de Elementos:* Epetra_Map, Epetra_LocalMap, Epetra_BlockMap - contêm informações que sobre a distribuição dos elementos dos vetores, matrizes e outros objetos numa máquina paralela ou serial.

- *Classes de Vetores:* Epetra_Vector - vetores reais de precisão dupla. Suportam a construção e o uso de vetores em máquina paralela. Epetra_FEVector é a especialização do Epetra_Vector indicado para suportar a construção de vetores oriundos de aplicações de elementos finitos. Um Epetra_Vector é um Epetra_MultiVector. Desta forma, qualquer argumento requerido em um Epetra_MultiVector pode ser aceito em um Epetra_Vector. Um Epetra_MultiVector é a generalização de um array 2D.

- *Grafos Esparsos por Linha:* Epetra_CrsGraph - Permite a construção de um grafo, paralelo ou serial. O Grafo determina uma comunicação entre os objetos de matrizes. Toda matriz esparsa do Epetra (Epetra_CrsMatrix, Epetra_VbrMatrix, Epetra_FECrsMatrix e Epetra_FEVbrMatrix) tem um objeto Epetra_CrsGraph. Este grafo pode ser ou construído implicitamente pelo usuário no momento da construção da matriz, ou passado como parâmetro para o método construtor pelo o usuário. No último caso, o grafo pode ser ou construído pelo usuário ou extraído de uma matriz esparsa já existente através do método `Graph()`.

- *Classe Virtual de Matriz Linha:* Epetra_RowMatrix - classe virtual que especifica a interface necessária a maioria das operações requeridas por matrizes linha. A classe Epetra_LinearProblem, quando utilizada, espera uma classe Epetra_RowMatrix. As classes Epetra_CrsMatrix, Epetra_VbrMatrix, Epetra_FECrsMatrix e Epetra_FEVbrMatrix implementam uma interface Epetra_RowMatrix e, assim, objetos destas classes podem ser utilizados com a Epetra_LinearProblem.

- *Armazenamento Compactado por Linha:* Epetra_CrsMatrix - classe para matrizes esparsas compactadas por linha, suporte a construção e uso, com rotinas de otimização de armazenamento, tais como `FillComplete()` e `OptimizeStorage()`. Epetra_FECrsMatrix - classe especializada para dar suporte a aplicações de elementos finitos.

- *Armazenamento em Blocos Compactado por Linha:* Epetra_VbrMatrix - classe para matrizes esparsas armazenadas por blocos. Epetra_FEVbrMatrix - classe especializada para aplicações de elementos finitos.

Em (SALA; HEROUX; DAY, 2004), podem ser encontradas mais classes Epetra, além destas citada aqui. É importante observar que o Trilinos utiliza para cálculos básicos de Álgebra Linear, tais como produto matriz-vetor, produto matriz-matriz, etc, um conjunto de subrotinas básicas de Álgebra Linear (BLAS), fornecidas pelo usuário. O Trilinos fornece interfaces para a chamada das rotinas do BLAS que serão descritas na seção a seguir. Além, disso fornecem interfaces para solucionadores diretos, reordenadores, pré-condicionadores externos, permitindo flexibilidade de uso.

5.4 BLAS

As subrotinas BLAS (Basic Linear Algebra Subprograms (BLAST Forum, 2001)) são rotinas originalmente escritas em Fortran 77, mas que hoje estão disponíveis nas principais linguagens e adaptadas às principais arquiteturas e compiladores existentes e que oferecem meios para que operações básicas com matrizes e vetores sejam executadas. Este conjunto de subrotinas está dividido em:

- BLAS Nível 1 - executa operações escalar-vetor e vetor-vetor (LAWSON et al., 1979);
- BLAS Nível 2 - executa operações matriz-vetor (DONGARRA et al., 1988);
- BLAS Nível 3 - executa operação matriz-matriz (DONGARRA et al., 1990).

Levando-se em consideração que uma matriz esparsa pode ser divida em blocos que, dependendo do tamanho, podem ter a maior parte de seus elementos não-nulos, esta seção se dedica a mostrar teoricamente cinco distribuições de BLAS (ACML, ATLAS, BLAS Original, MKL e GotoBLAS) voltadas ao tratamento de problemas densos. É de extrema importância saber que o Trilinos (descrito na seção 5.3) pode ser compilado para utilizar estas cinco abordagens de BLAS, ou qualquer outra implantação de BLAS disponível.

5.4.1 ACML

A ACML (AMD Core Math Library) é uma biblioteca desenvolvida pela AMD® que consiste dos seguintes itens:

- A plena implementação do Nível 1, 2 e 3 Basic Linear Algebra Subroutines (BLAS), com as principais rotinas otimizadas para alta performance em processadores AMD Opteron [1].
- Um pacote completo de Álgebra Linear (LAPACK) rotinas.
- Transformadas Rápidas de Fourier (FFTs), precisão simples, dupla e dados complexos.
- Geradores de números aleatórios em precisão simples e dupla.
- Suporte a OpenMP.

5.4.2 MKL

A MKL (Math Kernel Library) é a biblioteca desenvolvida pela Intel® que apresenta os seguintes recursos:

- BLAS, LAPACK, ScaLAPACK e rotinas auxiliares.
- Solvers Iterativos e Diretos.
- Suporte a Multi-Thread e outras.

5.4.3 GotoBLAS

Durante a última década, alguns projetos de grupos de pesquisa no mundo todo têm perseguido alta performance no que diz respeito a operações básicas de Álgebra Linear. Tipicamente, estes projetos vêm organizando a computação em torno de um "núcleo interno", $C = A^T B + C$, que guarda um dos operadores na cache L1, enquanto mantêm parte dos outros operadores fora desta cache. Algumas variações incluem abordagens que utilizam este princípio para vários níveis de cache ou que aplicam o mesmo princípio à cache L2, enquanto ignoram a L1. O objetivo principal destas abordagens é amenizar o custo de se mover dados através dos níveis de memória.

O GotoBlas (GOTO; GEIJN, 2002) é uma biblioteca desenvolvida pelo Centro de Computação Avançada do Texas, Universidade de Austin que apresenta uma abordagem diferente, observando que, para a geração atual de arquiteturas de computadores, parte dos custos com a computação deste núcleo interno está associada

[1] Neste trabalho, as distribuições de BLAS não serão testadas em processadores AMD Opteron, por não haver disponibilidade destes no laboratório de teste.

às perdas na TLB (*Translation Look-aside Buffer Table*). Embora seja levada em consideração à importância das caches, o fato de maior importância nesta abordagem é a minimização dessas perdas na TLB, como será visto a seguir.

5.4.3.1 Estratégia para reduzir perdas de TLB

Considerando-se as seguintes matrizes

$$C = \left[\begin{array}{c|c|c} C_{11} & \dots & C_{1N} \\ \hline \vdots & & \vdots \\ \hline C_{M1} & \dots & C_{MN} \end{array}\right], \qquad A = \left[\begin{array}{c|c|c} A_{11} & \dots & A_{1K} \\ \hline \vdots & & \vdots \\ \hline A_{M1} & \dots & A_{MK} \end{array}\right] \qquad \text{e}$$

$$B = \left[\begin{array}{c|c|c} B_{11} & \dots & B_{1N} \\ \hline \vdots & & \vdots \\ \hline B_{K1} & \dots & B_{KN} \end{array}\right],$$

para se fazer

$$C = AB + C,$$

temos os seguinte somatório

$$C_{ij} = \sum_{p=1}^{K} A_{ip} B_{pj} + C_{ij},$$

implementado, pelo seguinte algoritmo

Algoritmo 5.1 Implementação de $C = AB + C$

```
1.  Para i = 1:M
2.    Para p = 1:K
3.      Para j = 1:N
3.        C_ij = A_ip B_pj + C_ij
4.      fim-para
8.  fim-para
```

Para começar a otimizar a multiplicação de matrizes, começaremos destacando o núcleo interno

$$C_{ij} = A_{ip} B_{pj} + C_{ij},$$

e partiremos do seguinte princípio:

Haverá um aumento de performance quando A_{ip} for mantida na cache L1 enquanto os elementos de C_{ij} e B_{pj} são mantidos num nível mais baixo da pirâmide,

considerando a Figura 7. A dimensão de Aip é otimizada de tal forma que o núcleo interno tenha a melhor performance possível.

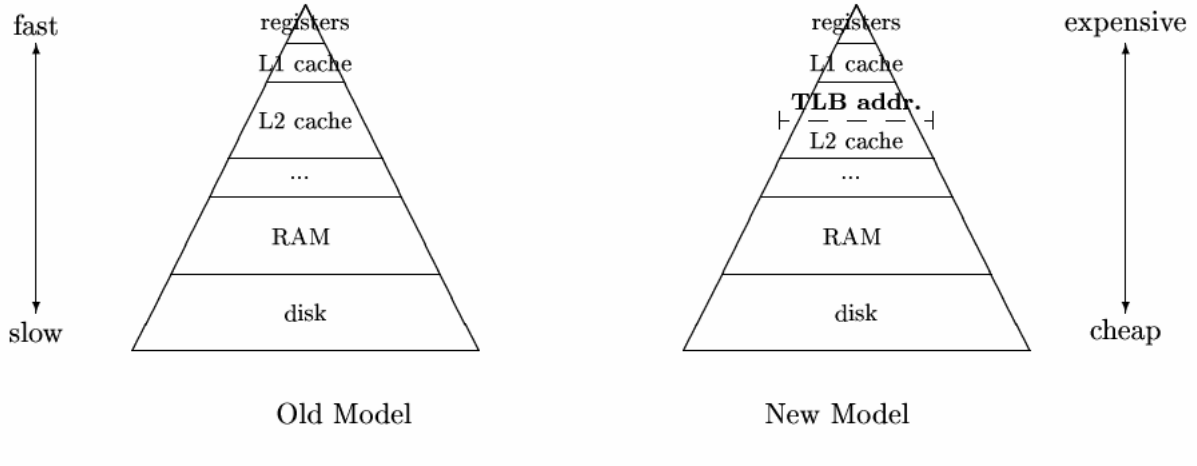


Figura 7: Considerações básicas sobre arquitetura.

Nesta abordagem, é aconselhável armazenar A_{ip} continuamente na memória, principalmente se esta matriz apresenta um grande número de linhas ou colunas, para que haja redução nas perdas de TLB.

Também é aconselhável **transpor** A_{ip} para que o acesso à memória seja contínuo no momento do cálculo do produto interno das colunas de A_{ip}^T e B_{pj} , quando C_{ij} for atualizado.

Reescrevendo o algoritmo 5.1 na forma do algoritmo abaixo,

Algoritmo 5.2 Reescrevendo o Algoritmo 5.1

1. **Para** i = 1:M
2. **Para** p = 1:K
3. $\left(\; C_{i1} \; \middle| \; \cdots \; \middle| \; C_{iN} \; \right) = A_{ip} \left(\; B_{p1} \; \middle| \; \cdots \; \middle| \; B_{pN} \; \right) + \left(\; C_{i1} \; \middle| \; \cdots \; \middle| \; C_{iN} \; \right)$
4. **fim-para**
8. **fim-para**

podemos fazer algumas observações:

- Se pudermos otimizar a linha

$$\left(\; C_{i1} \; \middle| \; \cdots \; \middle| \; C_{iN} \; \right) = A_{ip} \left(\; B_{p1} \; \middle| \; \cdots \; \middle| \; B_{pN} \; \right) + \left(\; C_{i1} \; \middle| \; \cdots \; \middle| \; C_{iN} \; \right)$$

estaremos num bom caminho.

- Para otimizar a linha acima, é aconselhável fazer a transposição $\hat{A} = A_{ip}^T$ e executar

$$\left(\; C_{i1} \; \middle| \; \cdots \; \middle| \; C_{iN} \; \right) = \hat{A}^T \left(\; B_{p1} \; \middle| \; \cdots \; \middle| \; B_{pN} \; \right) + \left(\; C_{i1} \; \middle| \; \cdots \; \middle| \; C_{iN} \; \right).$$

  Isto possibilita o acesso à memória de forma contínua enquanto o produto interno entre as colunas de A_{ip} e B_{pj} é executado. Isto também evita lixo na cache L1.

- É importante completar um loop através de todas as entradas de $\hat{A}$ sem criar saltos na corrente de dados e na computação. Para que isto seja satisfeito, $\hat{A}$ deve ser armazenado continuamente em memória, assegurando-se que o acesso de $\hat{A}$ não criará perdas de TLB.

- Uma perda considerável vem do custo de acessar C_{ij} e B_{pj} pela primeira vez como parte da computação de $C_{ij} = \hat{A}^T B_{pj} + C_{ij}$. Vamos assumir que este custo inclui um custo de inicialização (latência) e um custo proporcional ao tamanho de B_{pj}. Grande parte do custo de latência vem do custo com perdas de TLB associado ao acesso a C_{ij} e B_{pj} pela primeira vez. Escolhendo B_{pj} e C_{ij} de forma que a dimensão das linhas seja relativamente grande, este custo sobre a inicialização de muitos elementos de C_{ij} e B_{pj} é amenizado.

- Se o fluxo de dados se dá de tal forma que a CPU não pare, então o que pode reduzir performance é a transposição *(packing)* de A_{ip} em $\hat{A}$.

  Concluindo, temos:

- $\hat{A}$ deve ser relativamente quadrada e deve estar em sua grande parte na cache L2;

- As submatrizes C_{ij} e B_{pj} devem ser relativamente pequenas para que se tenha poucas entradas na TLB relativas a estas submatrizes.

5.4.3.2 Uma Abordagem Prática

Nesta abordagem, consideraremos a geração atual de microprocessadores, tais como o Intel Pentium ® 4. Podemos observar que neste tipo de arquitetura a largura de banda entre a cache L2 e o Registrador é tal que, dentro do intervalo de tempo que leva para carregar um número de ponto flutuante da L2 para o Registrador, apenas algumas operações de ponto flutuante (freqüentemente, apenas uma) são executadas. Assim, podemos assumir que a razão entre o custo de se carregar um número de ponto flutuante da L2 para o Registrador e a execução de operações de

ponto flutuante no mesmo intervalo de tempo é igual a 1. Desta forma, a abordagem a seguir melhoraria a performance:

1. Particionar A, B e C, como foi feito anteriormente, porém escolhendo C_{ij} e B_{pj} tenham apenas uma coluna:

$$C = \left[\begin{array}{c|c|c} c_{11} & \dots & c_{1n} \\ \hline \vdots & & \vdots \\ \hline c_{M1} & \dots & c_{Mn} \end{array}\right], \qquad A = \left[\begin{array}{c|c|c} A_{11} & \dots & A_{1K} \\ \hline \vdots & & \vdots \\ \hline A_{M1} & \dots & A_{MK} \end{array}\right] \qquad \text{e}$$

$$B = \left[\begin{array}{c|c|c} b_{11} & \dots & b_{1n} \\ \hline \vdots & & \vdots \\ \hline b_{K1} & \dots & b_{Kn} \end{array}\right],$$

   Note que os elementos de c_{ij} e b_{pj} serão contínuos na memória.

2. Considerando o núcleo de computação

$$\left(\begin{array}{c|c|c} c_{i1} & \cdots & c_{in} \end{array} \right) = A_{ip} \left(\begin{array}{c|c|c} b_{p1} & \cdots & b_{pn} \end{array} \right) + \left(\begin{array}{c|c|c} c_{i1} & \cdots & c_{in} \end{array} \right)$$

   Implementando-o com a transposição $\hat{A} = A_{ip}^T$, temos que

$$\left(\begin{array}{c|c|c} c_{i1} & \cdots & c_{in} \end{array} \right) = \hat{A}^T \left(\begin{array}{c|c|c} b_{p1} & \cdots & b_{pn} \end{array} \right) + \left(\begin{array}{c|c|c} c_{i1} & \cdots & c_{in} \end{array} \right) \tag{5.1}$$

3. Se:

   (a) $\hat{A}$ é armazenada de forma contínua na memória;

   (b) A transposição $\hat{A} = A_{ip}^T$ é cuidadosamente ordenada;

   (c) O primeiro elemento de $\hat{A}$ é alinhado em uma página;

   (d) $\hat{A}$ e, para todo j, c_{ij}, $c_{i(j+1)}$, b_{pj}, $b_{p(j+1)}$, juntas, não podem ultrapassar a capacidade da TLB;

   (e) $\hat{A}$ cabe na L2;

   (f) Eq. 5.1 é executada pelo loop
   for i = 1:n
   $c_{ij} = \hat{A}^T b_{pj} + c_{ij}$
   end
   Então, a princípio,

      i. $\hat{A}$ será carregada na L2 e na TLB durante a transposição $\hat{A} = A_{ip}^T$;

ii. As páginas correspondentes a Â carregadas na L2 e na TLB serão mantidas até o término da execução de Eq. 5.1;

iii. O fluxo de dados deve permitir a computação de cada $\hat{A}^T b_{pj} + c_{ij}$ para que seja atingida a melhor performance.

5.4.4 ATLAS

O ATLAS, isto é, Automatically Tuned Linear Algebra Software (WHALEY; PETITET; DONGARRA, 2001), é um projeto de pesquisa cujo objetivo principal é aplicar técnicas empíricas para garantir portabilidade a Rotinas de Álgebra Linear em termos de performance.

Em Álgebra Linear, há muitas operações otimizáveis, de tal forma que um código otimizado pode executar determinada tarefa muito mais rápido que um código simples. Porém, vários tipos de otimização são direcionadas para uma plataforma específica, isto é, o código pode ser executado muito bem em uma dada arquitetura e não funcionar devidamente bem em outra. O método tradicional para se resolver este problema é produzir rotinas otimizadas para cada tipo de máquina. Este processo é trabalhoso e requer meses de treinamento direcionado, tanto na área de Álgebra Linear quanto na área de otimização computacional.

O método tradicional de otimização pode se tornar pouco eficiente se forem considerados os seguintes fatores:

- A velocidade com que as máquinas estão evoluindo nos últimos anos;
- A existência de várias camadas de software (sistema operacional, compiladores, etc) que também interferem no desempenho do software em questão.

5.4.4.1 AEOS

Nos tempos modernos da computação, um novo paradigma se faz necessário no que se refere à produção de rotinas de alta performance. Este paradigma é conhecido como *AEOS* (WHALEY; PETITET; DONGARRA, 2001), isto é, *Automated Empirical Optimization of Software*.

Em pacotes baseados na AEOS, como o ATLAS por exemplo, são oferecidos vários métodos de se fazer as operações requisitadas e, a partir destas requisições, é feita empiracamente uma análise de tempo para se decidir qual dos métodos oferecidos é o melhor para determinado tipo de arquitetura.

Neste contexto, o ATLAS usa geradores de código (i.e., programas que escre-

vem programas) para oferecer diferentes métodos de se resolver uma dada operação e, além disto, apresenta sofisticados scripts de busca e mecanismos robustos de medição de tempo para se encontrar o melhor método para uma determinada arquitetura.

A dificuldade em se otimizar rotinas básicas de Álgebra Linear se torna ainda maior conforme os procesadores vão sendo modificados de acordo com a Lei de Moore.

Assim, é possível que a metodologia da AEOS seja capaz de atingir diretamente este problema, podendo causar impacto na produção e manutenção de bibliotecas.

5.4.4.2 Condições Básicas para AEOS

As condições básicas para se utilizar uma biblioteca que usa as metodologias da AEOS, são:

- *Isolamento das rotinas críticas:* Assim como as bibliotecas tradicionais, os núcleos críticos de computação devem ser identificados e separados em subrotinas;
- *Um método de adaptação de software para diferentes tipos de ambiente:* Tendo em vista que a AEOS depende da tentativa de vários meios de se executar operações críticas em termos de performance, o programador deve ser capaz de oferecer rotinas que utilizem diversos tipos de otimização. Isto pode ser feito através de parâmetros em um código fixo que, quando variam, correspondem por exemplo a diferentes tamanhos de cache, etc. Ou, de maneira mais geral, através de um gerador de código parametrizado que produza uma infinidade de implementações. Por mais geral que seja a estratégia de adaptação, existirão limitações de acordo com a arquitetura que devem ser identificadas com o objetivo de aumentar a flexibilidade do código;
- Robustez e sensibilidade de tempo: Tendo em vista que a medição do tempo de execução é usada para selecionar o melhor código, o temporizador deve ser robusto o suficiente, independente do tipo de máquina.
- Mecanismo de busca apropriado: Para que possa ser feita a busca pela melhor implementação.

5.4.4.3 Métodos de adaptação de Software

Essencialmente, existem 2 métodos diferentes de adaptação de software. O primeiro e mais utilizado envolve a parametrização de características que variam conforme a máquina. Na Álgebra Linear, o parâmetro mais importante seria provavelmente o tamanho de blocos de matrizes que variam dependendo da memória cache disponível. Este método é conhecido como *adaptação parametrizada*. Porém, nem toda variável de arquitetura pode ser passada como parâmetro, como por exemplo cache de instrução, tamanho de ponto flutuante, etc. Portanto, há a necessidade de um outro método, conhecido como *adaptação de código fonte*, que envolve diferentes implementações da mesma operação.

Há duas formas, no mínimo, de se implementar a adaptação de um código fonte. A mais simples é reunir vários códigos otimizados e, através de uma busca heurística escolher o melhor deles por meio de tentativas.

O segundo método de adaptação de código fonte é conhecido como *geração de código*. Neste método, um gerador de código é produzido. Este gerador recebe como parâmetro as várias adaptações de código fonte a serem feitas, tais como tamanho de cache de instrução, tamanho de ponto flutuante, etc. Dependendo dos parâmetros, o gerador de código produz um código fonte com as características requisitadas.

O ATLAS é um projeto cuja as metodologias da AEOS vem sendo aplicadas. Porém, não foi o único. O primeiro projeto a utilizar estas técnicas para Álgebra Linear foi o PHiPAC. O objetivo inicial do ATLAS foi oferecer uma implementação eficiente do BLAS (Basic Linear Algebra), porém já foi feita a primeira tentativa de ir além do BLAS (por exemplo, o mais recente release contém algumas rotinas de níveis mais alto vindas do LAPACK (ANDERSON et al., 1995)).

Agora que já se tem uma base teórica a respeito dessas bibliotecas que podem ser utilizadas em métodos iterativos, serão apresentados, no próximo capítulo, testes de desempenho utilizando o BLAS para tratamento de multiplicação matriz-matriz e matriz-vetor e o Trilinos para comparação de armazenamento de matrizes.

6 TESTES E RESULTADOS

Neste capítulo serão vistos alguns testes utilizando-se as bibliotecas vistas no capítulo anterior. Na seção 6.1 será feito um teste de comparação entre as distribuições de BLAS abordadas aqui neste trabalho. Na seção 6.2, será visto uma comparação entre dois formatos de armazenamento de matrizes, o *CRS* e o *BCRS*, utilizando-se as classes do Epetra. Serão utilizadas para esses testes os seguintes computadores:

Computador 1: Processador: Intel Core 2 Duo, modelo E6600, 2.4 GHz, 4 MB de L2; Memória RAM: 2 GB, DDR-2/533 MHz; Sistema Operacional: Linux (Ubuntu 8.04).

Computador 2: Processador: AMD Athlon X2, modelo 4.200+, 2.2 GHz, 1 MB de L2; Memória RAM: 4 GB, DDR/400 MHz; Sistema Operacional:Linux (Debian Etch 4.0).

Computador 3: Processador: Intel Xeon Quad, modelo E5310, 1.6 GHz, 8 MB de L2 (placa com dois processadores com 2$\times$4 MB de L2); Memória RAM: 4 GB, DDR-2/667 MHz; Sistema Operacional: Linux (Debian Etch).

6.1 BLAS

Nos algoritmos do Capítulo 5, é possível observar a presença freqüente de sistemas lineares triangulares e das *multiplicações matriz-vetor*, que neste trabalho, juntamente com as *multiplicações matriz-matriz*, são chamadas de *núcleos de computação intensiva* (NCI).

Nesta seção, a fim de se dar um tratamento mais adequado aos NCI que podem ser decisivos em relação ao tempo de uma simulação, testou-se as operações matriz-vetor (por aparecer constantemente em algoritmos como os relacionados no do Capítulo 3) e matriz-matriz (onde se gasta muito tempo e recurso de hardware). Assim, seja A um matriz densa de números randômicos, de ordem n, e x um vetor de n elementos randômicos, para se testar a multiplicação matriz-vetor, repetiu-se a operação Ax por 1000 (mil) vezes e, para se testar a multiplicação matriz-matriz, repetiu-se a operação A^2 por 10 (dez) vezes, utilizando-se, respectivamente, as subrotinas DGEMV (D=double, GE=general, MV=matrix-vector) e DGEMM (D=double, GE=general, MM=matrix-matrix), oriundas das subrotinas básicas de álgebra linear vistas no capítulo anterior. Os resultados dos testes estão expressos nas Tabelas 2, 3, 4, 5, 6 e 7, para as máquinas descritas acima.

Tabela 2: Tempo (em segundos) de Execução da rotina DGEMV no Computador 1.

Dimensão ($\times 10^3$)	ACML	ATLAS	BLAS	Goto	MKL
1,0	3,00	**1,37**	4,76	3,08	1,61
2,0	13,52	**6,98**	19,03	14,41	7,26
3,0	31,92	**15,79**	42,79	32,66	16,19
4,0	58,03	**28,04**	76,04	57,69	28,76
6,0	131,01	**63,67**	170,89	129,87	64,68
7,0	175,69	**87,16**	232,67	177,66	88,73

Tabela 3: Tempo (em segundos) de Execução da rotina DGEMM no Computador 1.

Dimensão ($\times 10^3$)	ACML	ATLAS	BLAS	Goto	MKL
1,0	4,64	2,70	47,52	2,65	**2,50**
2,0	37,40	21,54	381,50	**18,72**	18,87
3,0	125,05	72,39	1282,50	67,38	**62,68**
4,0	299,80	171,06	3045,30	151,84	**145,53**
6,0	1008,20	580,09	10337,00	501,49	**488,83**
7,0	1593,50	915,88	16439,00	842,59	**777,50**

As Tabelas 2 e 3 (que mostram os resultador obtidos para o computador 1), assim como as tabelas com os resultados das outras configurações, têm como objetivo comparar o BLAS original, amplamente utilizado em resolução de sistemas lineares, com as demais distribuições de BLAS. Neste primeiro teste, a biblioteca MKL foi a que apresentou os melhores tempos para a multiplicação matriz-matriz (DGEMM - Tabela 3). Já na multiplicação matriz-vetor (DGEMV - Tabela 2), o ATLAS apresentou resultados melhores, porém bem próximos aos da MKL.

Agora, fazendo-se o teste com um processador AMD (Computador 2), observa-se, na Tabela 4, que a biblioteca ATLAS apresentou o melhor desempenho na multiplicação matriz-vetor. É importante notar, ainda na Tabela 4, os tempos obtidos pelas bibliotecas ACML e Goto em relação ao BLAS original. Percebe-se que, nesta configuração (computador 2), essas bibliotecas levaram mais tempo para executar a operação Ax. Já na multiplicação matriz-matriz, Tabela 5, a biblioteca ACML apresentou os melhores tempos.

Tabela 4: Tempo (em segundos) de Execução da rotina DGEMV no Computador 2.

Dimensão ($\times 10^3$)	ACML	ATLAS	BLAS	Goto	MKL
1,0	5,39	**2,77**	3,88	5,29	3,00
2,0	21,33	**10,99**	15,48	20,80	12,26
3,0	47,42	**25,88**	34,74	46,35	27,71
4,0	83,67	**45,47**	61,83	81,89	48,92
6,0	187,63	**104,51**	142,69	182,84	111,71
7,0	254,59	**139,61**	195,20	248,43	151,05

Tabela 5: Tempo (em segundos) de Execução da rotina DGEMM no Computador 2.

Dimensão ($\times 10^3$)	ACML	ATLAS	BLAS	Goto	MKL
1,0	**5,40**	5,42	39,27	9,37	6,65
2,0	**41,18**	42,24	312,67	65,07	48,07
3,0	**137,42**	141,33	1052,45	218,50	155,38
4,0	**323,28**	334,89	2485,30	508,73	375,61
6,0	**1080,87**	1553,05	8638,07	1728,61	1232,99
7,0	**1712,03**	2360,11	13880,99	2759,26	1918,63

Fazendo-se o teste para o Computador 3, é possível perceber que os melhores tempos para a rotina DGEMV foram apresentados pelo GotoBLAS (Tabela 6), enquanto que para a rotina DGEMM, o MKL se mostrou bastante eficiente (Tabela 7).

Tabela 6: Tempo (em segundos) de Execução da rotina DGEMV no Computador 3.

Dimensão ($\times 10^3$)	ACML	ATLAS	BLAS	Goto	MKL
1,0	2,37	2,72	7,13	**2,43**	1,61
2,0	66,63	13,43	28,62	**10,88**	69,52
3,0	169,00	29,55	64,39	**24,58**	164,71
4,0	294,54	52,11	114,44	**43,40**	292,82
6,0	646,30	117,75	257,34	**97,02**	646,16
7,0	883,58	164,43	349,85	**133,93**	888,67

Tabela 7: Tempo (em segundos) de Execução da rotina DGEMM no Computador 3.

Dimensão ($\times 10^3$)	ACML	ATLAS	BLAS	Goto	MKL
1,0	6,11	**4,38**	71,38	4,69	4,41
2,0	67,77	33,95	572,50	31,99	**31,00**
3,0	216,57	114,06	1929,47	102,83	**99,58**
4,0	498,69	264,61	4572,53	239,11	**231,78**
6,0	628,32	906,61	15429,04	782,30	**764,34**
7,0	2556,15	1423,91	24494,29	1249,74	**1215,83**

É importante lembrar que as bibliotecas aqui utilizadas foram compiladas com as orientações básicas dos desenvolvedores, exceto a biblioteca BLAS padrão que foi obtida a partir dos repositórios do Ubuntu e do Debian. É importante ressaltar também que os programas que utilizaram a ACML e a ATLAS foram compilados com a opção `- fopenmp`, o que utilizou a biblioteca GotoBLAS foi "linkado" com a biblioteca `-lpthread` e o que utilizou a MKL utilizou as bibliotecas `-lmkl_em64t`, `-lguide` e `-lpthread`, conforme as orientações dos respectivos desenvolvedores.

6.2 Armazenamento de Matrizes

Para se fazer o teste comparativo entre a classe Epetra_CrsMatrix e a classe Epetra_VbrMatrix, foi adotada a seguinte metodologia:

1. Em primeiro lugar, foi construída uma matriz utilizando o MATLAB, assim

$$\mathbf{A} = \begin{bmatrix} A_{i,i} & 0 \\ 0 & A_{i,i} \end{bmatrix}. \tag{6.1}$$

2. Cada $A_{i,i}$ é uma matriz densa, onde $m = n = 840$, que foi obtida randomicamente.

3. Foram implementados os seguintes códigos:

```
/* Multiplicacao Matriz-Vetor
  utilizando a classe Epetra_VbrMatrix */

t0 = clock();
for (int k = 0 ; k < repet ; k++)
{
```

```
        // Objeto da classe Epetra_VbrMatrix
        readVbrA->Multiply(false,vbrx,vbry);
}
tf = clock();
dt1 = (double)(tf-t0)/CLOCKS_PER_SEC;

/* onde vbrx e vbry são objetos
 da classe Epetra_Vector */
```

e

```
/* Multiplicacao Matriz-Vetor
 utilizando a classe Epetra_CrsMatrix */

t0 = clock();
for (int k = 0; k < repet; k++)
{
     // Objeto da classe Epetra_CrsMatrix
     readA->Multiply(false,x,y);
}
tf = clock();
dt2 = (double)(tf-t0)/CLOCKS_PER_SEC;
```

um computador com dois processadores

```
/* onde x e y são objetos(Intel Core 2 Duo)
 da classe Epetra_Vector */
```

Isto é, a multiplicação Matriz-Vetor foi executada numa estrutura de repetição (onde o número de repetições é igual `repet`, nos códigos acima). Apurou-se o tempo através do comando `clock()` do C++. A cada teste realizado, foi-se aumentando o tamanho do bloco da matriz represetada no código acima pela classe Epetra_VbrMatrix (considerando sempre blocos fixos). Os resultados para `repet = 1000`, estão representados pelo gráfico da Figura 8, para o Computador 1, e pelo gráfico da Figura 9, para o Computador 2. Neste teste, estamos considerando o Trilinos compilado com o ATLAS, uma vez que o GotoBLAS executou a multiplicação somente para tamanho de bloco menor ou igual a 6. Lembrando-se que o tempo apurado nestes gráficos é o tempo total de 1000 repetições.

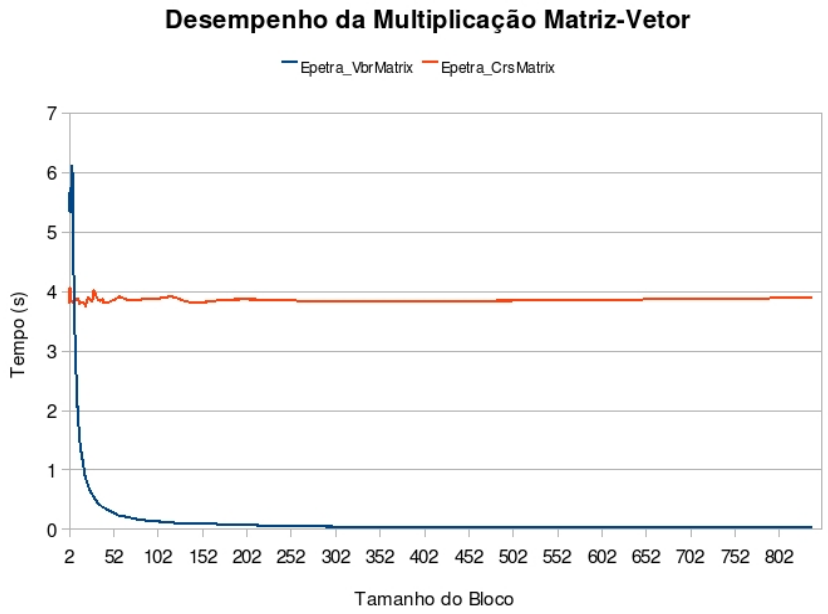


Figura 8: Gráfico de Desempenho dos objetos das classes Epetra_VbrMatrix e Epetra_CrsMatrix. Aqui foi utilizado um computador Intel Core 2 Duo, Modelo E6600, 2.4 MHz, com 4096 KB de memória cache L2 e com 2 GB de memória RAM/DDR2-533MHz. Lembrando-se que o tempo mostrado aqui é o tempo total de 1000 repetições.

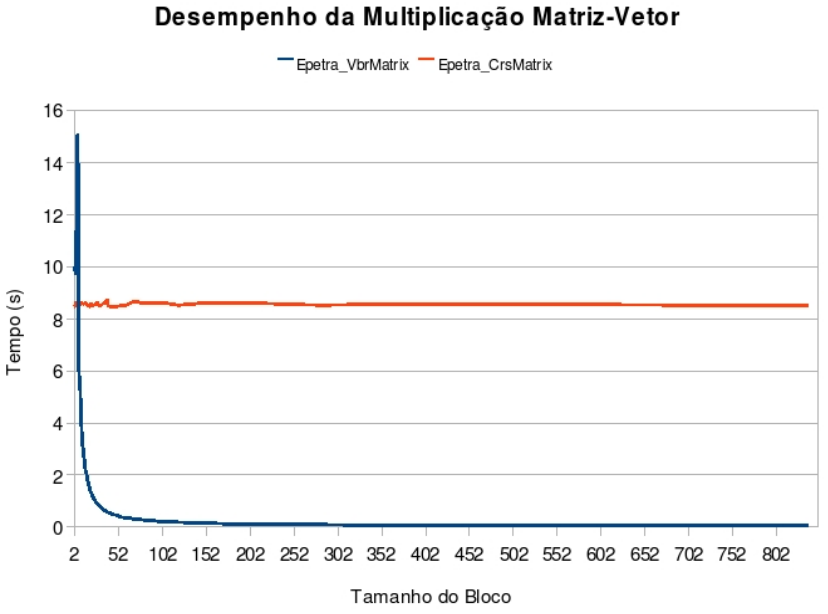


Figura 9: Gráfico de Desempenho dos objetos das classes Epetra_VbrMatrix e Epetra_CrsMatrix. Aqui foi utilizado um computador com dois processadores Intel Xeon Quad, modelo E5310, 1.6 GHz, 8 MB de L2 e com 4 GB de memória RAM/DDR2-667MHz. Lembrando-se que o tempo mostrado aqui é o tempo total de 1000 repetições.

O objetivo deste teste foi mostrar a diferença entre formas de armazenamento de matrizes esparsas. Dependendo do contexto, a forma de armazenamento pode influenciar no desempenho de um programa.

7 CONCLUSÃO

Neste trabalho procuramos listar alguns aspectos relevantes da solução de sistemas lineares provenientes da discretização de sistemas de equações diferenciais parciais utilizados para modelar o enchimento de reservatórios de hidrelétricas. Para isso, fizemos uma descrição do problema físico que estamos tratando, da sua formulação matemática e dos métodos utilizados para sua discretização. Apresentamos algumas abordagens para a solução iterativa de sistemas lineares de grande porte. Mostramos alguns métodos baseados em projeções em espaços de Krylov, discutimos reordenamentos e armazenamentos de matrizes e pré-condicionadores. Em especial, estudamos implementações atuais dos núcleos de cálculo intensivo: produto matriz-vetor e produto matriz-matriz, para os quais fizemos testes comparativos em máquinas de nosso laboratório.

A partir dos testes feitos no capítulo 6, percebe-se que é importante, no que se refere à resolução de sistemas lineares de dimensões elevadas, dedicar especial atenção a pequenos núcleos computacionais presentes em todo algoritmo que implementa um método iterativo. Nestes testes, conclui-se também que o tipo de armazenamento de uma matriz, dependendo de sua estrutura, pode prejudicar o desempenho de um algoritmo. Combinamos essas bibliotecas com o pacote Trilinos (visto no capítulo 5), pudemos observar o impacto importante da escolha da estrutura de dados. O Trilinos apresenta vários outros pacotes de computação científica (além do Epetra) que estão implementados em paralelo utilizando o paradigma de passagem de mensagens.

Sendo assim, são objetos de futuras pesquisas:

- O estudo da performance das alternativas de BLAS em máquinas multicore e clusters;
- Comparação entre os demais pacotes do Trilinos, tais como, o *AztecOO* (resolução Iterativa de de Sistemas Lineares), *Ifpack* (pré-condicionadores), *ML* (pré-condicionadores multi-grid e multi-nível), *Meros* (pré-condicionadores em bloco, utilizados em problemas onde há acoplamento de variáveis, tais como, os regidos pelas equações de Navier-Stokes), além de outros.
- Otimização do BLAS, estudando também as alternativas do BLAS esparso.

REFERÊNCIAS

ANDERSON, E. et al. *LAPACK - Users' Guide*. 2nd. ed. Philadelphia: SIAM, 1995.

ANJOS, G. R. dos. *Solução do Campo Hidrodinâmico em Células Eletroquímicas pelo Método de Elementos Finitos*. Dissertação (Mestrado) — COPPE/UFRJ, Março 2007.

ARNOLDI, W. The principle of minimized iteration in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, v. 9, p. 17–25, 1951.

BENZI, M.; GOLUB, G. H.; LIESEN, J. Numerical solution of saddle point problems. *Acta Numerica*, v. 14, p. 1–137, May 2005.

BLAST Forum. *Basic linear algebra subprograms techical (BLAST) forum*. [S.l.], August 2001.

BONDY, J. A.; MURTY, U. S. R. *Graph Theory with Applications*. London: Macmillan, 1976.

CHANG, W.; GIRALDO, F.; PEROT, B. Analysis of an exact fractional step method. *Journal of Computational Physics*, v. 180, p. 183–199, 2002.

CHORIN, A. J. Numerical solution of the navier-stokes equations. *Mathmatics of Computation*, v. 22, p. 745–762, 1968.

CUTHILL, E.; MCKEE, J. Reducing the bandwidth of sparse symmetric matrices. In: *Proceedings of the 1969 24th national conference*. New York, NY, USA: ACM Press, 1969. p. 157–172. Disponível em: <http://portal.acm.org/citation.cfm?id=805928>.

DAVIES, A. J. *The finite element method: a first approach*. [S.l.]: Claredon Press - Oxford, 1980.

DEMMEL, J. W. *Applied Numerical Linear Algebra*. Berkeley, California: SIAM, 1997.

DONGARRA, J. et al. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Soft.*, v. 16, p. 1–17, 1990.

DONGARRA, J. J. et al. An extended set of FORTRAN Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, v. 14, n. 1, p. 1–17, 1988.

EISENSTAT, S. C.; ELMAN, H. C.; SCHULTZ, M. H. Variational iterative methods for nonsymmetric systems of linear equations. *SIAM Journal on Numerical Analysis*, SIAM, v. 20, n. 2, p. 345–357, 1983. Disponível em: <http://link.aip.org/link/?SNA/20-/345/1>.

FLETCHER, R. Conjugate Gradient Methods for Indefinite Systems. In: WATSON, G. (Ed.). *Proc. of the Dundee Biennial Conference on Numerical Analysis (1975)*. [S.l.]: Springer Verlag„ Berlin, Heidelberg, New York, 1975. (Lecture Notes in Mathematics, v. 506), p. 73–89.

FORTUNA, A. O. *Técnicas Computacionais para Dinâmica dos Fluidos*. [S.l.]: Edusp, 2000.

GAMA, R. M. S. da. *Elementos de Cinemática dos Meios Contínuos*. [S.l.]: EdUerj, 2005.

GEORGE, A. *Computer implementation of the finite element model*. Computer Sci. Dept., Stanford Univ., Stanford, California, 1971.

GEORGE, A.; LIU, J. W. H. An implementation of a pseudoperipheral node finder. *ACM Trans. Math. Softw.*, ACM Press, New York, NY, USA, v. 5, n. 3, p. 284–295, 1979. ISSN 0098-3500. Disponível em: <http://doi.acm.org/10.1145/355841.355845>.

GEORGE, A.; LIU, W. H. The evolution of the minimum degree ordering algorithm. *SIAM Rev.*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, v. 31, n. 1, p. 1–19, March 1989. ISSN 0036-1445. Disponível em: <http://portal.acm-.org/citation.cfm?id=75574>.

GOLUB, G. H.; LOAN, C. F. V. *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)*. The Johns Hopkins University Press, 1996. Paperback. ISBN 0801854148. Disponível em: <http://www.amazon.ca/exec/obidos-/redirect?tag=citeulike09-20\&path=ASIN/0801854148>.

GOTO, K.; GEIJN, R. *On reducing TLB misses in matrix multiplication*. 2002. Technical Report TR-2002-55, The University of Texas at Austin, Department of Computer Sciences, 2002. FLAME Working Note #9. (ftp://ftp.cs.utexas.edu/pub/techreports/tr02-55.ps.gz). Disponível em: <citeseer.ist.psu.edu/goto02reducing.html>.

GREENBAUM, A. *Iterative methods for solving linear systems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1997. ISBN 0-89871-396-X.

GRESHO, P. On the theory of semi-implicit projection methods for viscous incompressible flow and its implementation via the finite element method that also introduces a nerly consistent mass matrix. part 1: Theory. *International Journal for Numerical Methods in Fluids*, v. 11, p. 587–620, 1990.

GRESHO, P.; CHAN, S. T. On the theory of semi-implicit projection methods for viscous incompressible flow and its implementation via the finite element method that

also introduces a nerly consistent mass matrix. part 2: Implementation. *International Journal for Numerical Methods in Fluids*, v. 11, p. 621–659, 1990.

HEROUX, M. et al. *An Overview of Trilinos*. [S.l.], 2003.

HIGHAM, N. J. *Accuracy and Stability of Numerical Algorithms*. Manchester, england. [S.l.]: SIAM, 1996.

HORN, R.; JOHNSON, C. R. *Matrix Analysis*. Cambridge University Press, 1985. xiii + 561 p. ISBN 0-521-30586-1 (hard), 0-521-38632-2 (soft). Disponível em: <http://www.loc.gov/catdir/description/cam023/85007736.html>.

IPSEN, I. C. F.; MEYER, C. D. The Idea Behind Krylov Methods. *American Mathematical Monthly*, v. 105, n. 10, p. 889–899, 1998.

LAWSON, C. L. et al. Basic linear algebra subprograms for fortran usage. *Application for Computing Machinery Transactions on Mathematical Software*, v. 5, p. 308–323, 1979.

LEE, M. J.; OH, B. D.; KIM, Y. B. Canonical fractional-step methods and consistent boundary conditions for the incompressible navier-stokes equations. *Journal of Computational Physics*, v. 168, p. 73–100, 2001.

LIU, W.-H.; SHERMAN, A. H. Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices. *SIAM Journal on Numerical Analysis*, v. 13, n. 2, p. 198–213, abr. 1976.

NI, M.-J.; KOMORI, S.; MORLEY, N. Projection methods for the calculation of incompressible unsteady flows. *Numerical Heat Transfer*, v. 44, p. 553–551, 2003.

PEROT, J. B. An analysis of the fractional step method. *Journal of Computational Physics*, v. 108, p. 51–58, 1993.

POZO, R.; REMINGTON, K. A. *IML++*. [S.l.], April 1996.

SAAD, Y. *Iterative Methods for Sparse Linear Systems*. 2nd. ed. [S.l.]: SIAM, 2003.

SAAD, Y.; SCHULTZ, M. H. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, SIAM, v. 7, n. 3, p. 856–869, 1986. Disponível em: <http://link.aip.org-/link/?SCE/7/856/1>.

SALA, M.; HEROUX, M. A.; DAY, D. M. *Trilinos Tutorial*. [S.l.], 2004.

SHAPIRA, Y. *Solving PDEs in C++: Numerical Methods in a Unified Object-Oriented Approach*. [S.l.]: Technion-Institute of Technology, Haifa, Israel, 2006.

SHEWCHUK, J. R. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain Edition 1$\frac{1}{4}$*. [S.l.], August 1994.

SIMEONI, F. *Simulação de Escoamentos Multifásicos em Malhas não Estruturadas*. Tese (Doutorado) — Universidade de São Paulo, São Carlos„ Agosto 2005.

SONNEVELD, P. CGS, A Fast Lanczos-Type Solver for Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, SIAM, v. 10, n. 1, p. 36–52, 1989. Disponível em: <http://link.aip.org/link/?SCE/10/36/1>.

TREFETHEN, L. N.; Bau, III, D. (Ed.). *Numerical Linear Algebra*. [S.l.]: SIAM, 1997. xii + 361 p. ISBN 0-89871-361-7.

VORST, H. A. van der. BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, SIAM, v. 13, n. 2, p. 631–644, 1992. Disponível em: <http://link.aip.org/link/?SCE/13/631/1>.

VORST, H. A. van der. *Iterative Krylov Methods for Large Linear systems*. [S.l.]: Cambridge University Press, 2003.

VORST, H. van der. Krylov subspace iteration. *Computing in Science & Engineering*, v. 2, n. 1, p. 32–37, Jan/Feb 2000. ISSN 1521-9615.

WHALEY, R. C.; PETITET, A.; DONGARRA, J. J. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, v. 27, n. 1–2, p. 3–35, 2001. Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 (www.netlib.org/lapack/lawns/lawn147.ps).

ZIENKIEWICZ, O. C.; TAYLOR, R. L. *The Finite Element Method Volume 1: The Basics*. 5th. ed. [S.l.]: Butterworth-Heinemann, 2000.

ZIENKIEWICZ, O. C.; TAYLOR, R. L. *The Finite Element Method Volume 3: Fluid Mechanics*. 5th. ed. [S.l.]: Butterworth-Heinemann, 2000.

APÊNDICE A – Alguns Tópicos de Álgebra Linear

Nesta seção estão algumas definições importantes que serão utilizadas ao longo da issertação.

Definição A.1 *Seja $\mathcal{B}$ um espaço vetorial de $\mathbb{R}^n$. Pode-se dizer que $\mathcal{B}$ é um* espaço vetorial normado *se existir uma função $||\cdot|| : \mathcal{B} \to \mathbb{R}$ (*Norma *ou* Norma Vetorial*) tal que*

1.$||x|| \geq 0$ *e* $||x|| = 0$ *se e somente se* $x = 0$*;*

2.$||\alpha x|| = |\alpha| \cdot ||x||$*, para* $\alpha \in \mathbb{R}$ *qualquer;*

3.$||x + y|| \leq ||x|| + ||y||$*.*

Observação A.1 *Caso o espaço vetorial seja o das matrizes, a função $||\cdot||$ (definida em A.1) recebe o nome de* Norma Matricial*.*

Definição A.2 *Uma matriz real simétrica A é dita* positiva-definida *(s.p.d.) se* $x^T Ax > 0$*,* $\forall x \neq 0$*.*

Definição A.3 *Uma matriz complexa $A = [a_{i,j}]$ $n \times n$ é dita* diagonal-dominante *se*

$$|a_{i,i}| \geqslant \sum_{\substack{i=1 \\ j \neq i}}^{n} |a_{i,j}|$$

para todo $1 \leqslant i \leqslant n$*.*

Definição A.4 *Seja $||\cdot||_{m\times n}$ uma norma matricial sobre o espaço das matrizes de dimensão $m \times n$ e $||\cdot||_{m\times p}$ uma norma matricial sobre o espaço das matrizes de dimensão $m \times p$. Estas normas são ditas* mutuamente consistentes *se*

$$||A \cdot B||_{m\times p} \leq ||A||_{m\times n} \cdot ||B||_{n\times p},$$

onde A é uma matriz $m \times n$ e B é uma matriz $n \times p$.

Definição A.5 *Seja A uma matriz $m \times n$, $||\cdot||_{\hat{m}}$ uma norma sobre o espaço vetorial $\mathbb{R}^m$ e $||\cdot||_{\hat{n}}$ uma norma sobre o espaço vetorial $\mathbb{R}^n$. Tem-se que*

$$||A||_{mn} \equiv \max_{x\neq 0,\, x\in\mathbb{R}^n} \frac{||Ax||_m}{||x||_n} \quad \textit{ou} \quad ||A||_{mn} \equiv \max_{||x||=1,\, x\in\mathbb{R}^n} ||Ax||_m$$

é dita Norma Induzida *ou* Subordinada*.*

Lema A.1 *Uma norma induzida é uma norma matricial (DEMMEL, 1997).*

Definição A.6 *Chama-se* Norma-p *a norma induzida onde* $\hat{m} = \hat{n} = 1, 2, \cdots, \infty$ *e representa-se por* $|| \cdot ||_p$.

Definição A.7 *A* Norma de Frobenius *é dada por*

$$||A||_F = \left(\sum_{j=1}^{m} \sum_{i=1}^{n} |a_{ij}|^2 \right)^{\frac{1}{2}}.$$

Lema A.2 $||AB|| \leq ||A|| \cdot ||B||$ *(mutuamente consistente) para qualquer norma induzida ou norma de Frobenius (DEMMEL, 1997).*

Lema A.3 $||Ax|| \leq ||A|| \cdot ||x||$ *para uma norma vetorial e sua correspondente matricial, ou para norma-2 vetorial e norma de Frobenius (DEMMEL, 1997).*