unesp
UNIVERSIDADE ESTADUAL PAULISTA
Instituto de Biociências, Letras e Ciências Exatas
DEPARTAMENTO DE CIÊNCIAS DA COMPUTAÇÃO E ESTATÍSTICA

ESTUDO DE MÉTODOS ITERATIVOS E PRÉ-CONDICIONADORES PARA SISTEMAS LINEARES ESPARSOS NÃO-SIMÉTRICOS

Fábio Henrique Pereira

Dissertação de Mestrado
Pós-Graduação em Matemática Aplicada

Rua Cristovão Colombo, 2265
Caixa Postal 136
15054-000 - São José do Rio Preto - SP - Brasil
Telefone: (017) 221-2444
Fax: (017) 221-2445

ESTUDO DE MÉTODOS ITERATIVOS E PRÉ-CONDICIONADORES PARA SISTEMAS LINEARES ESPARSOS NÃO-SIMÉTRICOS

Fábio Henrique Pereira

Orientador: Prof. Dr. Sérgio Luís Lopes Verardi

Dissertação apresentada ao Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto, São Paulo, para a obtenção do título de Mestre em Matemática Aplicada.

São José do Rio Preto

20 de Janeiro de 2003

À minha noiva, amiga e amada,

Solange.

Aos meus pais,

José Pereira e Maria da Glória *(in memorian)*

Dedico.

"É preciso amar as pessoas como se não houvesse o amanhã,

porque se você parar pra pensar, na verdade não há!"

Renato Russo.

Agradecimentos

A Deus, pela vida.

Ao Prof. Dr. Sérgio Luís Lopes Verardi, pela orientação e paciência.

Aos demais alunos e professores do DCCE, pela amizade.

À CAPES, pelo apoio financeiro.

À minha família, pela compreensão e apoio.

Resumo

Neste trabalho, os fundamentos teóricos e a implementação computacional dos principais métodos iterativos e técnicas de pré-condicionamento para solução de sistemas matriciais lineares não-simétricos e esparsos são discutidos. São também apresentados resultados numéricos da aplicação de tais métodos à solução do problema do fluxo magnetohidrodinâmico (MHD) em canais.

Palavras-chave: Métodos iterativos, Pré-condicionadores, Matrizes esparsas, Subespaço de Krylov.

Abstract

In this work, we study iterative methods and preconditioning technics for the solution of sparse unsymmetric linear systems, and we discuss their computational implementation. Such methods are applied to the solution magnetohydrodynamics (MHD) channel flow, and related numerical results are presented.

Keywords: Iterative methods, Preconditioning, Sparse matrix, Krylov subspace.

Sumário

Capítulo 1

Introdução

1.1 Motivação

Ao longo de todo este trabalho, serão discutidos formas eficientes de resolver um sistema de equações lineares. Matematicamente, trata-se de um problema relativamente simples, que pode ser posto como: dado uma matriz quadrada A de ordem n e um vetor b de mesma ordem, encontrar um vetor x tal que

$$Ax = b \tag{1.1}$$

Porém, do ponto de vista computacional, especialmente em função das limitações de memória, esse problema pode se tornar muito complicado e, em alguns casos, até mesmo impossível.

Segundo Van der Vorst [34] a maior parte do tempo gasto em computação científica é destinado à solução de sistemas de equações lineares. Esses sistemas em geral são muito grandes e exigem realmente um grande esforço computacional para serem resolvidos. Sendo assim, a tarefa de escolher uma ferramenta adequada para auxiliar na solução desses sistemas deve ser levada a sério.

Nesse sentido, existem duas opções: escolher um método direto ou utilizar uma técnica iterativa.

1.2 A opção pelos métodos iterativos

Imaginemos uma situação comum em problemas realísticos onde precisamos discretizar, através do uso de elementos (volumes) finitos, uma equação diferencial parcial em um domínio irregular. Suponha ainda que existem pontos deste domínio onde a solução apresenta grandes variações, exigindo por isso uma forte discretização. Esta situação surge naturalmente em um grande número de aplicações reais.

Uma vez que o número de pontos (vértices dos elementos) da discretização determinará a dimensão do sistema linear, poderemos obter sistemas lineares muito grandes impedindo o uso de métodos de solução direta baseadas em eliminação Gaussiana, em função das limitações de memória computacional. Note que com a aplicação do processo de eliminação de Gauss, uma matriz que era originalmente esparsa, como é o caso das matrizes que surgem desses problemas, acaba se tornando praticamente densa. *

A alternativa então é partir para o uso dos métodos iterativos. Na prática isso precisa ser feito com a ajuda de uma técnica de pré-condicionamento como veremos a seguir.

1.3 A necessidade de pré-condicionamento

Além de normalmente serem muito grandes, outra característica das matrizes que surgem de problemas reais é o fato de serem muito mal condicionadas. Como exemplo, podemos citar as matrizes que surgem de aplicações da dinâmica de fluidos computacional, como discretizações por elementos (volumes) finitos das equações de Navier-Stokes em domínios irregulares [36].

Nessas matrizes, as propriedades espectrais não favorecem a convergência dos métodos iterativos, pois os autovalores se distribuem num intervalo mui-

*Durante o processo de eliminação Gaussiana posições que eram nulas podem passar a ocupar elementos diferentes de zero. Na literatura, isto é conhecido como *"fill-in"*.

to grande ou alguns deles são muito próximos de zero. Como resultado, os métodos iterativos convergem muito lentamente e, muitas vezes divergem.

Uma saída para essa dificuldade é aplicar uma transformação linear adequada no sistema original para obter um sistema equivalente que possua propriedades espectrais mais favoráveis ao bom desempenho dos métodos iterativos. Na prática, pode ser comprovado que a aplicação de uma técnica adequada de pré-condicionamento melhora substancialmente a convergência de um método iterativo. Atualmente existem duas classes de pré-condicionadores, os implícitos e os explícitos.

Os pré-condicionadores chamados implícitos são baseados em técnicas de fatorização realizadas de forma incompleta, e aproximam diretamente a matriz original do sistema. Para serem aplicados, esses pré-condicionadores exigem a solução de sistemas triangulares envolvendo os fatores calculados. Estes são os pré-condicionadores mais populares [26].

Os pré-condicionadores explícitos são um pouco mais recentes e ainda não tão conhecidos e utilizados. São baseados na aproximação direta da inversa da matriz original e, por isso, sua aplicação é reduzida à realização de um simples produto matriz vetor. Essa classe de pré-condicionadores é interessante pelo fato de não sofrer dos problemas típicos dos métodos de fatorização que geram os pré-condicionadores implícitos, como o caso de encontrar pivô nulo durante a fatorização LU. Além disso, a forma como são construídos facilita enormemente a utilização de arquiteturas em paralelo. Essas duas classes de pré-condicionadores serão tratadas neste trabalho.

1.4 Um panorama dos métodos iterativos

Os métodos iterativos foram inicialmente desenvolvidos para resolverem grandes sistemas lineares com a matriz dos coeficientes diagonalmente dominante. Nestas situações, esses métodos sempre foram amplamente utilizados, principalmente em função da pouca quantidade de memória exigida por eles [26].

Porém, à medida que os problemas se tornaram mais complexos, como por exemplo, aqueles que surgem da aplicação da técnica de elementos finitos, os métodos iterativos eram praticamente abandonados. De acordo com Saad [26], até o final dos anos 80 quase nenhum dos grandes pacotes comerciais para problemas com elementos finitos incluíam métodos iterativos. Nesses casos predominavam o uso dos métodos diretos.

Essa situação começou a mudar a partir do surgimento de técnicas iterativas mais poderosas para solução de grandes sistemas lineares. Na prática, o desenvolvimento dessas técnicas está apoiado em trabalhos que surgiram a partir dos anos 50.

Os métodos baseados em relaxação, por exemplo, ganharam grande popularidade e produziram uma rica teoria para os métodos iterativos [1]. Porém, foi apenas a partir da criação do método do Gradiente Conjugado por Hestenes e Stiefel [28] e do algoritmo de Lanczos para sistemas lineares [13] que os métodos iterativos realmente começaram a ganhar força.

Definidas apenas para sistemas simétricos essas técnicas iterativas obtiveram grande sucesso e motivaram a criação de outros métodos iterativos para casos não simétricos. Nessa tendência, surgiram os trabalhos de Lanczos [19] e de Arnoldi [23] que vieram completar a base dos atuais métodos iterativos no subespaço de Krylov [†][23].

Neste trabalho abordaremos os principais métodos no subespaço de Krylov disponíveis na literatura especializada. Além do método dos Gradientes Conjugados, desenvolvidos para matrizes Simétricas Definidas Positivas (SDP), abordaremos diversos métodos para casos não simétricos, como, por exemplo, os métodos BiCG (*"Bi-Conjugate Gradient"*), CGS (*"Conjugate Gradient Squared"*), GPBICG (*"Generalized Product Type Methods based BICG"*), GMRES (*"Generalized Minimal Residual"*), DQGMRES (*"Direct Quase GMRES"*), além de uma versão do método BiCG sem o uso da matriz transposta,

[†]É o subespaço gerado por um vetor v_1 e sucesivas potências da matriz A aplicadas a esse vetor.

chamado TFBICG.

Atualmente, com o desenvolvimento de técnicas de pré-condicionamento, os métodos iterativos ampliaram seu espaço e passaram a ser utilizados em um número bem maior de situações [26].

1.5 Os testes numéricos

Todos os métodos iterativos e as técnicas de pré-condicionamento apresentadas neste trabalho foram submetidos a testes práticos. Esses testes foram divididos em duas etapas. A primeira corresponde à validação dos códigos implementados selecionando-se matrizes esparsas da coleção "Matrix Market [‡]"[21], e os resultados obtidos foram comparados com outros resultados conhecidos em trabalhos disponíveis na literatura.

Na segunda etapa, os métodos foram aplicados a um problema real obtido de uma formulação de Elementos Finitos Estabilizados do fluxo magnetohidrodinâmico (MHD).

1.6 Notação utilizada

Como é usual, letras maiúsculas do alfabeto português representam matrizes e letras minúsculas representam vetores.

As letras gregas minúsculas serão usadas para denotar escalares, e letras gregas maiúsculas, acompanhadas ou não de uma variável entre parênteses, denotarão polinômios. Assim,

$$\Phi_j = \Phi_j(t) = \text{ polinômio e } \alpha = \text{ escalar.}$$

Quando aparecer $\Phi_j(A)$ entenda-se um polinômio na matriz A. O índice que acompanha o polinômio sempre representa o grau deste polinômio.

[‡]Trata-se de uma base de dados testes para uso em estudos comparativos de algoritmos para álgebra linear numérica.

A^T é a transposta da matriz A e x^T é o vetor linha, transposto de x.

I representa a matriz identidade, isto é, uma matriz cujos elementos são tais que

$$a_{ij} = 0 \text{ e } a_{ii} = 1.$$

I_n é a matriz identidade de ordem n e e_j é j-ésima coluna da matriz identidade.

Em geral, um polinômio aplicado a um vetor será tratado como um vetor, e expressões que se reduzem a um escalar serão tratados com um escalar. Dessa forma, $\Pi_j(A)r$ = vetor, e $x^T y$ = escalar.

(u, v) denota o produto escalar entre dois vetores,

$$(u, v) = \sum_{i=1}^{n} u_i v_i. \tag{1.2}$$

O índice acompanhando um vetor ou um escalar geralmente denota o passo do processo iterativo.

O símbolo ■ é usado para indicar o final da demonstração de um teorema ou da prova de uma proposição.

Letras caligráficas representam subespaços vetoriais e o índice que as acompanha representa a dimensão do subespaço.

Por fim, $Span\{v_1, v_2, \ldots, v_n\}$ denota espaço gerado pelos vetores $\{v_1, v_2, \ldots, v_n\}$.

1.7 Organização da dissertação

Esta dissertação está dividida em oito capítulos. No primeiro capítulo é apresentado um panorama geral dos métodos iterativos, e a motivação para o estudo e aplicação desses métodos associados às técnicas de pré-condicionamento. No capítulo 2 são tratadas as técnicas de projeção, um pré requisito básico para o estudo teórico dos métodos iterativos feito nos capítulos 3 e 4. As técnicas de pré-condicionamento utilizadas são tratadas no capítulo 5.

A parte prática é abordada em dois capítulos. No capítulo 6 são descritos os aspectos gerais da implementação computacional dos métodos apresentados

nos capítulos anteriores, e os resultados dos testes numéricos são apresentados no capítulo 7.

Por fim, no último capítulo, são descritas as conclusões obtidas com a realização deste trabalho.

Capítulo 2

Os métodos iterativos e a técnica das projeções

A maioria das técnicas iterativas existentes para solução de grandes sistemas de equações lineares utilizam um processo de projeção em sua formulação [23]. Um processo de projeção representa uma forma canônica de obter uma aproximação para a solução de um sistema de equações lineares.

Considere o sistema linear

$$Ax = b \tag{2.1}$$

onde A é uma matriz real $n \times n$. A idéia principal das técnicas de projeção é extrair de um subespaço do $\mathbb{R}^n$ uma solução aproximada para o problema (2.1).

No contexto dos métodos iterativos isso significa resolver um sistema simplificado $Kx_0 = b$, onde K é uma matriz que aproxima a matriz A, e então tomar x_0 como uma aproximação para x. O processo iterativo surge na medida em que se deseja buscar uma correção z que satisfaça

$$A(x_0 + z) = b. \tag{2.2}$$

Isto fornece um novo sistema linear $Az = b - Ax_0$ que novamente é resolvido aproximadamente usando a matriz K. Este processo corresponde aos sucessivos passos de um método de projeção ([23],[34]).

Se $\mathcal{K}$ é este subespaço, chamado de subespaço das possíveis aproximações ou subespaço de busca, e se m é sua dimensão, então, em geral, m restrições precisam ser impostas para que seja possível extrair uma aproximação. Uma maneira típica de descrever essas restrições é impor m (independentes) condições de ortogonalidade. Especificamente, o vetor resíduo $b-Ax$ é condicionado a ser ortogonal a m vetores linearmente independentes. Estas restrições definem um outro subespaço $\mathcal{L}$, também de dimensão m, chamado subespaço das restrições.

A conexão entre a técnica de projeção e os métodos iterativos é clara. Quando resolvemos o sistema aproximado $Kx_0 = b$ e tomamos x_0 como uma solução aproximada para o sistema original, estamos buscando uma solução aproximada no subespaço gerado pela matriz K. Além disso, não basta buscarmos qualquer vetor x_0 pertencente à imagem da matriz K, é preciso impor restrições à solução aproximada de forma que a cada passo do processo essa solução esteja, em algum sentido, mais próxima da solução exata.

2.1 Método de Projeção geral

Seja A uma matriz real de dimensão n e $\mathcal{K}$, $\mathcal{L}$ dois subespaços do $\mathbb{R}^n$ ambos com dimensão m. Uma técnica de projeção sobre o subespaço $\mathcal{K}$ e ortogonal a $\mathcal{L}$ é um processo pelo qual se encontra uma solução aproximada $\widetilde{x}$ para o problema (2.1) impondo como condição que $\widetilde{x}$ pertença a $\mathcal{K}$ e que o novo vetor resíduo seja ortogonal a $\mathcal{L}$, ou seja:

encontrar $\widetilde{x} \in \mathcal{K}$, tal que

$$b - A\widetilde{x} \perp \mathcal{L} \tag{2.3}$$

Se uma estimativa inicial x_0 para solução é utilizada, deve-se buscar uma aproximação no subespaço afim $x_0 + \mathcal{K}$ e não mais no subespaço $\mathcal{K}$. Isto requer uma ligeira modificação da formulação (2.3). O problema aproximado deve ser redefinido como,

encontrar $\widetilde{x} \in x_0 + \mathcal{K}$, tal que

$$b - A\widetilde{x} \perp \mathcal{L}. \tag{2.4}$$

Note que se $\widetilde{x}$ é escrito na forma $\widetilde{x} = x_0 + \delta$, e o vetor resíduo inicial r_0 é definido como

$$r_0 = b - Ax \tag{2.5}$$

então a equação (2.4) torna-se:

$$b - A\widetilde{x} = b - A(x_0 + \delta) \perp \mathcal{L}$$

ou

$$r_0 - A\delta \perp \mathcal{L}.$$

Em outras palavras, a solução aproximada pode ser definida como

$$\widetilde{x} = x_0 + \delta, \quad \delta \in \mathcal{K} \tag{2.6}$$

$$(r_0 - A\delta,\ w) = 0, \quad \forall\ w \in \mathcal{L}. \tag{2.7}$$

Este é um passo de projeção básico, na sua forma mais geral. Grande parte das técnicas padrões usam uma sucessão dessas projeções. Nessa sucessão, normalmente um novo passo de projeção usa um novo par de subespaços $\mathcal{K}$ e $\mathcal{L}$, e uma estimativa inicial x_0 igual à aproximação mais recente obtida do passo de projeção anterior.

2.2 Identificando o subespaço das soluções aproximadas

Estabelecida uma conexão entre os métodos iterativos e a técnica das projeções o próximo passo é identificar o subespaço onde as sucessivas soluções aproximadas estão localizadas.

Para isso, retomamos o sistema linear definido pela relação (2.2) e o resolvemos aproximadamente usando novamente a matriz K. Obtemos assim o sistema linear aproximado

$$Kz_0 = b - Ax_0, \tag{2.8}$$

que fornece a nova aproximação

$$x_1 = x_0 + z_0. \tag{2.9}$$

Repetindo esse procedimento para x_1 cria-se um processo iterativo que estabelece as seguintes iterações básicas,

$$\begin{aligned} x_{i+1} &= x_i + z_i \\ &= x_i + K^{-1}(b - Ax_i) \\ &= x_i + \tilde{b} - \tilde{A}x_i, \end{aligned} \tag{2.10}$$

onde $\tilde{b}$ é o vetor obtido resolvendo $K\tilde{b} = b$, e $\tilde{A}$ a matriz que resolve $K\tilde{A} = A$.

A formulação (2.10) pode ser vista como as iterações básicas para o sistema linear pré-condicionado

$$\tilde{A}x = \tilde{b}, \tag{2.11}$$

com aproximação $K = I$ para $\tilde{A} = K^{-1}A$ [34].

Então, substituindo $\tilde{A}$ por A, e $\tilde{b}$ por b, por simplicidade, obtemos de (2.10)

$$\begin{aligned} x_{i+1} &= b + x_i - Ax_i \\ &= b + (I - A)x_i \\ &= x_i + r_i, \end{aligned} \tag{2.12}$$

onde foi usado que $r_i = b - Ax_i$.

Se este processo é repetido $i + 1$ vezes nós podemos observar que (2.12) produzirá

$$x_{i+1} = x_0 + r_0 + r_1 + r_2 + \ldots + r_i. \tag{2.13}$$

Além disso, multiplicando (2.12) por $-A$ e somando b obtemos

$$b - Ax_{i+1} = b - Ax_i - Ar_i, \tag{2.14}$$

ou, equivalentemente

$$\begin{aligned} r_{i+1} &= r_i - Ar_i && (2.15)\\ &= (I - A)r_i && (2.16)\\ &= (I - A)(I - A)r_{i-1} && (2.17)\\ &\vdots \\ &= (I - A)^{i+1}r_0 && (2.18)\\ &= \Phi_{i+1}(A)r_0, && (2.19) \end{aligned}$$

onde Φ_{i+1} é um polinômio de grau $i+1$ tal que $\Phi_{i+1}(0) = 1$.

Assim, supondo $x_0 = 0$ e usando as relações (2.13) e (2.16), obtemos uma nova aproximação no passo $i+1$

$$\begin{aligned} x_{i+1} &= r_0 + r_1 + r_2 + \ldots + r_i && (2.20)\\ &= \sum_{j=0}^{i} (I - A)^j r_0 && (2.21)\\ &\in Span\{r_0, Ar_0, \ldots, A_0^r\} \equiv \mathcal{K}_{i+1}(A, r_0). && (2.22) \end{aligned}$$

Podemos concluir então que a solução aproximada x_{i+1} está no subespaço $\mathcal{K}_{i+1}(A, r_0)$. Este tipo de subespaço, gerado por um vetor, r_0 neste caso, e sucessivas potências de A aplicada a esse vetor, é chamado subespaço de Krylov. No contexto dos métodos iterativos básicos a matriz K usada para resolver o sistema aproximadamente é aquela cujas colunas formam uma base para o subespaço de Krylov envolvido.

2.3 Resultados de otimalidade

Em um processo de projeção existem varias possibilidades de escolha para o subespaço das restrições $\mathcal{L}$. Quando $\mathcal{L} = \mathcal{K}$ temos um processo de projeção

ortogonal, pois nesse caso a restrição (2.3) imposta ao vetor resíduo implica que ele seja ortogonal ao subespaço de busca $\mathcal{K}$. Por outro lado, quando $\mathcal{L} \neq \mathcal{K}$ essa mesma restrição produz um vetor resíduo ortogonal ao subespaço $\mathcal{L}$ e, conseqüentemente, oblíquo ao subespaço de busca $\mathcal{K}$. Dessa forma, quando $\mathcal{L} \neq \mathcal{K}$, tem-se um processo de projeção oblíqua sobre o subespaço $\mathcal{K}$.

Como veremos no capítulo seguinte, essa variedade na escolha do subespaço $\mathcal{L}$ dá origem a diferentes métodos iterativos, e cada um desses métodos herda as propriedades do processo de projeção no qual se baseia. Dessa forma, é importante buscar resultados que possam ser usados como ferramentas para analisar a qualidade da aproximação obtida através de um processo de projeção.

Existem alguns resultados teóricos gerais que podem ser explorados sem que se tenha de especificar quais são os subespaços $\mathcal{K}$ e $\mathcal{L}$ que estão sendo usados.

Um exemplo desses resultados são as duas propriedades de otimalidade dos métodos de projeção, que são satisfeitos pelas soluções aproximadas em alguns casos e que serão estabelecidas a seguir.

Consideremos primeiro o caso onde A é simétrica definida positiva.

Proposição 2.3.1. *Assuma que A é SDP e $\mathcal{L} = \mathcal{K}$. Então, um vetor $\widetilde{x}$ é o resultado de um método de projeção ortogonal sobre $\mathcal{K}$ com o vetor inicial x_0 se, e somente se, ele minimiza a A-norma do vetor erro $(x_* - x)$ sobre $x_0 + \mathcal{K}$, isto é, se e somente se*

$$E(\widetilde{x}) = \min_{x \in x_0 + \mathcal{K}} E(x),$$

onde

$$E(x) \equiv (A(x_* - x), x_* - x)^{1/2}$$

e x_* é a solução exata.

Consideremos agora o caso onde $\mathcal{L}$ é definido por $\mathcal{L} = A\mathcal{K}$.

Proposição 2.3.2. *seja A uma matriz quadrada qualquer e assuma que $\mathcal{L} = A\mathcal{K}$. Então o vetor $\widetilde{x}$ é o resultado de um método de projeção oblíqua*

sobre $\mathcal{K}$, com o vetor inicial x_0, e ortogonalmente a $\mathcal{L}$ se, e somente se, ele minimiza a norma 2 do vetor resíduo $b - Ax$ sobre $x_0 + \mathcal{K}$,i.e., se e somente se

$$R(\widetilde{x}) = \min_{x \in x_0+\mathcal{K}} R(x),$$

onde $R(x) \equiv \| b - Ax \|_2$.

As provas para essas duas últimas proposições podem ser encontradas em [23].

2.4 Processos de projeção unidimensional.

Os casos de projeção unidimensional são definidos quando $\mathcal{K}$ é o subespaço gerado por um vetor v e $\mathcal{L}$ é o subespaço gerado por um vetor w. Neste caso, a nova aproximação toma a forma $x_{k+1} = x_k + \alpha v$, e a restrição imposta ao vetor resíduo $r_{k+1} = b - Ax_{k+1}$ é que

$$r_{k+1} \perp w.$$

Como

$$\begin{aligned} r_{k+1} &= b - Ax_{k+1} \\ &= b - Ax_k - \alpha Av \\ &= r_k - \alpha Av, \end{aligned} \tag{2.23}$$

da restrição imposta deduz-se que

$$\alpha = \frac{(r_k, w)}{(Av, w)} \tag{2.24}$$

As três escolhas mais populares para os vetores v e w são consideradas a seguir.

2.4.1 Método da descida máxima

O método de descida máxima é definido para o caso onde a matriz A é SDP. Este método consiste em tomar em cada passo $v = r$ e $w = r$, onde r é o vetor

resíduo, produzindo o seguinte algoritmo.

ALGORITMO 2.1:Método da Descida Máxima

1. Até a convergência faça:
2. $r = b - Ax$
3. $\alpha = (r, r)/(Ar, r)$
4. $x = x + \alpha r$
5. fim.

Cada passo do algoritmo acima minimiza

$$f(x) = \| x - x_* \|_A^2 = (A(x - x_*), (x - x_*)),$$

sobre todos os vetores da forma $x + \alpha d$, onde d é a direção oposta ao gradiente da função f, isto é, $d = -\nabla f$. A direção oposta ao gradiente é, localmente, a direção que produz o mais rápido decréscimo no valor de f.

2.4.2 Método do resíduo mínimo

Para este método não é necessário que a matriz A seja simétrica mas apenas definida positiva, isto é, é apenas necessário que $A + A^T$ seja Simétrica Definida Positiva. Com relação aos vetores, toma-se $v = r$ e $w = Ar$ em cada passo do procedimento, produzindo o seguinte algoritmo que minimiza a função $f(x) = \|b - Ax\|_2^2$ na direção do vetor r.

ALGORITMO 2.2:Método do resíduo mínimo (MR)

1. Até a convergência faça:
2. $r = b - Ax$
3. $\alpha = (Ar, r)/(Ar, Ar)$
4. $x = x + \alpha r$
5. fim.

2.4.3 Método do decréscimo máximo da norma do resíduo

Este método é uma combinação dos dois métodos anteriores. Minimiza a norma do vetor resíduo tomando como direção aquela oposta ao gradiente da função $f(x) = \| b - Ax \|_2^2$.

Em razão das escolhas feitas para v e w, $v = A^T r$ e $w = Av$, a única exigência feita sobre a matriz A é que ela seja não singular. Estas escolhas resultam na seguinte seqüência de operações:

$$\begin{aligned} r &= b - Ax, \\ v &= A^T r, \\ \alpha &= \| v \|_2^2 \, / \, \| Av \|_2^2 \\ x &= x + \alpha v. \end{aligned} \tag{2.25}$$

onde o α é obtido como em (2.24).

O algoritmo baseado na seqüência de operações acima requer três produtos de matriz por vetor o que o deixa com um custo computacional muito elevado. Porém, uma alteração na forma de calcular o resíduo permite reduzir para dois o número de produtos de matriz por vetor em cada iteração, produzindo o seguinte algoritmo.

ALGORITMO 2.3:Decréscimo máximo da norma do resíduo

1. Calcule $r = b - Ax$
2. Até a convergência faça:
3. $\quad v = A^T r$
4. $\quad$ Calcule Av e $\alpha = \| v \|_2^2 \, / \, \| Av \|_2^2$
5. $\quad x = x + \alpha v$
4. $\quad r = r - \alpha Av$
7. fim.

Capítulo 3

O método GMRES e o método dos Gradientes Conjugados

Como resultado do que foi exposto no capítulo anterior surgiram vários métodos iterativos que exploram o subespaço de Krylov na busca de uma boa solução aproximada para um sistema de equações lineares.

Esses métodos no subespaço de Krylov [13] são considerados atualmente como algumas das principais técnicas disponíveis para solução de grandes sistemas lineares. Esses métodos são processos de projeção onde a solução aproximada x_m do sistema linear

$$Ax = b, \tag{3.1}$$

é buscada em um subespaço de Krylov $\mathcal{K}_m$ gerado pelo conjunto de vetores

$$\{r_0, Ar_0, \ldots, A^{m-1}r_0\},$$

onde $r_0 = b - Ax_0$, com x_0 uma estimativa inicial para solução do sistema (3.1).

As diferentes escolhas possíveis para o subespaço $\mathcal{L}_m$, isto é, para as condições usadas para construir a aproximação, exercem uma importante influência no efeito das técnicas iterativas e dão origem às diferentes versões de métodos no subespaço de Krylov.

Na prática esses métodos podem ser divididos em três classes distintas, assumindo $x_0 = 0$, de acordo com a escolha do subespaço $\mathcal{L}_m$ [10].

1. **Busca do resíduo mínimo:** Busca uma solução aproximada x_m para a qual a norma Euclidiana $\|b - Ax_m\|_2$ é mínima sobre $\mathcal{K}_m(A, r_0)$. Como estabelecido na proposição 2.3.2 isto implica na escolha $\mathcal{L}_m = A\mathcal{K}_m$. Como exemplo de métodos desta classe podemos citar o GMRES e suas variantes.

2. **Condição de Galerkin:** A solução aproximada é tal que o resíduo por ela gerado é ortogonal ao subespaço de Krylov onde ela está localizada. Ou seja,

$$x_m \in \mathcal{K}_m(A, r_0)$$

e

$$b - Ax_m \perp \mathcal{K}_m(A, r_0).$$

Em outras palavras, $\mathcal{L}_m = \mathcal{K}_m$. Nos casos onde a matriz A é simétrica definida positiva esta escolha de $\mathcal{L}_m$ minimiza a $A-$norma do vetor erro, assim como estabelecido na proposição 2.3.1, e produz o método dos Gradientes Conjugados (GC).

3. **Condição de Petrov-Galerkin:** Busca uma solução aproximada $x_m \in \mathcal{K}_m(A, r_0)$, impondo que o vetor resíduo $b - Ax_m$ seja ortogonal a um outro subespaço de Krylov $m-$dimensional adequado. Ou seja, $\mathcal{L}_m \neq \mathcal{K}_m$. Na verdade, a escolha usual é $\mathcal{L}_m = \mathcal{K}_m(A^T, r_0^*)$, onde r_0^* é um vetor tal que $(r_0, r_0^*) \neq 0$. Desta escolha surgem, por exemplo, os métodos BiCG, CGS, BiCGSTAB, GPBiCG, e TFBiCG.

Vale lembrar que para tratarmos dos métodos pertencentes a essas classes necessitamos de bases adequadas para o subespaço de Krylov envolvido. Essas bases podem ser eficientemente construídas através dos procedimentos de Arnoldi e de Lanczos que serão apresentados em seguida.

3.1 Procedimento de Arnoldi

O procedimento de Arnoldi é um algoritmo que constrói uma base ortogonal para o subespaço de Krylov $\mathcal{K}_m$. Em sua formulação podem ser usados os métodos de ortogonalização Gram-Schmidt padrão, Gram-Schmidt modificado ou Householder [23]. Por simplicidade, assumimos aritmética exata e analisamos o procedimento de Arnoldi com o método de Gram-Schmidt padrão. Porém, por razões computacionais, utilizamos na prática o procedimento de Arnoldi com o método Gram-Schmidt modificado, que é matematicamente equivalente.

Os algoritmos dessas duas variantes do procedimento de Arnoldi são dados a seguir:

ALGORITMO 3.1: Procedimento de Arnoldi com Gram-Schmidt padrão.

1. Escolha um vetor v_1 de norma 1
2. Para $j = 1, 2, \ldots, m$ faça:
3. Calcule $h_{ij} = (Av_j, v_i)$ para $i = 1, \ldots, j$.
4. Calcule $w_j = Av_j - \sum_{i-1}^{j} h_{ij} v_i$
5. $h_{j+1,j} = \| w_j \|_2$
6. Se $h_{j+1,j} = 0$, então pare
7. $v_{j+1} = w_j / h_{j+1,j}$
8. fim.

ALGORITMO 3.2: Procedimento de Arnoldi com Gram-Schmidt modificado.

1. Escolha um vetor v_1 de norma 1
2. Para $j = 1, \ldots, m$ Faça:
3. Calcule $w_j = Av_j$
4. Para $i = 1, 2, \ldots, j.$ faça:
5. $h_{ij} = (w_j, v_i)$

6. $w_j = w_j - h_{ij}v_i$
7. fim
8. $h_{j+1,j} = \| w_j \|_2$. Se $h_{j+1,j} = 0$ pare
9. $v_{j+1} = w_j / h_{j+1,j}$
10. Fim.

É importante frisar que, do ponto de vista numérico, o algoritmo 3.2 é mais confiável. Porém, existem casos onde os cancelamentos realizados durante o processo de ortogonalização são tão severos que mesmo a escolha deste algoritmo torna-se inadequada. Para estes casos pode ser explorada uma formulação utilizando o algoritmo de Householder como técnica de ortogonalização (veja [23]).

Agora, a partir do algoritmo 3.1 alguns resultados muito úteis podem ser provados.

Proposição 3.1.1. *Assuma que o algoritmo 3.1 não é interrompido antes do m-ésimo passo. Então os vetores $v_1, v_2, \ldots, v_m$ formam uma base ortonormal do subespaço de Krylov*

$$\mathcal{K}_m = Span\{v_1, Av_1, \ldots, A^{m-1}v_m\}.$$

Prova:

Os vetores $v_j,\ j = 1, 2, \ldots, m$ são ortonormais por construção. Resta provar que cada vetor v_j é da forma $\psi_{j-1}(A)v_1$ onde ψ_{j-1} é um polinômio de grau $j-1$. Por indução temos:

Para $j = 1$ o resultado é imediato pois $v_1 = \psi_0(A)v_1$ com $\psi_0(t) = 1$. Suponhamos que o resultado é válido para todo inteiro $\leq j$ e consideremos v_{j+1}. Das linhas 7 e 4 do algoritmo temos:

$$\begin{aligned} h_{j+1,j}v_{j+1} &= Av_j - \sum_{i=1}^{j} h_{ij}v_i \\ &= A\psi_{j-1}(A)v_1 - \sum_{i=1}^{j} h_{i,j}\psi_{i-1}(A)v_1. \end{aligned} \tag{3.2}$$

o que mostra que v_{j+1} pode ser escrito como $\psi_j(A)v_1$ onde ψ_j é um polinômio de grau j o que completa a prova.

■

Proposição 3.1.2. *Denotando por V_m a matriz $n \times m$ cujas colunas são os vetores $v_1, v_2, \ldots, v_m$, por $\overline{H}_m$ a matriz de Hessenberg $(m+1) \times m$ cujos elementos h_{ij} não nulos são definidos pelo algoritmo 3.1, e por H_m a matriz obtida de $\overline{H}_m$ pela eliminação de sua última linha, as seguintes relações são válidas*

$$\begin{aligned} AV_m &= V_m H_m + w_m e_m^T && (3.3)\\ &= V_{m+1}\overline{H}_m, && (3.4)\\ V_m^T A V_m &= H_m. && (3.5) \end{aligned}$$

Prova:

Das linhas 4 e 7 do algoritmo 3.1 temos que,

$$\begin{aligned} Av_j &= \sum_{i=1}^{j} h_{ij} v_i + w_j && (3.6)\\ &= \sum_{i=1}^{j} h_{ij} v_i + h_{j+1,j} v_{j+1} && (3.7)\\ &= \sum_{i=1}^{j+1} h_{ij} v_i \qquad j = 1, 2, \ldots, m && (3.8) \end{aligned}$$

o que prova a relação (3.4).

A relação (3.3) é uma reformulação matricial de (3.6). Por fim, multiplicando ambos os lados de (3.3) por V_m^T e explorando a ortogonalidade dos vetores $\{v_1, \ldots, v_m\}$

$$V_m^T A V_m = V_m^T V_m H_m + V_m^T w_m e_m^T = H_m$$

obtemos (3.5).

■

3.2 O método GMRES

O Método Mínimo Resíduo Generalizado (GMRES) [25] é um método de projeção que usa $\mathcal{K} = \mathcal{K}_m$ e $\mathcal{L} = A\mathcal{K}_m$, onde $\mathcal{K}_m$ é o subespaço de Krylov de dimensão m com $v_1 = r_0 / \parallel r_0 \parallel_2$. Este método calcula a solução aproximada da forma $x_0 + z$ que minimiza a norma de resíduo sobre $z \in \mathcal{K}_m$, usando uma base gerada pelo método de Arnoldi, Algoritmo 3.2.

Da proposição 3.1.1 vem que qualquer vetor x em $x_0 + \mathcal{K}_m$ pode ser escrito como

$$x = x_0 + V_m y, \tag{3.9}$$

onde y é um vetor de dimensão igual a m.

Definindo

$$J(y) = \parallel b - Ax \parallel_2 \tag{3.10}$$

e utilizando a relação (3.4) e as propriedades de ortogonalidade de V_m, obtemos

$$\begin{aligned} b - Ax &= b - Ax_0 - AV_m y \\ &= r_0 - V_{m+1}\overline{H}_m y \\ &= V_{m+1}(\beta e_1 - \overline{H}_m y), \end{aligned} \tag{3.11}$$

onde $\beta = \parallel r_0 \parallel_2$, e e_1 é a primeira coluna da matriz identidade $(m+1)\times(m+1)$.

Portanto, visto que os vetores-coluna de V_{m+1} são ortonormais, temos

$$J(y) \equiv \parallel b - Ax \parallel_2 = \parallel \beta e_1 - \overline{H}_m y \parallel_2 . \tag{3.12}$$

A solução aproximada obtida pelo método GMRES é o único vetor de $x_0 + \mathcal{K}_m$ que minimiza (3.10). Esta aproximação pode ser obtida fazendo $x_m = x_0 + V_m y_m$ onde y_m minimiza a função $J(y) = \parallel b - Ax \parallel_2 = \parallel \beta e_1 - \overline{H}_m y \parallel_2$, isto é,

$$x_m = x_0 + V_m y_m, \tag{3.13}$$

onde

$$y_m = arg \min_{y \in I\!R^m} \parallel \beta e_1 - \overline{H}_m y \parallel_2 . \tag{3.14}$$

Esta seqüência de operações produz o algoritmo do Método GMRES que está descrito a seguir.

ALGORITMO 3.3: Método GMRES.

1. Calcule $r_0 = b - Ax_0, \beta = \| r_0 \|_2, e\ v_1 = r_0/\beta$
2. Defina a matriz $(m+1) \times m\ \overline{H}_m = \{h_{ij}\}_{1 \leq i \leq m+1;\ 1 \leq j \leq m}$ e faça $\overline{H}_m = 0$
3. Para $j = 1, 2, \ldots, m$ Faça:
4. Calcule $w_j = Av_j$
5. Para $i = 1, \ldots, j$ faça:
6. $h_{ij} = (w_j, v_i)$
7. $w_j = w_j - h_{ij}v_i$
8. fim
9. $h_{j+1,j} = \| w_j \|_2$. Se $h_{j+1,j} = 0$ faça $m = j$ a vá para o passo 12
10. $v_{j+1} = w_j/h_{j+1,j}$
11. Fim
12. Calcule y_m que minimiza $\| \beta e_1 - \overline{H}_m y \|_2$ e $x_m = x_0 + V_m y_m$.

3.2.1 Aspectos práticos da implementação do GMRES

A aplicação do método GMRES através do algoritmo 3.3 requer, na linha 12, a solução de um problema de mínimos quadrados. Em razão da estrutura especial da matriz $\overline{H}_m$ a solução deste problema de mínimos quadrados é relativamente simples. Através do uso de rotações planas os elementos subdiagonais da matriz $\overline{H}_m$ são eliminados, transformando-a em uma forma triangular superior denotada por $\overline{R}_m$. Dessa forma temos

$$\overline{R}_m = Q_m \overline{H}_m,$$

onde Q_m denota o produto das sucessivas rotações aplicadas à matriz $\overline{H}_m$ para eliminar seus elementos $h_{j+1,j}$, para $j = 1, \ldots, m$.

Para uma descrição mais detalhada dessas operações, definimos as matrizes de rotação,

$$\Omega_i = \begin{pmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & c_i & s_i & & & \\ & & & -s_i & c_i & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{pmatrix} \longleftarrow linha\ i \tag{3.15}$$

com $c_i^2 + s_i^2 = 1$. Se m passos do algoritmo 3.3 forem realizados então essas matrizes terão dimensão $(m+1) \times (m+1)$.

Nas matrizes de rotação os coeficientes c_i e s_i devem ser escolhidos de tal modo que o elemento da posição $(i+1, i)$, na matriz resultante do produto de Ω_i por $\overline{H}_m$ seja nulo, ou seja, o produto $\Omega_i \overline{H}_m$ deve eliminar o elemento $h_{i+1,i}$ em $\overline{H}_m$. Dessa forma, se, por exemplo, $i = 1$ temos que:

$$-s_1 h_{11} + c_1 h_{21} = 0 \Rightarrow s_1 = c_1 \frac{h_{21}}{h_{11}}, \tag{3.16}$$

temos ainda

$$s_1^2 + c_1^2 = 1. \tag{3.17}$$

Assim, substituindo (3.16) em (3.17), obtemos

$$s_1 = \frac{h_{21}}{\sqrt{h_{11}^2 + h_{21}^2}}, \quad c_1 = \frac{h_{11}}{\sqrt{h_{11}^2 + h_{21}^2}} \tag{3.18}$$

Portanto, a premultiplicação da matriz de Hessenberg $\overline{H}_m$ e do correspondente vetor do lado direito $\overline{g}_0 \equiv \beta e_1$ pelas matrizes de rotação, produz uma forma triangular superior de $\overline{H}_m$ de dimensão $(m+1) \times m$. Como exemplo, se $m = 5$, nós teremos

$$\overline{H}_5 = \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} \\ h_{21} & h_{22} & h_{23} & h_{24} & h_{25} \\ & h_{32} & h_{33} & h_{34} & h_{35} \\ & & h_{43} & h_{44} & h_{45} \\ & & & h_{54} & h_{55} \\ & & & & h_{65} \end{pmatrix}, \quad \overline{g}_0 = \begin{pmatrix} \beta \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{3.19}$$

Então, premultiplicando $\overline{H}_5$ por

$$\Omega_1 = \begin{pmatrix} c_1 & s_1 & & & & \\ -s_1 & c_1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{pmatrix}$$

com c_1 e s_1 como em (3.18).

Obtemos a matriz e o vetor do lado direito

$$\overline{H}_5^{(1)} = \begin{pmatrix} h_{11}^{(1)} & h_{12}^{(1)} & h_{13}^{(1)} & h_{14}^{(1)} & h_{15}^{(1)} \\ & h_{22}^{(1)} & h_{23}^{(1)} & h_{24}^{(1)} & h_{25}^{(1)} \\ & h_{32} & h_{33} & h_{34} & h_{35} \\ & & h_{43} & h_{44} & h_{45} \\ & & & h_{54} & h_{55} \\ & & & & h_{65} \end{pmatrix}, \quad \overline{g}_1 = \begin{pmatrix} c_1\beta \\ -s_1\beta \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{3.20}$$

Pode-se proceder da mesma forma para eliminar h_{32}. A premultiplicação agora é feita com Ω_2 tomando

$$s_2 = \frac{h_{32}}{\sqrt{(h_{22}^{(1)})^2 + h_{32}^2}}, \quad c_1 = \frac{h_{22}^{(1)}}{\sqrt{(h_{22}^{(1)})^2 + h_{32}^2}}.$$

Repetindo m vezes este processo de eliminação transformamos o problema de mínimos quadrados original em um problema envolvendo a matriz e o vetor do lado direito,

$$\overline{H}_5^{(5)} = \begin{pmatrix} h_{11}^{(5)} & h_{12}^{(5)} & h_{13}^{(5)} & h_{14}^{(5)} & h_{15}^{(5)} \\ & h_{22}^{(5)} & h_{23}^{(5)} & h_{24}^{(5)} & h_{25}^{(5)} \\ & & h_{33}^{(5)} & h_{34}^{(5)} & h_{35}^{(5)} \\ & & & h_{44}^{(5)} & h_{45}^{(5)} \\ & & & & h_{55}^{(5)} \\ & & & & 0 \end{pmatrix}, \quad \overline{g}_5 = \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \\ \gamma_5 \\ \gamma_6 \end{pmatrix}. \tag{3.21}$$

Generalizando, os escalares c_i e s_i da i-ésima rotação Ω_i são definidos como

$$s_i = \frac{h_{i+1,i}}{\sqrt{(h_{ii}^{(i-1)})^2 + h_{i+1,i}^2}}, \quad c_i = \frac{h_{ii}^{(i-1)}}{\sqrt{(h_{ii}^{(i-1)})^2 + h_{i+1,i}^2}}. \tag{3.22}$$

Definindo Q_m o produto das matrizes de rotação Ω_i,

$$Q_m = \Omega_m \Omega_{m-1} \ldots \Omega_1 \tag{3.23}$$

e,

$$\overline{R}_m = \overline{H}_m^{(m)} = Q_m \overline{H}_m \tag{3.24}$$

$$\overline{g}_m = Q_m(\beta e_1) = (r_1, \ldots r_{m+1})^T \tag{3.25}$$

temos para o problema original

$$\min \parallel \beta e_1 - \overline{H}_m y \parallel_2 = \min \parallel Q_m^{-1}(\overline{g}_m - \overline{R}_m y) \parallel_2 . \tag{3.26}$$

Note porém que

$$\Omega_i^T \cdot \Omega_i = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & c_i & -s_i & & \\ & & & s_i & c_i & & \\ & & & & & 1 & \\ & & & & & & \ddots \\ & & & & & & & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & c_i & s_i & & \\ & & & -s_i & c_i & & \\ & & & & & 1 & \\ & & & & & & \ddots \\ & & & & & & & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & c_i^2 + s_i^2 & 0 & & & \\ & & & 0 & s_i^2 + c_i^2 & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{pmatrix} = I_{m+1}, \quad i = 1, \ldots, m,$$

onde I_{m+1} é a matriz identidade de ordem $m+1$.Logo,

$$Q_m^T Q_m = (\Omega_m \Omega_{m-1} \ldots \Omega_1)^T (\Omega_m \Omega_{m-1} \ldots \Omega_1) = I_{m+1}$$

e portanto, Q_m é unitária. Assim, a solução do problema de mínimos quadrados acima é obtida simplesmente resolvendo um sistema triangular resultante da eliminação da última linha da matriz $\overline{H}_m$ e da última componente do vetor do lado direito $\overline{g}_m$ em (3.21). Dessa forma a equação (3.26) torna-se,

$$\min \parallel \beta e_1 - \overline{H}_m y \parallel_2 = \min \parallel (\overline{g}_m - \overline{R}_m y) \parallel_2 .$$

Esta maneira de resolver o problema de mínimos quadrados soluciona ainda um outro problema de algoritmo 3.3 que é a dificuldade de determinar quando parar. Observe que o algoritmo não calcula explicitamente a solução aproximada x_m em cada passo. Utilizando, no entanto, as matrizes de rotação Ω_i para resolver o problema de mínimos quadrados temos claramente que, para a solução y_*, o "resíduo" $\|\beta e_1 - \overline{H}_m y_*\|$ é nada mais que o último elemento do vetor do lado direito, isto é, o termo γ_6 ilustrado em (3.21). Esses resultados são formalizados na proposição abaixo.

Proposição 3.2.1. *Seja $\Omega_i, i = 1, \ldots, m$ as matrizes de rotação usadas para transformar $\overline{H}_m$ em uma forma triangular superior, e $\overline{R}_m, \overline{g}_m = (\gamma_1, \ldots, \gamma_{m+1})^T$ a matriz e o vetor do lado direito resultantes, como definido em (3.24), (3.25). Denote por R_m a matriz triangular superior $m \times m$ obtida de $\overline{R}_m$ pela eliminação de sua última linha e por g_m o vetor de dimensão m obtido de $\overline{g}_m$ pela*

eliminação de sua última componente. Então,

1. o posto de AV_m *é igual ao posto de* R_m*. Em particular, se* $r_{mm} = 0$ *então* A *é singular.*

2. o vetor y_m *que minimiza* $\|\beta e_1 - \overline{H}_m y\|_2$ *é dado por* $y_m = R_m^{-1} g_m$*.*

3. o vetor resíduo no passo m *satisfaz*

$$b - Ax_m = V_{m+1}(\beta e_1 - \overline{H}_m y_m) = V_{m+1} Q_m^T (\gamma_{m+1} e_{m+1}) \tag{3.27}$$

e como resultado,

$$\|b - Ax_m\|_2 = |\gamma_{m+1}|. \tag{3.28}$$

A prova desta proposição pode ser encontrada em [23].

Na prática esse processo é implementado de maneira progressiva, em cada passo do algorítimo do método GMRES. Assim, o elemento h_{21} da matriz de Hessenberg $\overline{H}$ é eliminado, através da aplicação da matriz de rotação Ω_1, imediatamente após a criação da primeira coluna da matriz $\overline{H}$, no passo 1 do algoritmo 3.3. No segundo passo, após a criação da segunda coluna da matriz $\overline{H}$, aplica-se a matriz de rotação Ω_1 e posteriormente a rotação Ω_2 para eliminar o elemento h_{32}. Generalizando, no passo j do algorítmo aplica-se as rotações $\Omega_1, \Omega_2, \ldots, \Omega_{j-1}$ e finalmente a rotação Ω_j para eliminar o elemento $h_{j+1,j}$.

É claro que as rotações devem ser aplicadas ao vetor do lado direito, e isto também é feito progressivamente. Como resultado, o passo 1 produz as relações apresentadas em (3.20)

$$\gamma_1 = c_1\beta \;\text{ e }\; \gamma_2 = -s_1\beta,$$

onde c_1 e s_1 são os escalares da rotação Ω_1.

Procedendo de forma progressiva, no passo j obtem-se

$$\gamma_j = c_j\gamma_j, \quad \gamma_{j+1} = -s_j\gamma_j.$$

Assim, a norma do resíduo que é dada por $|\gamma_{j+1}| = |-s_j\gamma_j|$ fica disponível após a aplicação da rotação Ω_j em cada passo j do algoritmo 2.3. Em particular, se $s_j = 0$ então a norma do resíduo é igual a zero e a solução é exata nesse passo.

3.2.2 Interrupção anormal do processo no método GMRES (*"Breakdowns"*).

Uma cuidadosa observação do algoritmo 3.3 nos permite notar que a única possibilidade de ocorrer uma interrupção anormal do processo ("Breakdown") surge no laço externo, se $h_{j+1,j} = 0$ em um dado passo j. Nesta situação, o algoritmo para porque o próximo vetor não pode ser gerado. Porém, neste caso, o vetor resíduo é nulo, isto é, o algoritmo encontrará a solução exata neste passo. Na verdade, a recíproca também é válida.Ou seja, se o algoritmo para no passo j, com $b - Ax_j = 0$, então $h_{j+1,j} = 0$.

Proposição 3.2.2. *Seja A uma matriz não singular. Então, o algoritmo GMRES é interrompido no passo j, i.e., $h_{j+1,j} = 0$ se somente se a solução aproximada é exata.*

Prova:

A condição necessária é satisfeita uma vez que de acordo com (3.22)

$$s_j = \frac{h_{j+1,j}}{\sqrt{(h_{jj}^{(j-1)})^2 + h_{j+1,h}^2}} = 0$$

então, das relações

$$|\gamma_{j+1}| = |-s_j\gamma_j| \quad e \quad \|b - Ax_j\|_2 = |\gamma_{j+1}|, \tag{3.29}$$

vem que $r_j = 0$.

Para a condição suficiente usamos novamente as relações (3.29). Temos

$$\|b - Ax_j\|_2 = |\gamma_{j+1}| = |-s_j\gamma_j|$$

Logo, se a solução é exata no passo j e não no passo $j-1$ resulta que $s_j = 0$. Portanto de (3.22) vem que $h_{j+1,j} = 0$.

■

3.3 Variantes do método GMRES

No que diz respeito à memória computacional, o algoritmo GMRES requer o uso de $m+1$ vetores de dimensão n para salvar a matriz V_{m+1} e de seis vetores adicionais. São três vetores de dimensão n para armazenar o vetor resíduo atual, o vetor do lado direito do sistema original, e o produto da matriz A por um vetor, e outros três de dimensão m, para armazenar os escalares c_i e s_i das rotações Ω_i, e o vetor do lado direito do sistema triangular obtido da matriz de Hessenberg,

$$(m+1)n + 3n + 3m.$$

Além disso, a matriz de Hessenberg H_m deve ser armazenada, ocupando um espaço de aproximadamente

$$\frac{m^2}{2}.$$

Portanto, a memória total utilizada pelo GMRES durante seu processo é de aproximadamente

$$(m+4)n + \frac{m}{2}(6+m).$$

Como na maioria dos casos m é pequeno com relação a n, o custo é dominado pelo primeiro termo da equação acima. Do ponto de vista prático, este algoritmo pode se tornar impraticável à medida que m cresce. Por outro lado, para um n muito grande, o valor máximo permitido para m fica muito limitado. Existem duas soluções para este problema: a primeira é reiniciar o algoritmo periodicamente, e a segunda é baseada no truncamento do processo de ortogonalização.

3.3.1 O método GMRES com reinício

Nesta variação do método GMRES o valor de m é escolhido de acordo com as limitações de memória da máquina. O processo então é realizado normalmente até que sejam dados m passos. A partir daí o vetor solução atual x_m é usado como estimativa inicial e o processo é reiniciado.

ALGORITMO 3.4: Método GMRES com reinício.

1. Calcule $r_0 = b - Ax_0, \beta = \| r_0 \|_2, e\ v_1 = r_0/\beta$
2. Gere a base de Arnoldi e a matriz $\overline{H}_m$ usando o algoritmo 3.2
3. Comece com v_1
4. Calcule y_m que minimiza $\|\beta e_1 - \overline{H}_m y\|_2$ e $x_m = x_0 + V_m y_m$
5. Se convergiu então pare, senão faça $x_0 = x_m$ e vá para Passo 1.

Observe que todos os detalhes de implementação discutidos na subseção 3.2.1 podem ser aplicados aqui, produzindo a norma do vetor resíduo em cada subpasso j do processo sem a necessidade de computar a solução x_j.

Uma dificuldade bem conhecida do GMRES com reinício e que ele pode ficar estagnado quando a matriz não é definida positiva. No caso do método GMRES padrão, algoritmo 3.3, a convergência é garantida em no máximo n passos, o que pode ser impraticável se n for muito grande. É claro que para reduzir a quantidade de passos necessários podem ser usadas algumas técnicas de pré-condicionamento, que serão vistas mais adiante.

3.3.2 Truncamento do processo de ortogonalização

A segunda alternativa para o GMRES é truncar o procedimento de Arnoldi. Um número inteiro k é selecionado e a seguinte ortogonalização incompleta é realizada.

ALGORITMO 3.5: Processo de Ortogonalização Incompleto

1. Para $j = 1, 2, \ldots, m$ Faça:
2. Calcule $w = Av_j$
3. Para $i = \max\{1, j-k+1\}, \ldots, j$ faça:
4. $h_{ij} = (w, v_i)$
5. $w = w - h_{ij} v_i$

6. fim

7. Calcule $h_{j+1,j} = \| w \|_2$ e $v_{j+1} = w/h_{j+1,j}$

8. Fim.

O número k, assim como m, é escolhido de acordo com as limitações de memória computacional.

Note que agora é necessário armazenar somente os k vetores v_i anteriores. Os outros não são necessários no processo e por isso podem ser descartados. Porém, a dificuldade é que quando a solução é computada pela fórmula (3.13) todos os vetores $v_i, i = 1, \ldots, m$ são necessários. Uma alternativa é desenvolver uma formulação através da qual a solução atual x_m possa ser atualizada através da solução anterior x_{m-1} e que utilize um pequeno número de vetores. Isto é possível através de uma formulação progressiva da solução que fornece um algoritmo denominado *"Direct Quasi GMRES"* (DQGMRES)[27], que será deduzido agora.

3.3.3 O método DQGMRES

A matriz de Hessenberg $\overline{H}_m$ produzida pelo processo de ortogonalização incompleto tem uma estrutura de banda com $k+1$ diagonais. Por exemplo, quando $k = 2$ e $m = 5$, ela tem forma

$$\overline{H}_5 = \begin{pmatrix} h_{11} & h_{12} & & & \\ h_{21} & h_{22} & h_{23} & & \\ & h_{32} & h_{33} & h_{34} & \\ & & h_{43} & h_{44} & h_{45} \\ & & & h_{54} & h_{55} \end{pmatrix}, \quad \overline{g}_0 = \begin{pmatrix} \beta \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{3.30}$$

então a premultiplicação pelas matrizes de rotação Ω_i, como descrito na seção 2.3, além de eliminar os elementos sub-diagonais introduzirá uma diagonal adicional na parte superior da matriz de Hessenberg. Para o caso acima, o

problema de mínimos quadrados resultante é

$$\overline{R}_5 y = \overline{g}_5$$

com,

$$\overline{R}_5 = \begin{pmatrix} r_{11} & r_{12} & r_{13} & & \\ & r_{22} & r_{23} & r_{24} & \\ & & r_{33} & r_{34} & r_{35} \\ & & & r_{44} & r_{45} \\ & & & & r_{55} \\ & & & & 0 \end{pmatrix}, \quad \overline{g}_5 = \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \\ \gamma_5 \\ \gamma_6 \end{pmatrix}. \tag{3.31}$$

A solução aproximada é dada por

$$x_m = x_0 + V_m R_m^{-1} g_m$$

onde R_m e g_m são obtidos eliminando a última linha de $\overline{R}_m$ e $\overline{g}_m$, respectivamente.

Definindo

$$P_m \equiv V_m R_m^{-1}$$

então,

$$x_m = x_0 + P_m g_m.$$

Observe ainda que

$$g_m = \begin{bmatrix} g_{m-1} \\ \gamma_m \end{bmatrix}$$

na qual

$$\gamma_m = c_m \gamma_m^{(m-1)},$$

onde c_m é um dos escalares da rotação Ω_m, definido como em (3.22), e $\gamma_m^{(m-1)}$ é a última componente do vetor $\overline{g}_{m-1}$, isto é, o vetor do lado direito antes de aplicar a m-ésima rotação. Deste modo, x_m pode ser atualizado em cada passo, pela relação

$$x_m = x_{m-1} + \gamma_m p_m,$$

o que produz o seguinte algorítmo

ALGORITMO 3.6: O método DQGMRES

1. Calcule $r_0 = b - Ax_0, \gamma_1 = \| r_0 \|_2, e\ v_1 = r_0/\gamma_1$
2. Para $m = 1, 2, \ldots,$até convergência faça:
3. Calcule $h_{im}, i = \max\{1, m-k+1\}, \ldots, m$ e
4. v_{m+1} como nas linhas 2 a 6 do algoritmo 3.5
5. Atualize a fatorização QR de $\overline{H}_m$, isto é,
6. Aplique $\Omega_i, i = m-k, \ldots, m-1$ na $m-$ésima coluna de $\overline{H}_m$
7. Calcule os coeficientes de rotação c_m e s_m por (3.22)
8. Aplique Ω_m para $\overline{H}_m$ e $\overline{g}_m$,isto é,calcule:
9. $\gamma_{m+1} = -s_m \gamma_m$
10. $\gamma_m = c_m \gamma_m$
11. $h_{mm} = c_m h_{mm} + s_m h_{m+1,m}$
12. $p_m = (v_m - \sum_{i=m-k}^{m-1} h_{im} p_i)/h_{mm}$
13. $x_m = x_{m-1} + \gamma_m p_m$
14. se $|\gamma_{m+1}|$ é pequeno o suficiente então pare
15. fim.

Este algoritmo não minimiza a norma do vetor resíduo sobre $x_0 + \mathcal{K}_m$. Na verdade, ele tenta realizar uma minimização aproximada. A fórmula (3.27) ainda é válida uma vez que não depende da ortogonalidade dos vetores $v_i's$. Porém, a fórmula (3.28) não é mais válida. No entanto, os resultados disponíveis na literatura, (veja, por exemplo [23]), indicam que na prática o valor $|\gamma_{m+1}|$ permanece como uma boa estimativa para a norma do vetor resíduo.

3.4 O Procedimento de Lanczos simétrico

Quando a matriz A é simétrica o procedimento de Arnoldi, descrito na seção 3.1, pode ser sensivelmente simplificado. Essa simplificação dá origem ao chamado algoritmo de Lanczos simétrico que produz uma matriz de Hessenberg H_m simétrica tridiagonal, como mostra o seguinte teorema extraído de [23].

Teorema 3.1. *Assuma que o método de Arnoldi é aplicado a uma matriz simétrica real A. Então os coeficientes h_{ij} gerados pelo algoritmo são tais que*

$$h_{ij} = 0, \quad para \quad 1 \leq i < j - 1 \tag{3.32}$$

$$h_{j,j+1} = h_{j+1,j}, \quad j = 1, 2, \ldots, m. \tag{3.33}$$

Em outras palavras, a matriz H_m obtida do processo de Arnoldi é tridiagonal e simétrica.

Prova:

Sabemos de (3.5) que $H_m = V_m^T A V_m$ é uma matriz de Hessenberg, isto é, uma matriz onde os elementos da parte triangular inferior são nulos, exceto aqueles da subdiagonal. Além disso, H_m é uma matriz simétrica pois $H_m = (V_m^T A V_m)^T = V_m^T A V_m = H_m$. Portanto, H_m deve ser uma matriz simétrica tridiagonal.

■

Introduzindo a seguinte notação

$$\alpha_j = h_{jj}, \quad \beta_j = h_{j-1,j},$$

e denotando por T_m a matriz tridiagonal resultante,

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \cdot & \cdot & \cdot & \\ & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \beta_m & \alpha_m \end{pmatrix}, \tag{3.34}$$

obtemos o algoritmo de Lanczos, que pode ser visto como um caso particular do algoritmo de Arnoldi para o caso simétrico. Uma formulação desse algoritmo, utilizando o processo de ortogonalização Gram-Schmidt modificado é apresentada a seguir.

ALGORITMO 3.7: Procedimento de Lanczos simétrico.

1. Escolha um vetor inicial v_1 de norma 1. Atribua $\beta_1 \equiv 0, v_0 \equiv 0$
2. Para $j = 1, \ldots, m$ Faça:
3. Calcule $w_j = Av_j - \beta_j v_{j-1}$
4. $\alpha_j = (w_j, v_j)$
5. $w_j = w_j - \alpha_j v_j$
6. $\beta_{j+1} = \| w_j \|_2$. Se $\beta_{j+1} = 0$ pare
7. $v_{j+1} = w_j / \beta_{j+1}$
10. Fim.

Na prática, além da matriz de Hessenberg H_m ser tridiagonal outra importante diferença entre este algoritmo e o algoritmo de Arnoldi é que apenas três vetores precisam ser armazenados, a menos que algum processo de reortogonalização seja aplicado.

3.5 O método dos Gradientes Conjugados

O método dos Gradientes Conjugados (GC) foi desenvolvido para resolver sistemas lineares onde a matriz A é simétrica definida positiva (SDP). Trata-se de uma técnica de projeção ortogonal na qual $\mathcal{L} = \mathcal{K} = \mathcal{K}_m(A, r_0)$, onde $\mathcal{K}_m(A, r_0)$ é o subespaço de Kyrlov gerado pelos vetores $\{r_0, Ar_0, \ldots, A^{m-1}r_0\}$ e $r_0 = b - Ax_0$.

3.5.1 A dedução do método GC

Aplicando o procedimento de Lanczos para gerar uma base para o subespaço de Krylov $\mathcal{K}_m(A, r_0)$ obtemos as mesmas relações (3.4) e (3.5) porém agora com $\overline{H}_m$ tridiagonal e denotada por $\overline{T}_m$,

$$AV_m = V_{m+1}\overline{T}_m \tag{3.35}$$

$$V_m^T A V_m = T_m, \tag{3.36}$$

onde T_m é a versão tridiagonal de H_m.

Assim, para m-ésima coluna de V_m, temos

$$Av_m = \beta_{m+1}v_{m+1} + \alpha_m v_m + \beta_{m-1}v_{m-1} \tag{3.37}$$

que é a relação central do algoritmo 3.7.

Impondo a condição de Galerkin, o novo vetor resíduo r_m, no passo m, deve ser ortogonal ao subespaço gerado pelos vetores $v_1,\ v_2, \ldots,\ v_m$, isto é,

$$r_m = b - Ax_m \quad \perp \quad V_m. \tag{3.38}$$

Temos ainda que para $x_m \in x_0 + \mathcal{K}_m(A, r_0)$

$$x_m = x_0 + V_m y_m \tag{3.39}$$

onde y_m é um vetor de dimensão m. Então substituindo (3.39) em (3.38) vem

$$\begin{aligned} r_m = b - Ax_m &= b - Ax_0 - AV_m y_m \\ &= r_0 - AV_m y_m \quad \perp \quad V_m. \end{aligned} \tag{3.40}$$

que implica

$$V_m^T r_0 = V_m^T A V_m y_m \tag{3.41}$$

Agora, usando $v_1 = r_0/\beta$, com $\beta = \|r_0\|_2$, e a relação (3.36), obtemos

$$V_m^T(\beta v_1) = T_m y_m \Rightarrow \beta e_1 = T_m y_m. \tag{3.42}$$

Assim, de (3.42) e (3.39) resulta a forma da solução aproximada

$$x_m = x_0 + V_m T_m^{-1}(\beta e_1) \tag{3.43}$$

Portanto, através dessa formulação é possível encontrar uma solução aproximada para o sistema linear $Ax = b$ onde A é SDP. Note porém, que esta formulação apresenta algumas dificuldades: Em primeiro lugar o cálculo da solução aproximada x_m é dependente de um parâmetro m que é a dimensão do subespaço de Krylov. Dessa forma a norma do resíduo não pode ser calculada em cada iteração dificultando a escolha do momento de parar o processo. Além disso, o cálculo de x_m através de (3.43) requer o uso de todos os vetores v_i, $i = 1, ..., m$, muitos dos quais não foram armazenados pelo algoritmo 3.7. Uma alternativa então é explorar a estrutura tridiagonal da matriz T_m buscando uma formulação progressiva que não exija o uso de todos os vetores da base V_m, e que forneça a norma do vetor resíduo em cada passo do processo iterativo.

Essa formulação progressiva tem origem na fatorização LU da matriz tridiagonal T_m. Em primeiro lugar escreveremos T_m como $T_m = L_m U_m$, onde L_m é bidiagonal inferior, com elementos iguais a 1 na diagonal principal, e U_m é bidiagonal superior. Fazendo por exemplo m=5, essa fatorização tem a seguinte forma

$$T_m = \begin{pmatrix} 1 & & & & \\ \lambda_2 & 1 & & & \\ & \lambda_3 & 1 & & \\ & & \lambda_4 & 1 & \\ & & & \lambda_5 & 1 \end{pmatrix} \times \begin{pmatrix} \eta_1 & \beta_2 & & & \\ & \eta_2 & \beta_3 & & \\ & & \eta_3 & \beta_4 & \\ & & & \eta_4 & \beta_5 \\ & & & & \eta_5 \end{pmatrix} \tag{3.44}$$

Usando (3.44), a equação (3.43) torna-se

$$x_m = x_0 + V_m U_m^{-1} L_m^{-1}(\beta e_1). \tag{3.45}$$

Então, tomando

$$P_m = V_m U_m^{-1} \tag{3.46}$$

e

$$z_m = L_m^{-1}\beta e_1 \tag{3.47}$$

obtemos uma nova expressão para x_m

$$x_m = x_0 + P_m z_m. \tag{3.48}$$

Note que agora é possível explorar a estrutura simples da matriz U_m e calcular facilmente cada coluna p_m da matriz P_m somente em função da coluna anterior p_{m-1} e do vetor v_m. Usando (3.46) temos que

$$P_m U_m = V_m = [p_1\ p_2\ \ldots\ p_m] \begin{bmatrix} \eta_1 & \beta_2 & & & \\ & \eta_2 & \beta_3 & & \\ & & \ddots & \ddots & \\ & & & \eta_{m-1} & \beta_m \\ & & & & \eta_m \end{bmatrix} = [v_1\ v_2\ \ldots\ v_m]$$

que produz, para a coluna p_m

$$\begin{aligned} \beta_m p_{m-1} + \eta_m p_m &= v_m \\ \Rightarrow\ p_m &= \eta_m^{-1}[v_m - \beta_m p_{m-1}]. \end{aligned} \tag{3.49}$$

De forma análoga para z_m fazemos,

$$L_m z_m = \beta e_1 \tag{3.50}$$

que implica

$$z_1 = \beta \tag{3.51}$$

e

$$z_m = \begin{bmatrix} z_{m-1} \\ \zeta_m \end{bmatrix} \tag{3.52}$$

onde $\zeta_m = -\lambda_m \zeta_{m-1}$.

Como resultado, uma nova fórmula para x_m é dada por

$$x_m = x_{m-1} + \zeta_m p_m \tag{3.53}$$

onde p_m é definido como em (3.49).

Nessa formulação progressiva os coeficientes λ_m e η_m surgem do processo de eliminação de Gauss usado implicitamente na matriz tridiagonal T_m. Na prática esse processo é realizado sem pivoteamento e como consequencia pode ser muito instável em determinados casos, como por exemplo em matrizes com elementos diagonais muito pequenos.

No entanto, da condição de Galerkin (3.38) e do fato de v_{m+1} ser ortogonal a V_m resulta que o vetor resíduo r_m está na mesma direção de v_{m+1} e, como consequência os vetores resíduos r_i's são ortogonais. Além disso, a relação $P_m = V_m U_m^{-1}$ implica que os vetores auxiliares p_i formam um conjunto A-ortogonal, isto é, $(Ap_i, p_j) = 0$ se $i \neq j$. De fato, pois com

$$V_m^T A U_m = T_m = L_m U_m, \text{ temos}$$

$$\begin{aligned} P_m^T A P_m &= U_m^{-T} V_m^T A V_m U_m^{-1} \\ &= U_m^{-T} T_m U_m^{-1} \\ &= U_m^{-T} L_m, \end{aligned}$$

onde $U_m^{-T} = (U_m^T)^{-1}$. Como $U_m^{-T} L_m$ é triangular inferior e também simétrica, uma vez que é igual a $P_m^T A P_m$, vem que $P_m^T A P_m$ é diagonal.

Portanto, podemos abandonar o processo de eliminação Gaussiana e encontrar os coeficientes da formulação progressiva impondo as duas seguintes condições:

i. os vetores resíduos são ortogonais e podem substituir os v_m's;

ii. os vetores auxiliares p_i's são A-ortogonais (ou conjugados).

Como consequência a formulação progressiva acima pode ser reformulada para produzir o algoritmo dos Gradientes Conjugados que a seguir deduziremos.

Nessa nova formulação, o vetor x_{j+1} pode ser expressado como

$$x_{j+1} = x_j + \alpha_j p_j \tag{3.54}$$

então, para r_{j+1} temos

$$\begin{aligned} r_{j+1} = b - Ax_{j+1} &= b - Ax_j - \alpha_j Ap_j \\ r_{j+1} &= r_j - \alpha_j Ap_j. \end{aligned} \tag{3.55}$$

Impondo a condição (i) obtemos

$$(r_{j+1}, r_j) = 0 \Rightarrow (r_j - \alpha_j Ap_j, r_j) = 0$$

que resulta

$$\alpha_j = \frac{(r_j, r_j)}{(Ap_j, r_j)}. \tag{3.56}$$

Por outro lado, de (3.49) temos

$$p_{j+1} = r_{j+1} + \beta_j p_j \tag{3.57}$$

que implica

$$\begin{aligned} (Ap_j, r_j) &= (Ap_j, p_j - \beta_{j-1} p_{j-1}) \\ &= (Ap_j, p_j) - \beta_{j-1}(Ap_j, p_{j-1}) \\ &= (Ap_j, p_j), \end{aligned} \tag{3.58}$$

onde foi usada a A-ortogonalidade dos vetores p_j,s. Então (3.56) torna-se

$$\alpha_j = \frac{(r_j, r_j)}{(Ap_j, p_j)}. \tag{3.59}$$

Além disso, usando a relação (3.57) e impondo novamente a condição de A-ortogonalidade dos p_j's (condição ii)

$$\begin{aligned} 0 = (Ap_j, p_{j+1}) &= (Ap_j, r_{j+1} + \beta_j p_j) \\ &= (Ap_j, r_{j+1}) + \beta_j (Ap_j, p_j) \end{aligned}$$

obtemos

$$\beta_j = -\frac{(Ap_j, r_{j+1})}{(Ap_j, p_j)}. \tag{3.60}$$

No entanto, temos ainda de (3.55)

$$Ap_j = -\frac{1}{\alpha_j}(r_{j+1} - r_j), \tag{3.61}$$

que substituindo no numerador da equação (3.60) fornece uma nova expressão para β_j

$$\begin{aligned} \beta_j &= \frac{1}{\alpha_j} \times \frac{((r_{j+1} - r_j), r_{j+1})}{(Ap_j, p_j)} \\ &= \frac{(Ap_j, p_j)}{(r_j, r_j)} \times \frac{(r_{j+1}, r_{j+1})}{(Ap_j, p_j)} \\ &= \frac{(r_{j+1}, r_{j+1})}{(r_j, r_j)}, \end{aligned} \tag{3.62}$$

onde foi usado o fato de que r_{j+1} é ortogonal a r_j.

Essas últimas relações podem ser usadas para formar o algoritmo dos Gradientes Conjugados.

ALGORITMO 3.8: O Método dos Gradientes Conjugados

1. Calcule $r_0 = b - Ax_0$, e faça $p_0 = r_0$
2. Para $j = 0, 1, \ldots,$ até convergência faça:
3. $\quad \alpha_j = (r_j, r_j)/(Ap_j, p_j)$
4. $\quad x_{j+1} = x_j + \alpha_j p_j$
5. $\quad r_{j+1} = r_j - \alpha_j Ap_j$
6. $\quad \beta_j = (r_{j+1}, r_{j+1})/(r_j, r_j)$
7. $\quad p_{j+1} = r_{j+1} + \beta_j p_j$
8. Fim.

Uma outra apresentação do método dos Gradientes conjugados, através de uma visão bem geométrica e intuitiva, é apresentada em [28].

Capítulo 4

O Método BiCG e os métodos tipo-produto

A terceira e última classe dos métodos no subespaço de Krylov tratados nesse trabalho é baseada na condição de Petrov-Galerkin. Para esses métodos o subespaço das condições $\mathcal{L}_m$ é igual ao subespaço de Krylov m-dimensional gerado pela matriz A^T, isto é, $\mathcal{L}_m = \mathcal{K}_m(A^T, v)$. Dentro dessa classe de métodos, o BiCG é visto como um método precursor e a partir dele várias formulações são desenvolvidas visando eliminar a necessidade, presente no BiCG, de trabalhar com a matriz A^T. Como veremos, estas formulações são baseadas em relações de recorrência que envolvem produtos polinomiais, daí serem chamados de métodos tipo-produto.

4.1 O procedimento de bi-ortogonalização de Lanczos

Para podermos dar início à dedução dos métodos dessa classe, precisamos, como antes, construir bases vetoriais para os subespaços de Krylov envolvidos nessa formulação. Nesse sentido, a presença da matriz A^T cria uma condição adequada para a utilização do procedimento de bi-ortogonalização de Lanczos.

Esse procedimento de bi-ortogonalização, proposto por Lanczos em 1952 [26], consiste em construir duas seqüências de vetores bi-ortogonais, ou seja, $v_i,\ i = 1, \ldots, m,$ e $w_j,\ j = 1, \ldots, m,$ tais que $(v_i, w_j) = \delta_{ij},\ i, j = 1, \ldots, m,$ onde δ_{ij} é a função delta de Kronecker. O algoritmo que realiza esse procedimento de bi-ortogonalização de Lanczos é apresentado a seguir.

ALGORITMO 4.1: Bi-ortoganalização de Lanczos.

1. Escolha dois vetores v_1 e w_1 tais que $(v_1, w_1) = 1$
2. Atribua $\beta_1 = \delta_1 \equiv 0$ e $v_0 = w_0 = 0$
3. Para $j = 1, 2, \ldots, m$ faça:
4. $\alpha_j = (Av_j, w_j)$
5. $\hat{v}_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}$
6. $\hat{w}_{j+1} = A^T w_j - \alpha_j w_j - \delta_j w_{j-1}$
7. $\delta_{j+1} = |(\hat{v}_{j+1}, \hat{w}_{j+1})|^{1/2}$. Se $\delta_{j+1} = 0$ pare
8. $\beta_{j+1} = (\hat{v}_{j+1}, \hat{w}_{j+1})/\delta_{j+1}$
9. $w_{j+1} = \hat{w}_{j+1}/\beta_{j+1}$
10. $v_{j+1} = \hat{v}_{j+1}/\delta_{j+1}$
11. fim

Neste algoritmo existem muitas possibilidades de escolha para os escalares δ_{j+1} e β_{j+1} nas linhas 7 e 8. Estes dois parâmetros são obtidos através dos dois vetores v_{j+1} e w_{j+1} e podem ser selecionados impondo que

$$(v_{j+1}, w_{j+1}) = 1. \tag{4.1}$$

Note porém, que em função das relações apresentadas nas linhas 9 e 10 do algoritmo 4.1, para satisfazer (4.1) basta escolher $\beta_{j+1},\ \delta_{j+1}$ de modo que

$$\delta_{j+1}\beta_{j+1} = (\hat{v}_{j+1}, \hat{w}_{j+1}). \tag{4.2}$$

Além disso, se $\beta_{j+1}, \delta_{j+1}$ forem escolhidos como nas linhas 7 e 8 do algoritmo 4.1 então os δ_j's serão positivos e $\beta_j = \pm\delta_j$.

Antes de deduzir métodos iterativos baseados no procedimento de bi-ortogonalização descrito, é conveniente discutir alguns resultados práticos do algoritmo 4.1. Nesse sentido a proposição apresentada a seguir, cuja prova pode ser encontrada em [23], estabelece importantes resultados.

Proposição 4.1.1. *Se o algoritmo 4.1 não é interrompido antes do m-ésimo passo, então os vetores* $v_i, i = 1, \ldots, m$, *e* $w_j, j = 1, \ldots, m$, *formam um sistema bi-ortogonal, isto é,*

$$(v_i, w_j) = \delta_{i,j}, \quad 1 \leq i, j \leq m.$$

Além disso, $\{v_i\}_{i=1,2,\ldots,m}$ *e uma base de* $\mathcal{K}_m(A, v_1)$ *e* $\{w_i\}_{i=1,2,\ldots,m}$ *e uma base de* $\mathcal{K}_m(A^T, w_1)$ *e as seguintes relações são válidas,*

$$AV_m = V_m T_m + \delta_{m+1} v_{m+1} e_m^T, \tag{4.3}$$

$$A^T W_m = W_m T_m^T + \beta_{m+1} w_{m+1} e_m^T, \tag{4.4}$$

$$W_m^T A V_m = T_m, \tag{4.5}$$

onde T_m *denota a matriz tridiagonal*

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \beta_3 & & \\ & \cdot & \cdot & \cdot & \\ & & \delta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \delta_m & \alpha_m \end{pmatrix}, \tag{4.6}$$

e os escalares $\delta_{j+1}, \beta_{j+1}$ *é qualquer par que satisfaça a relação (4.2).*

Em primeiro lugar é importante destacar que o método produz uma matriz tridiagonal T_m que representa a projeção da matriz A sobre o espaço gerado pelos vetores v_j e ortogonalmente ao espaço gerado pelos vetores w_j. De forma semelhante, T_m^T representa a projeção de A^T sobre o espaço gerado pelos w_j e ortogonalmente ao espaço gerado pelos v_j.

Além disso, o algoritmo 4.1 é baseado em uma dupla relação de recorrência de dois termos, o que representa uma vantagem do ponto de vista computacional com relação ao método de Arnoldi, uma vez que uma pequena quantidade

de vetores precisa ser armazenada. Por outro lado, a construção de duas seqüências de vetores, $\{v_j\}_{j=1,\ldots,m}$ e $\{w_j\}_{j=1,\ldots,m}$, representa uma clara desvantagem desse procedimento, que é agravada pelo fato de envolver operações com a transposta da matriz original.

Existem ainda outros aspectos importantes: o algoritmo 4.1 é adequado para o caso onde dois sistemas precisam ser resolvidos, um com a matriz A e outro com A^T, uma vez que isto já é feito implicitamente pelo método; as duas seqüências de vetores $\{v_j\}_{j=1,\ldots,m}$ e $\{w_j\}_{j=1,\ldots,m}$ geradas pelo algoritmo 4.1 formam respectivamente, bases para os subespaços de Krylov $\mathcal{K}_m(A, v_1)$ e $\mathcal{K}_m(A^T, w_1)$, assim como estabelece a proposição anterior.

4.2 O método BiCG

O método dos Gradientes Bi-conjugados (BiCG) é um processo de projeção sobre o subespaço de Krylov

$$\mathcal{K}_m = Span\{v_1, Av_1, \ldots, A^{m-1}v_1\}$$

ortogonalmente a

$$\mathcal{L}_m = Span\{w_1, A^T w_1, \ldots, (A^T)^{m-1}w_1\}$$

tomando, como usual, $v_1 = r_0/\|r_0\|_2$. O vetor w_1 é arbitrário desde que $(v_1, w_1) \neq 0$. Porém na prática, se nenhum sistema com a matriz A^T precisa ser resolvido, usa-se freqüentemente $w_1 = v_1$. Caso contrário, o vetor w_1 deve ser obtido como $w_1 = r_0^*/\|r_0^*\|_2$, com $r_0^* = b^* - A^T x_0^*$ onde b^* e x_0^* são, respectivamente,o vetor do lado direito e a estimativa inicial para o sistema com A^T.

Temos neste caso, uma situação muito semelhante àquela obtida com a aplicação do algoritmo de Lanczos simétrico, que deu origem ao método dos Gradientes Conjugados (GC). O algoritmo 4.1 produz uma matriz tridiagonal e armazena apenas uma parte dos vetores que são gerados. Devemos então,

como antes, buscar uma formulação progressiva através da fatorização LU da matriz tridiagonal T_m. Para isso, escrevemos a decomposiçao LU da matriz T_m como

$$T_m = L_m U_m \tag{4.7}$$

e definimos

$$P_m = V_m U_m^{-1}. \tag{4.8}$$

De forma semelhante definimos a matriz

$$P_m^* = W_m L_m^{-T}, \tag{4.9}$$

que é obtida aplicando as relações (3.39) a (3.45) no sistema implícito com A^T. Assim, como resultado as relações (4.8) e (4.9), temos que os vetores coluna p_i^* da matriz P_m^* e os vetores coluna p_i da matriz P_m são A-ortogonais (conjugados), uma vez que

$$(P_m^*)^T A P_m = L_m^{-1} W_m^T A V_m U_m^{-1} = L_m^{-1} T_m U_m^{-1} = I, \tag{4.10}$$

onde foi usada a relação (4.5). Além disso, como no algoritmo dos Gradientes Conjugados (GC), os vetores r_j e r_j^* estão na mesma direção de v_{j+1} e w_{j+1} respectivamente, formando então uma seqüência bi-ortogonal.

Podemos portanto utilizar essas informações e obter facilmente um algoritmo tipo gradiente conjugado conhecido como Gradiente Bi-conjugado.

ALGORITMO 4.2: Método BiCG.

1. Calcule $r_0 = b - Ax_0$ para alguma estimativa inicial x_0. Escolha r_0^* tal que $(r_0, r_0^*) \neq 0$
2. Atribua, $p_0 = r_0,\ p_0^* = r_0^*$
3. Para $j = 0, 1, \ldots,$ até convergência, faça:
4. $\alpha_j = (r_j, r_j^*)/(Ap_j, p_j^*)$
5. $x_{j+1} = x_j + \alpha_j p_j$
6. $r_{j+1} = r_j - \alpha_j A p_j$

7. $r^*_{j+1} = r^*_j - \alpha_j A^T p^*_j$

8. $\beta_j = (r_{j+1}, r^*_{j+1})/(r_j, r^*_j)$

9. $p_{j+1} = r_{j+1} + \beta_j p_j$

10. $p^*_{j+1} = r^*_{j+1} + \beta_j p^*_j$

11. Fim.

Se um sistema envolvendo a matriz A^T e um vetor do lado direito b^* também precisar ser resolvido, então o resíduo r^*_0 deve ser definido como $r^*_0 = b^* - A^T x^*_0$ e a solução aproximada para esse sistema atualizada após a linha 5 do algoritmo por $x^*_{j+1} = x^*_j + \alpha_j p^*_j$.

4.2.1 Interrupção anormal do processo no BiCG *("Breakdowns")*

Além das dificuldades comentadas na seção anterior, como a necessidade de construir bases para dois subespaços de Krylov e o envolvimento de operações com A^T, o método BiCG apresenta ainda outros problemas que dificultaram sua aceitação entre os analistas numéricos [26]. Estes problemas estão concentrados principalmente nas várias possibilidades de interrupção *("breakdowns")* que o algoritmo 4.1 apresenta. Note que o algoritmo para na linha 7 sempre que,

$$(\hat{v}_{j+1}, \hat{w}_{j+1}) = 0. \tag{4.11}$$

Isto pode ocorrer de duas formas diferentes. Se um dos vetores $\hat{v}_{j+1}$ ou $\hat{w}_{j+1}$ é zero, ou se ambos são não nulos, mas o produto interno entre eles é zero. No primeiro caso não há problemas pois se $\hat{v}_{j+1} = 0$ então o espaço gerado por V_j é invariante e, conseqüentemente, a solução aproximada é exata [23]. O mesmo ocorre para $\hat{w}_{j+1}$ com relação ao sistema dual com A^T. Nesta situação a interrupção é chamada de "interrupção feliz"*("lucky breakdown")*.

A outra interrupção é mais grave e ocorre quando o processo de bi-ortogonalização produz um produto interno nulo entre dois novos vetores nos subespaços de Krylov gerados com A e A^T. Neste caso a interrupção é chamada "interrup-

ção séria" *("serious breakdown")*, e pode ser evitada utilizando uma estratégia descrita em [15] chamada *"look-ahead"*.

4.2.2 A convergência do método BiCG

Existem poucos resultados teóricos a respeito da convergência do BiCG [1]. Para sistemas simétricos e definidos positivos o método fornece os mesmos resultados do gradiente conjugado, porém, a um custo duas vezes maior por iteração. Para os casos não simétricos a redução da norma do vetor resíduo é comparável, em muitas situações, àquela obtida através de outros métodos [14] o que é freqüentemente confirmado na prática. Porém, o comportamento da convergência do BiCG é muito irregular, apresentando uma grande variação na norma dos vetores resíduos o que pode comprometer a acurácia da aproximação em precisão finita. Além disso, como comentamos na seção anterior, o método apresenta diversas situações de *"breakdown"*.

4.3 Métodos relacionados ao BiCG

Sem dúvida uma das maiores dificuldades dos métodos baseados no processo de bi-ortogonalização de Lanczos, em especial o BiCG, são as operações envolvendo a matriz A^T.

Além disso, das duas bases vetoriais geradas apenas uma é explorada para a solução, a outra é usada apenas para calcular os produtos internos necessários para gerar bases bi-ortogonais. No caso específico do BiCG, os vetores p_i^* gerados com A^T são usados apenas para obter os escalares α_j e β_j.

Esta situação provocou o surgimento de métodos relacionados ao BiCG mas que evitam o uso da matriz A^T em suas formulações. A seguir apresentaremos alguns desses métodos que são conhecidos como *"transpose-free"*.

4.3.1 O método CGS

O método Gradiente Conjugado Quadrado CGS *(Conjugate Gradient Squared)* foi desenvolvido por Sonneveld em 1989 [29] a partir da observação de que as operações com a matriz A^T presentes no método BiCG poderiam ser reformuladas para envolverem apenas a matriz A, e que essas novas operações poderiam ser usadas para conseguir uma redução adicional da norma do vetor resíduo.

Note que no BiCG os vetores podem ser expressos por meio de polinômios. Assim, o vetor resíduo no passo j pode ser escrito como:

$$r_j = \Phi_j(A)r_0, \tag{4.12}$$

onde Φ_j é um polinômio de grau j satisfazendo $\Phi_j(0) = 1$. De forma semelhante temos,

$$p_j = \Pi_j(A)r_0 \tag{4.13}$$

$$r_j^* = \Phi_j(A^T)r_0^*, \tag{4.14}$$

$$p_j^* = \Pi_j(A^T)r_0^*, \tag{4.15}$$

onde Π_j é um polinômio de grau j. Além disso, os coeficientes α_j e β_j no BiCG são calculados através de produtos internos envolvendo os vetores r_j, r_j^*, p_j, p_j^*, e podem ser reescritos formalmente utilizando a forma polinominal. Assim temos que

$$\alpha_j = \frac{(\Phi_j(A)r_0, \Phi_j(A^T)r_0^*)}{(A\Pi_j(A)r_0, \Pi_j(A^T)r_0^*)} = \frac{(\Phi_j^2(A)r_0, r_0^*)}{(A\Pi_j^2(A)r_0, r_0^*)}, \tag{4.16}$$

o que indica que se for possível encontrar fórmulas de recorrência para $\Phi_j^2(A)r_0$ e $\Pi_j^2(A)r_0$ então, o cálculo de α_j e, de forma semelhante, o de β_j, poderão ser feitos sem maiores problemas.

O ponto de partida para a dedução do método CGS são as relações de recorrência usadas para calcular r_{j+1} e p_{j+1} nas linhas 6 e 9 do algoritmo

BiCG. Assim,

$$r_{j+1} = r_j - \alpha_j A p_j$$

$$p_{j+1} = r_{j+1} + \beta_j p_j.$$

Agora, substituindo nas duas relações anteriores as expressões polinominais definidas em (4.12) e (4.13) obtemos as seguintes relações de recorrência para Φ_{j+1} e Π_{j+1}

$$\Phi_{j+1}(t) = \Phi_j(t) - \alpha_j t \Pi_j(t) \tag{4.17}$$

$$\Pi_{j+1}(t) = \Phi_{j+1}(t) + \beta_j \Pi_j(t). \tag{4.18}$$

Como estamos interessados em recorrências para os polinômios Φ_j^2 e Π_j^2 elevamos a expressões (4.17) e (4.18) ao quadrado para obtermos

$$\Phi_{j+1}^2 = \Phi_j^2 - 2\alpha_j t \Phi_j \Pi_j + \alpha_j^2 t^2 \Pi_j^2, \tag{4.19}$$

$$\Pi_{j+1}^2 = \Phi_{j+1}^2 + 2\beta_j \Phi_{j+1} \Pi_j + \beta_j^2 \Pi_j^2, \tag{4.20}$$

onde por simplicidade foi omitida a variável (t), como será feito doravante quando não houver ambigüidades.

Note porém, que para poder utilizar as relações para Φ_{j+1}^2 e Π_{j+1}^2 definidas acima, é preciso de uma expressão para $\Phi_j \Pi_j$ e outra para $\Phi_{j+1}\Pi_j$. Estas expressões podem ser facilmente obtidas da seguinte forma: para a primeira tome a relação (4.18) no passo j, e multiplique por Φ_j para obter

$$\Phi_j \Pi_j = \Phi_j^2 + \beta_{j-1} \Phi_j \Pi_{j-1}. \tag{4.21}$$

Para a segunda, multiplique (4.17) por Π_j. Como resultado,

$$\Phi_{j+1}\Pi_j = \Phi_j \Pi_j - \alpha_j t \Pi_j^2. \tag{4.22}$$

Podemos agora definir os vetores

$$r_j \equiv \Phi_j^2(A) r_0 \tag{4.23}$$

$$p_j \equiv \Pi_j^2(A) r_0 \tag{4.24}$$

$$u_j \equiv \Phi_j(A)\Pi_j(A) \tag{4.25}$$

$$q_j \equiv \Phi_{j+1}(A)\Pi_j(A) \tag{4.26}$$

que nos permitem reescrever as recorrências polinomiais (4.19 - 4.22) na forma vetorial. Fazendo isso,

$$\begin{aligned} r_{j+1} &= r_j - 2\alpha_j A u_j + \alpha_j^2 A^2 p_j \\ &= r_j - \alpha_j A(2u_j - \alpha_j A p_j), \qquad (4.27) \\ p_{j+1} &= r_{j+1} + 2\beta_j q_j + \beta_j^2 p_j \\ &= r_{j+1} + \beta_j q_j + (\beta_j q_j + \beta_j^2 p_j), \qquad (4.28) \\ u_j &= r_j + \beta_{j-1} q_{j-1}, \qquad (4.29) \\ q_j &= u_j - \alpha_j A p_j, \qquad (4.30) \end{aligned}$$

obtemos as recorrências que formam a base do método CGS.

Note que é possível fazer ainda uma pequena simplificação substituindo (4.30) na expressão entre parênteses em (4.27), e substituindo (4.29) em (4.28). Dessa forma obtemos duas novas expressões para os vetores r_{j+1} e p_{j+1},

$$\begin{aligned} r_{j+1} &= r_j - \alpha_j A(u_j + q_j), \qquad (4.31) \\ p_{j+1} &= u_{j+1} + \beta_j(q_j + \beta_j p_j), \qquad (4.32) \end{aligned}$$

que juntamente com as relações (4.29) e (4.30) dão origem à seguinte formulação do algoritmo CGS.

ALGORITMO 4.3: Método CGS.

1. Calcule $r_0 = b - Ax_0; r_0^*$ arbitrário
2. Atribua $p_0 = u_0 = r_0$.
3. Para $j = 0, 1, 2, \ldots,$ até convergência faça:
4. $\alpha_j = (r_j, r_j^*)/(Ap_j, r_0^*)$
5. $q_j = u_j - \alpha_j A p_j$
6. $x_{j+1} = x_j + \alpha_j(u_j + q_j)$
7. $r_{j+1} = r_j - \alpha_j A(u_j + q_j)$
8. $\beta_j = (r_{j+1}, r_0^*)/(r_j, r_0^*)$
9. $u_{j+1} = r_{j+1} + \beta_j q_j$

10. $p_{j+1} = u_{j+1} + \beta_j(q_j + \beta_j p_j)$

11. fim

A fórmula de atualização do vetor x_{j+1} na linha 6 pode ser obtida facilmente a partir da recorrência para o vetor resíduo r_{j+1}. Observe que

$$r_{j+1} = b - Ax_{j+1}.$$

Assim,

$$\begin{aligned} Ax_{j+1} &= b - r_{j+1} \\ &= b - r_j + \alpha_j A(u_j + q_j) \\ &= b - b + Ax_j + \alpha_j A(u_j + q_j) \\ &= A(x_j + \alpha_j(u_j + q_j)). \end{aligned} \tag{4.33}$$

Note que as operações com a matriz A^T foram eliminadas e que agora dois produtos com a matriz A são realizados em cada passo.

No que diz respeito à convergência do método CGS, alguns comentários podem ser feitos. De acordo com o que foi apresentado por Sonneveld em [29] o método CGS pode apresentar uma convergência aproximadamente duas vezes mais rápida que o BiCG, o que é freqüentemente comprovado na prática.

Esta situação pode ser teoricamente observada na comparação das expressões de resíduo em cada um dos métodos. No CGS os vetores residuais podem ser expressos como $\tilde{r}_i = \Phi_i^2(A)r_0$. Assim, se o método BiCG produz um resíduo $r_i(= \Phi_i(A)r_0)$, que é pequeno se comparado com r_0, ou em outras palavras, se o operador $\Phi_i(A)$ transformou r_0 em um vetor cuja norma é menor, então aplicando o operador Φ_i^2, como no CGS, a redução será duas vezes maior.

Por outro lado, quando o operador Φ_i^2 é aplicado os erros de arredondamento tendem a ser mais prejudiciais à precisão da solução, em função das enormes oscilações na norma dos vetores resíduos gerados pelo método.

Mais detalhes sobre o comportamento da convergência do método CGS podem ser encontrados em [35].

4.3.2 O método BiCGSTAB

O método Gradiente Bi-conjugado Estabilizado (BiCGSTAB) [33] foi proposto por Van der Vorst como uma variante do BiCG que fornece uma convergência mais rápida e suave que o CGS [34].

Este método explora a idéia de eliminar as operações com A^T não com o polinômio $\Phi_j(A)$ que define o vetor resíduo no BiCG, como é feito pelo CGS, mas com outros polinômios em A.

Isto é feito porque a relação satisfeita pelo resíduo no método CGS, apesar de promover uma convergência cerca de duas vezes mais rápida que o método BiCG, frequentemente produz grandes oscilações na norma do vetor resíduo o que pode comprometer seriamente a solução quando se trabalha com aritmética de precisão finita. Dessa forma, a relação satisfeita pelo resíduo no CGS

$$r_j = \Phi_j^2(A) r_0 \tag{4.34}$$

é transformada, no BiCGSTAB, em

$$r_i = \Psi_i(A)\Phi_i(A) r_0, \tag{4.35}$$

onde Ψ_i é outro conjunto adequado de polinômios de grau i.
Na prática, o processo é semelhante àquele usado na obtenção do CGS. Adicionalmente, exploramos a relação de bi-ortogonalidade dos vetores resíduo no BiCG.

$$(\Phi_i(A) r_0, \Phi_j(A^T) r_0^*) = 0, j < i \tag{4.36}$$

que nos diz que o vetor $\Phi_i(A) r_0$ é ortogonal ao subespaço de Krylov $i-$dimensional $\mathcal{K}_i(A^T, r_0^*)$, gerado pelos vetores $\{r_0^*, A^T r_0^*, \ldots, (A^T)^{(i-1)} r_0^*\}$.

Podemos então adotar um procedimento análogo ao utilizado no CGS e impor que o vetor r_i seja ortogonal ao vetor $\Psi_j(A^T) r_0^*$, ou de forma equivalente

$$(\Psi_j(A)\Phi_i(A) r_0, r_0^*) = 0, \tag{4.37}$$

O método então é construído explorando a relação (4.37) e impondo que a solução aproximada x_j, em cada passo do processo, produza um resíduo da forma

$$r_j = \Psi_j(A)\Phi_j(A)r_0, \tag{4.38}$$

com

$$\Psi_j(A) = (I - \omega_0 A)(I - \omega_1 A)\ldots(I - \omega_{j-1}A), \tag{4.39}$$

onde a constante ω_j, no $j-$ésimo passo, é escolhida para minimizar o resíduo $r_j = \Psi_j(A)\Phi_j(A)r_0$.

A dedução do método BiCGSTAB parte novamente das duas importantes relações de recorrência para os vetores r_j e p_j no método BiCG,

$$r_j = r_{j-1} - \alpha_{j-1}Ap_{j-1} \tag{4.40}$$

$$p_j = r_j + \beta_{j-1}p_{j-1}, \tag{4.41}$$

e mais uma vez usando as formas polinomiais

$$r_j = \Phi_j(A)r_0$$

$$p_j = \Pi_j(A)r_0,$$

onde $\Phi_j(A)$ e $\Pi_j(A)$ são polinômios em A de grau j.
Como resultado obtém-se as relações de recorrência para essas formas polinomiais,

$$\Phi_j(A)r_0 = (\Phi_{j-1}(A) - \alpha_{j-1}A\Pi_{j-1}(A))r_0, \tag{4.42}$$

e

$$\Pi_j(A)r_0 = (\Phi_j(A) + \beta_{j-1}\Pi_{j-1}(A))r_0, \tag{4.43}$$

que serão usadas para buscar as desejadas relações de recorrência do método BiCGSTAB.

Uma vez que é preciso encontrar relações de recorrência para o vetor resíduo definido em (4.38), pode-se multiplicar por Ψ_j a relação (4.42). Fazendo isso obtém-se

$$\Psi_j(A)\Phi_j(A)r_0 = (I - \omega_j A)\Psi_{j-1}(A)(\Phi_{j-1}(A) - \alpha_{j-1}A\Pi_{j-1}(A))r_0. \tag{4.44}$$

A relação para o produto $\Psi_j(A)\Pi_j(A)r_0$ também é obtida das relações do BiCG, através da multiplicação da expressão (4.43) por $\Psi_j(A)$. Assim,

$$\begin{aligned}\Psi_j(A)\Pi_j(A)r_0 &= \Psi_j(A)\Phi_j(A)r_0 + \beta_{j-1}\Psi_j(A)\Pi_{j-1}(A)r_0 \\ &= \Psi_j(A)\Phi_j(A)r_0 + \beta_{j-1}(I-\omega_j A)\Psi_{j-1}(A)\Pi_{j-1}(A)r_0.\end{aligned}$$

O próximo passo então é buscar fórmulas para as constantes utilizadas pelo método. Para o cálculo de β_j, por exemplo, utiliza-se produtos internos envolvendo os novos vetores criados. Observe que no método BiCG o escalar β_j é calculado como $\beta_j = \rho_{j+1}/\rho_j$, onde

$$\begin{aligned}\rho_j &= (\Phi_j(A)r_0, \Phi_j(A^T)r_0^*) \\ &= (\Phi_j(A)^2 r_0, r_0^*). \qquad (4.45)\end{aligned}$$

Note, porém, não é possível calcular ρ_j pela fórmula (4.45) pois os vetores necessários não estão disponíveis. Uma saída é relacionar ρ_j com o escalar $\tilde{\rho}_j$ que é obtido como

$$\tilde{\rho}_j = (\Phi_j(A)r_0, \Psi_j(A^T)r_0^*). \qquad (4.46)$$

Para isso, expande-se $\Psi_j(A^T)r_0^*$ explicitamente a partir de (4.39), substituindo A por A^T,

$$\Psi_j(A^T)r_0^* = (I-\omega_0 A^T)(I-\omega_1 A^T)\ldots(I-\omega_{j-1}A^T)r_0^*. \qquad (4.47)$$

Esta expansão é necessária para poder encontrar o coeficiente de maior potência em $\Psi_j(A^T)r_0^*$. Na expansão, este é o único termo relevante uma vez que $\Phi_j(A)r_0$ é ortogonal a todos os vetores $(A^T)^k r_0^*$, com $k < j$.

Para $\Psi_j(A^T)r_0^*$ esse coeficiente de maior potência, que será denotado por $\zeta^{(j)}$, pode ser encontrado por indução. Observe que para $j = 1$ tem-se que

$$\Psi_1(A^T)r_0^* = (I-\omega_0 A^T)r_0^*,$$

cujo coeficiente do termo de maior potência é

$$-\omega_0 = (-1)^1\omega_0.$$

Suponha que para $j = k$ o termo de maior grau em $\Psi_j(A^T)r_0^*$ seja dado por

$$(-1)^k \omega_0 \omega_1 \ldots \omega_{k-1},$$

então, para $j = k+1$ tem-se,

$$\begin{aligned} \Psi_{k+1}(A^T)r_0^* &= \Psi_k(A^T)r_0^* \times (I - \omega_k A^T) \\ &= (I + \ldots (-1)^k \omega_0 \omega_1 \ldots \omega_{k-1}(A^T)^k) r_0^* \times (I - \omega_k A^T) \\ &= (I + \ldots (-1)^{k+1} \omega_0 \omega_1 \ldots \omega_k r_0^*. \end{aligned} \tag{4.48}$$

Assim, para qualquer j o coeficiente do termo de maior potência é dado por

$$\zeta^{(j)} = (-1)^j \omega_0 \omega_1 \ldots \omega_{j-1}. \tag{4.49}$$

Usando então a expressão (4.47) na relação (4.46) para $\tilde{\rho}_j$, e valorizando apenas o coeficiente de maior potência em $\Psi_j(A^T)$ obtém-se

$$\tilde{\rho}_j = (\Phi_j(A)r_0, \zeta^{(j)}(A^T)^j r_0^*). \tag{4.50}$$

Em particular, se $\eta^{(j)}$ representa o coeficiente de maior potência no polinômio Φ_j então é possível reescrever $\tilde{\rho}_j$ em função de ρ_j, usando (4.50) da seguinte forma

$$\begin{aligned} \tilde{\rho}_j &= (\Phi_j(A)r_0, \frac{\zeta^{(j)}}{\eta^{(j)}} \Phi_j(A^T)r_0^*) \\ &= \frac{\zeta^{(j)}}{\eta^{(j)}} \rho_j, \end{aligned} \tag{4.51}$$

onde foi usada a relação (4.45). Além disso, olhando para $\zeta^{(j)}$definido em (4.49) nota-se a seguinte relação

$$\zeta^{(j+1)} = -\omega_j \zeta^{(j)}, \tag{4.52}$$

da mesma forma para $\eta^{(j+1)}$

$$\eta^{(j+1)} = -\alpha_j \eta^{(j)}. \tag{4.53}$$

Como resultado,

$$\frac{\tilde{\rho}_{j+1}}{\tilde{\rho}_j} = \frac{\frac{\zeta^{(j+1)}}{\eta^{(j+1)}}\rho_{j+1}}{\frac{\zeta^{(j)}}{\eta^{(j)}}\rho_j} \tag{4.54}$$

$$= \frac{\frac{-\omega_j\zeta^{(j)}}{-\alpha_j\eta^{(j)}}\rho_{j+1}}{\frac{\zeta^{(j)}}{\eta^{(j)}}\rho_j} \tag{4.55}$$

$$= \frac{\omega_{(j)}}{\alpha_{(j)}} \times \frac{\rho_{j+1}}{\rho_j}, \tag{4.56}$$

que fornece

$$\beta_j = \frac{\rho_{j+1}}{\rho_j} = \frac{\tilde{\rho}_{j+1}}{\tilde{\rho}_j} \times \frac{\alpha_j}{\omega_j}. \tag{4.57}$$

Um processo análogo fornece o escalar α_j como

$$\alpha_j = \frac{\tilde{\rho}_j}{(Ap_j, r_0^*)}. \tag{4.58}$$

Já o parâmetro ω_j é escolhido de modo a minimizar a norma 2 do vetor resíduo

$$r_{j+1} = \Psi_{j+1}(A)\Phi_{j+1}(A)r_0 = (I - \omega_j A)\Psi_j(A)\Phi_{j+1}(A)r_0. \tag{4.59}$$

Então, substituindo a expressão para $\Phi_{j+1}(A)r_0$, definida em (4.42), no segundo membro da igualdade acima, e usando as relações

$$r_j = \Phi_j(A)\Psi_j(A)r_0,$$

e

$$p_j = \Psi_j(A)\Pi_j(A)r_0,$$

obtém-se

$$r_{j+1} = (I - \omega_j A)(r_j - \alpha_j A p_j), \tag{4.60}$$

que fornece o valor ótimo para ω_j

$$\omega_j = \frac{(As_j, s_j)}{(As_j, As_j)}, \tag{4.61}$$

onde

$$s_j \equiv r_j - \alpha_j A p_j. \tag{4.62}$$

Por fim, é preciso uma fórmula para atualizar a solução aproximada x_{j+1}. Das relações (4.60) e (4.62) tem-se

$$r_{j+1} = r_j - \alpha_j A p_j - \omega_j A s_j,$$

que fornece,

$$x_{j+1} = x_j + \alpha_j p_j + \omega_j s_j.$$

Essas relações formam o método BiCGSTAB.

ALGORITMO 4.4:**Método BiCGSTAB**.

1. Calcule $r_0 = b - Ax_0$; r_0^* arbitrário;
2. $p_0 = r_0$
3. Para $j = 0, 1, \ldots,$ até convergência faça:
4. $\quad \alpha_j = (r_j, r_0^*)/(Ap_j, r_0^*)$
5. $\quad s_j = r_j - \alpha_j A p_j$
6. $\quad \omega_j = (As_j, s_j)/(As_j, As_j)$
7. $\quad x_{j+1} = x_j + \alpha_j p_j + \omega_j s_j$
8. $\quad r_{j+1} = s_j - \omega_j A s_j$
9. $\quad \beta_j = \frac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)} \times \frac{\alpha_j}{\omega_j}$
10. $\quad p_{j+1} = r_{j+1} + \beta_j (p_j - \omega_j A p_j)$
11. fim.

4.3.3 O método GPBiCG

O procedimento desenvolvido por este método é uma generalização do que foi feito nos métodos CGS e BiCGSTAB, daí o nome método tipo-Produto Generalizado baseado no BiCG (GPBICG)[37]. Neste método, o polinômio residual é definido como sendo o produto de dois polinômios de grau j

$$\Theta_j(t)\Phi_j(t), \tag{4.63}$$

onde Φ_j é o polinômio que define o vetor resíduo no método BiCG e Θ_j é um polinômio indeterminado.

O polinômio Θ_j é construído eliminando Π_j das relações (4.17) e (4.18) e recuperando a relação de recorrência de três termos para Φ_j

$$\Phi_0(t) = 1, \quad \Phi_1(t) = (1 - \alpha_0 t)\Phi_0(t) \tag{4.64}$$

$$\Phi_{j+1}(t) = (1 + \frac{\beta_{j-1}}{\alpha_{j-1}}\alpha_j - \alpha_j t)\Phi_j(t) - \frac{\beta_{j-1}}{\alpha_{j-1}}\alpha_j\Phi_{j-1}(t), \quad j = 1, 2, \ldots. \tag{4.65}$$

Então, introduzindo dois parâmetros independentes, ζ_j e η_j, obtém-se um modelo para o polinômio Θ_j baseado nas relações de recorrência de três termos (4.64) e (4.65),

$$\Theta_0(t) = 1 \tag{4.66}$$

$$\Theta_1(t) = (1 - \zeta_0 t)\Theta_0(t), \tag{4.67}$$

$$\Theta_{j+1}(t) = (1 + \eta_j - \zeta_j t)\Theta_j(t) - \eta_j\Theta_{j-1}(t) \quad j = 1, 2, \ldots \tag{4.68}$$

onde os parâmetros η_j e ζ_j precisam ser determinados.

As relações de recorrência para os produtos polinomiais $\Theta_j\Phi_j$, $\Theta_j\Phi_{j+1}$, $(\Theta_{j-1} - \Theta_j)\Phi_{j+1}$, $\Theta_j\Pi_j$, e $\Theta_{j-1}\Pi_j$ são obtidas através das relações básicas (4.17) e (4.18), juntamente com a recorrência de três termos (4.68), definida para Θ_j.

Assim,

$$\Theta_{j+1}\Phi_{j+1} = \Theta_j\Phi_{j+1} - \eta - j(\Theta_{j-1} - \Theta_j)\Phi_{j+1} - \zeta_j t\Theta_j\Phi_{j+1} \tag{4.69}$$

$$\Theta_j\Phi_{j+1} = \Theta_j\Phi_j - \alpha_j t\Theta_j\Pi_j, \tag{4.70}$$

$$(\Theta_{j-1} - \Theta_j)\Phi_{j+1} = \Theta_{j-1}\Phi_j - \Theta_j\Phi_{j+1} - \alpha_j t\Theta_{j-1}\Pi_j, \tag{4.71}$$

$$\Theta_{j+1}\Pi_{j+1} = \Theta_{j+1}\Phi_{j+1} - \beta_j\eta_j\Theta_{j-1}\Pi_j + \beta_j(1 + \eta_j)\Theta_j\Pi_j - $$

$$-\beta_j\zeta_j t\Theta_j\Pi_j, \tag{4.72}$$

$$\Theta_j\Pi_{j+1} = \Theta_j\Phi_{j+1} + \beta_j\Theta_j\Pi_j. \tag{4.73}$$

Usando então o vetor residual definido pelo produto polinomial (4.63), e

utilizando os vetores auxiliares

$$\begin{aligned} t_j &= \Theta_j(A)\Phi_{j+1}r_0, \\ y_j &= (\Theta_{j-1}(A) - \Theta_j(A))\Phi_{j+1}r_0, \\ p_j &= \Theta_j(A)\Pi_j(A)r_0, \\ s_j &= \Theta_{j-1}(A)\Pi_j(A)r_0, \end{aligned} \tag{4.74}$$

escrevemos as relações (4.69) a (4.73) nas seguintes formas vetoriais

$$r_{j+1} = t_j - \eta_j y_j - \zeta_j A t_j, \tag{4.75}$$

$$t_j = r_j - \alpha_j A p_j, \tag{4.76}$$

$$y_j = t_{j-1} - t_j - \alpha_j A s_j, \tag{4.77}$$

$$p_{j+1} = r_{j+1} - \beta_j \eta_j s_j + \beta_j(1+\eta_j)p_j - \beta_j \zeta_j A p_j, \tag{4.78}$$

$$s_{j+1} = t_j + \beta_j p_j. \tag{4.79}$$

Além disso, levando-se em conta que o vetor resíduo é definido por

$$r_j = \Theta_j(A)\Phi_j(A)r_0 = b - Ax_j,$$

a fórmula para atualizar x_{j+1} torna-se,

$$x_{j+1} = -\eta_j(x_{j-1} + \alpha_{j-1}p_{j-1} + \alpha_j s_j) + (1+\eta_j)(x_j + \alpha_j p_j) + \zeta_j t_j. \tag{4.80}$$

Definidas então as fórmulas de recorrência na forma vetorial, e a fórmula de atualização da solução aproximada x_{j+1}, o próximo passo seria encontrar fórmulas para calcular as constantes necessárias e então produzir o algoritmo. No entanto, observe que a fórmula (4.80) é um pouco complicada e inadequada, uma vez que o cálculo de x_{j+1} exige o uso de x_j e x_{j-1}. Uma simplificação de (4.80) pode ser obtida definindo um polinômio auxiliar e reescrevendo a relação de recorrência para Θ_{j+1} em (4.68)

Note que $\Theta_j(t)$ satisfaz $\Theta_j(0) = 1$ para qualquer j. Então, $\Theta_{j+1}(0) - \Theta_j(0) = 0$ para qualquer j. Assim, o polinômio auxiliar de grau j denotado por $\Gamma_j(t)$, é definido como

$$\Gamma_j(t) = \frac{\Theta_j(t) - \Theta_{j+1}(t)}{\zeta_j(t)}.$$

É possível agora reescrever a relação de recorrência (4.68) e obter um novo modelo para o polinômio Θ_j, agora envolvendo também o polinômio auxiliar Γ_j. Essas novas relações são escritas como

$$\begin{aligned}
\Theta_0(t) &= 1, \qquad \Gamma_0(t) = 1 && (4.81)\\
\Theta_{j+1}(t) &= \Theta_j(t) - \zeta_j t\Gamma_j(t) && (4.82)\\
\Gamma_{j+1}(t) &= \Theta_{j+1}(t) + \zeta_j \frac{\eta_{j+1}}{\zeta_{j+1}}\Gamma_j(t) \quad j = 1, 2, \ldots. && (4.83)
\end{aligned}$$

Observe que uma vez que a relação para o polinômio Θ_j foi reformulada, é preciso também reformular as relações de recorrência para os produtos polinomiais (4.69 - 4.73). Substituindo as novas fórmulas (4.82) e (4.83) nas relações para os produtos, e realizando algumas operações algébricas, obtém-se

$$\begin{aligned}
\Theta_{j+1}\Phi_{j+1} &= \Theta_j\Phi_{j+1} - \eta_j\zeta_{j-1}t\Gamma_{j-1}\Phi_{j+1} - \zeta_j t\Theta_j\Phi_{j+1}, && (4.84)\\
&= \Theta_j\Phi_j - \alpha_j t\Theta_j\Pi_j - \zeta_j t\Gamma_j\Phi_{j+1}, && (4.85)\\
\Theta_j\Phi_{j+1} &= \Theta_j\Phi_j - \alpha_j t\Theta_j\Pi_j, && (4.86)\\
\zeta_j t\Gamma_j\Phi_{j+2} &= \Theta_j\Phi_{j+1} - \Theta_{j+1}\Phi_{j+1} - \alpha_{j+1}t\Theta_j\Pi_{j+1} + \alpha_{j+1}t\Theta_{j+1}\Pi_{j+1}, && (4.87)\\
\Theta_{j+1}\Pi_{j+1} &= \Theta_{j+1}\Phi_{j+1} + \beta_j\Theta_j\Pi_j - \beta_j\zeta_j t\Gamma_j\Pi_j, && (4.88)\\
t\Theta_j\Pi_{j+1} &= t\Theta_j\Phi_{j+1} + \beta_j t\Theta_j\Pi_j, && (4.89)\\
\zeta_j t\Gamma_j\Pi_j &= \zeta_j t\Theta_j\Pi_j + \eta_j(\Theta_{j-1}\Phi_j - \Theta_j\Phi_j + \beta_{j-1}\zeta_{j-1}t\Gamma_{j-1}\Pi_{j-1}), && (4.90)\\
\zeta_j\Gamma_j\Phi_{j+1} &= \zeta_j\Theta_j\Phi_j + \eta_j\zeta_{j-1}\Gamma_{j-1}\Phi_j - \alpha_j\zeta_j t\Gamma_j\Pi_j. && (4.91)
\end{aligned}$$

onde agora, como era de se esperar, os produtos polinomiais envolvem também o polinômio auxiliar Γ_j.

Definindo também os novos vetores w_j, u_j e z_j como

$$\begin{aligned}
w_j &= A\Theta_j(A)\Pi_{j+1}(A)r_0, && (4.92)\\
u_j &= \zeta_j A\Gamma_j(A)\Pi_j(A)r_0, && (4.93)\\
z_j &= \zeta_j\Gamma_j(A)\Phi_{j+1}(A)r_0, && (4.94)
\end{aligned}$$

as relações (4.84-4.91) podem ser escritas vetorialmente como

$$r_{j+1} = t_j - \eta_j y_j - \zeta_j A t_j \qquad (4.95)$$

$$
\begin{aligned}
&= r_j - \alpha_j A p_j - A z_j, && (4.96)\\
t_j &= r_j - \alpha_j A p_j, && (4.97)\\
y_{j+1} &= t_j - r_{j+1} - \alpha_{j+1} w_j + \alpha_{j+1} A p_{j+1}, && (4.98)\\
p_{j+1} &= r_{j+1} + \beta_j (p_j - u_j), && (4.99)\\
w_j &= A t_j + \beta_j A p_j, && (4.100)\\
u_j &= \zeta_j A p_j + \eta_j (t_{j-1} - r_j + \beta_{j-1} u_{j-1}), && (4.101)\\
z_j &= \zeta_j r_j + \eta_j z_{j-1} - \alpha_j u_j. && (4.102)
\end{aligned}
$$

onde os vetores $t_j, y_j, p_j es_j$ definidos em (4.74) também foram utilizados.

Assim, a partir de (4.96) uma forma mais compacta e adequada para atualizar x_{j+1} é obtida como

$$x_{j+1} = x_j + \alpha_j p_j + z_j, \tag{4.103}$$

que, claramente, é matematicamente equivalente à fórmula definida em (4.80).

Os escalares α_j e β_j são obtidos como no método BiCGSTAB. É feita uma expansão no polinômio Θ_j e os produtos internos (r_0^*, r_j) e (r_0^*, Ap_j) são calculados considerando apenas o termo de maior ordem da expansão. Como resultado,

$$
\begin{aligned}
\alpha_j &= \frac{(r_0^*, r_j)}{(r_0^*, A p_j)}\\
\beta_j &= \frac{\alpha_j}{\zeta_j} \cdot \frac{(r_0^*, r_{j+1})}{(r_0^*, r_j)}.
\end{aligned}
$$

Os parâmetros adicionais η_j e ζ_j são obtidos minimizando a norma 2 do vetor resíduo como uma função de ζ_j e η_j,

$$f(\zeta, \eta) = \|r_{j+1}\| = \|t_j - \eta y_j - \zeta A t_j\|,$$

produzindo o seguinte algoritmo para o método GPBiCG.

ALGORITMO 4.5: Método GPBiCG.

1. Escolha uma estimativa inicial x_0, e calcule $r_0 = b - Ax_0$
2. Escolha um vetor r_0^* arbitrário, tal que $(r_0, r_0^*) \neq 0$, por exemplo, $r_0^* = r_0$
3. Atribua $t_{-1} = w_{-1} = 0,\ \beta_{-1} = 0$
4. Para $n = 0, 1, \ldots,$ até convergência, faça:
5. $\quad p_n = r_n + \beta_{n-1}(p_{n-1} - u_{n-1})$
6. $\quad \alpha_n = \frac{(r_n, r_0^*)}{(Ap_n, r_0^*)}$
7. $\quad y_n = t_{n-1} - r_n - \alpha_n w_{n-1} + \alpha_n Ap_n$
8. $\quad t_n = r_n - \alpha_n Ap_n$
9. $\quad \zeta_n = \frac{(y_n,y_n)(At_n,t_n)-(y_n,t_n)(At_n,y_n)}{(At_n,At_n)(y_n,y_n)-(y_n,At_n)(At_n,y_n)}$
10. $\quad \eta_n = \frac{(At_n,At_n)(y_n,t_n)-(y_n,At_n)(At_n,t_n)}{(At_n,At_n)(y_n,y_n)-(y_n,At_n)(At_n,y_n)}$
11. $\quad$ Se $n = 0$, $\zeta_n = \frac{(At_n,t_n)}{(At_n,At_n)},\ \ \eta_n = 0$
12. $\quad u_n = \zeta_n Ap_n + \eta_n(t_{n-1} - r_n + \beta_{n-1}u_{n-1}$
13. $\quad z_n = \zeta_n r_n + \eta_n z_{n-1} - \alpha_n u_n$
14. $\quad x_{n+1} = x_n + \alpha_n p_n + z_n$
15. $\quad r_{n+1} = t_n - \eta_n y_n - \zeta_n At_n$
16. $\quad \beta_n = \frac{\alpha_n}{\zeta_n} \cdot \frac{(r_{n+1}, r_0^*)}{(r_n, r_0^*)}$
17. $\quad w_n = At_n + \beta_n Ap_n$
18. fim.

Este método é visto como uma generalização dos métodos tipo produto porque através da escolha de diferentes valores para os parâmetros ζ, η é possível obter métodos matematicamente equivalentes aos métodos tipo produto apresentados neste trabalho. Escolhendo, por exemplo, $\zeta_j = \alpha_j$ e $\eta_j = \frac{\beta_{j-1}}{\alpha_{j-1}}\alpha_j$ obtém-se uma variante matematicamente equivalente ao CGS.

4.4 Uma implementação do método BiCG sem uso da matriz transposta: TFBICG

Uma outra forma de obter uma formulação "transpose free"baseada no BiCG, é utilizar métodos distintos para desenvolvimento de versões híbridas. Um exemplo desse procedimento utiliza os métodos CGS e BiCG para criar uma versão do método BiCG sem o uso da matriz transposta, chamada de TFBiCG [9].

O algoritmo obtido é apresentado a seguir, usando os subescritos BCG e CGS para denotar os vetores extraídos do método BiCG e CGS, respectivamente.

ALGORITMO 4.6: Método TFBiCG.

1. Escolha uma estimativa inicial x_0, e calcule $r_0 = b - Ax_0$
2. Escolha r_0^* arbitrário, tal que $(r_0, r_0^*) \neq 0$
3. Atribua $p_0^{BCG} = 0;\ \rho_0 = 1;$
4. Atribua $p_0^{CGS} = 0;\ q_0 = 0;$
5. Para $i = 1, 2, \ldots,$ até convergência, faça:
6. $\quad \rho_i = (r_0^*, r_{i-1}^{CGS})$
7. $\quad \beta = \frac{\rho_i}{\rho_{i-1}}$
8. $\quad u = r_{i-1}^{CGS} + \beta q_{i-1}$
9. $\quad p_i^{CGS} = u + \beta(q_{i-1} + \beta p_{i-1}^{CGS})$
10. $\quad p_i^{BCG} = r_i^{BCG} + \beta p_{i-1}^{BCG}$
11. $\quad v = Ap_i^{CGS}$
12. $\quad \alpha = \frac{\rho_i}{(r_0^*, v)}$
13. $\quad q_i = u - \alpha v$
14. $\quad \widetilde{u} = u + q_i$
15. $\quad r_i^{CGS} = r_{i-1}^{CGS} - \alpha A\widetilde{u}$
16. $\quad x_i^{BCG} = x_{i-1}^{BCG} + \alpha p_i^{BCG}$
17. $\quad r_i^{BCG} = r_{i-1}^{BCG} - \alpha Ap_i^{BCG}$

18. fim.

O algoritmo acima elimina o uso da matriz A^T a um custo extra de um produto da matriz A por um vetor por iteração. Conseqüentemente, o custo computacional por iteração é ligeiramente maior que a maioria dos outros métodos. Por outro lado, o método TFBiCG apresenta uma convergência mais suave que o CGS em muitos casos.

Capítulo 5

Técnicas de pré-condicionamento

Atualmente os métodos no subespaço de Krylov representam uma das principais técnicas para a solução de grandes sistemas lineares esparsos. Porém, quando aplicados à problemas realísticos, esses métodos sozinhos dificilmente funcionam. Isso geralmente ocorre porque a convergência dos métodos iterativos depende das propriedades espectrais da matriz [16] que, em casos reais, muitas vezes não são ideais.

Nesses casos a solução usual é aplicar uma transformação linear adequada no sistema linear para melhorar as propriedades espectrais da matriz. Este processo é conhecido como pré-condicionamento.

Na prática, aplicar um pré-condicionador significa substituir o sistema original

$$Ax = b, \tag{5.1}$$

por um sistema equivalente

$$MAx = Mb \text{ ou } AMy = b, \tag{5.2}$$

que seja, obviamente, mas fácil de ser resolvido.

O problema geral pode ser definido então como encontrar um pré-condicionador M cuja construção seja viável, em termos de custo computacional, e que o sistema $My = z$ seja mais fácil de ser resolvido que o

sistema original. Duas classes dessas técnicas, que serão tratadas neste trabalho, são os pré-condicionadores baseados da fatorização LU da matriz original A, e os pré-condicionadores baseados na aproximação direta da inversa da matriz original. Os pré-condicionadores baseados na fatorização LU são conhecidos como pré-condicionadores implícitos uma vez que a inversa da matriz original não é obtida explicitamente, mas somente aplicada a um vetor. Como resultado esses métodos exigem a solução de sistemas triangulares envolvendo os fatores L e U obtidos do processo de fatorização. Os pré-condicionadores da outra classe, por aproximarem diretamente a inversa da matriz são conhecidos como pré-condicionadores explícitos.

5.1 Pré-condicionadores implícitos

As técnicas de pré-condicionamento mais comuns são baseadas no uso de métodos diretos de forma incompleta. Nesse processo, o método direto é aplicado, e parte dos cálculos são desprezados de acordo com uma regra pré-estabelecida. Isto conduz à noção de fatorização LU incompleta (ILU).

Na prática, as fatorizações são realizadas de forma incompleta para preservar a esparsidade das matrizes. Isto é necessário porque quando se aplica, por exemplo, uma fatorização LU completa em uma matriz esparsa, ela tende a se tornar densa em função do grande número de elementos que são gerados pelo processo de fatorização (os chamados *"fill-in"*). É claro que se pudéssemos trabalhar com matrizes densas seria mais adequado resolver o problema aplicando um método direto, o que dispensaria a construção de um pré-condicionador.

Um algoritmo geral para construção de uma fatorização ILU é obtido da realização de um processo de eliminação Gaussiana no qual alguns elementos não diagonais são eliminados. Isto normalmente é feito escolhendo algum padrão de esparsidade P, isto é, escolhendo um conjunto de posições na matriz que necessariamente deverão conter elemetos nulos. Esse conjunto P é definido

como

$$P \subset \{(i,j) : i \neq j, 1 \leq i,j \leq n\} \tag{5.3}$$

e realizando uma fatorização ILU que considere apenas os elementos que não estão nas posições determinadas pelo padrão de esparsidade escolhido. Esse procedimento geral é formalizado a seguir.

ALGORITMO 5.1: Fatorização ILU geral.

1. Para $k = 1, \ldots, n-1$ Faça:
2. Para $i = k+1, \ldots, n$ e se $(i,k) \notin P$ faça:
3. $a_{ik} = a_{ik}/a_{kk}$
4. para $j = k+1, \ldots, n$ e para $(i,j) \notin P$ faça:
5. $a_{ij} = a_{ij} - a_{ik} * a_{kj}$
6. fim
7. fim
8. Fim.

O processo de eliminação Gaussiana é um procedimento com três laços que podem ser arranjados de várias formas. Na verdade, o algoritmo de fatorização ILU geral, descrito acima, pode ser desenvolvido utilizando qualquer uma das possíveis formulações do processo de eliminação Gaussiana. Na prática, porém, a formulação utilizada no algoritmo 5.1 não é a melhor opção do ponto de vista numérico. Observe que o algoritmo 5.1 percorre todas as colunas k, com exeção da última, e para cada uma dessas colunas trata todas as linhas i tais que $i > k$. Dessa forma, uma linha i, por exemplo, precisa ser acessada para as colunas $k = 1, 2, \ldots, i-1$. Uma alternativa mais atraente consiste em acessar cada linha da matriz apenas uma vez durante todo o processo. Nesse caso, cada linha, com exceção da primeira, é acessada e só é abandonada depois que todos os cálculos que a envolvem forem realizados. Esta versão é conhecida como variante ikj do processo de eliminação Gaussiana [16] e será usada ao longo

desse trabalho.

A regra usada para controlar a eliminação desses elementos, dá origem aos diversos pré-condicionadores dessa classe.

5.1.1 Fatorização ILU(0)

A forma mais simples e barata de impor uma regra numérica para controlar a entrada de novos elementos na matriz esparsa original, é impor que as matrizes L e U obtidas com a fatorização, tenham a mesma estrutura esparsa das partes triangular inferior e triangular superior da matriz original, respectivamente. Ou seja, escolher o padrão de esparsidade P como sendo exatamente o mesmo padrão de esparsidade da matriz A. Essa formulação da origem ao algoritmo ILU(0) que é apresentado a seguir usando a variante ikj do algoritmo 5.1

ALGORITMO 5.2: Fatorização ILU(0).

1. Para $i = 2, \ldots, n$ Faça:
2. Para $k = 1, \ldots, i-1$ e para $(i,k) \in NZ(A)$ faça:
3. Calcule $a_{ik} = a_{ik}/a_{kk}$
4. Para $j = k+1, \ldots, n$ e para $(i,j) \in NZ(A)$, faça:
5. Calcule $a_{ij} = a_{ij} - a_{ik} \times a_{kj}$.
6. fim
7. fim
8. Fim.

Neste algoritmo $NZ(A)$ representa o conjunto $\{(i,j) : a_{i,j} \neq 0\}$, dessa forma o algoritmo produz os fatores L e U, triangular inferior e triangular superior, respectivamente, que juntas possuem o mesmo padrão de esparsidade da matriz A. Ou seja, um elemento (i,j) da matriz L ou da matriz U só é diferente de zero se o elemento correspondente na matriz A também for diferente de zero. Na prática, esses fatores são armazenados em uma única

matriz, e a diagonal principal de L, cujos elementos são todos iguais a 1, não é armazenada.

5.1.2 Fatorização ILU(p)

Em problemas um pouco mais complexos a precisão do pré-condicionador obtido através de uma fatorização ILU(0) pode não ser suficiente para garantir uma boa condição para a convergência dos métodos iterativos. Nestas situações é preciso buscar pré-condicionadores mais eficientes no sentido de uma melhor aproximação da matriz. No contexto da fatorização ILU isto claramente significa desenvolver formulações que permitam a entrada de novos elementos em posições originariamente nulas. Para isto, é preciso introduzir o conceito de nível de *"fill"*.

É atribuído um nível de *"fill"* para cada elemento da matriz e esse valor é atualizado sempre que o elemento é alterado pelo processo de eliminação. Então, uma regra numérica é aplicada com base no valor do nível de *"fill"* de cada elemento, determinando se o elemento será armazenado ou se ele será descartado. A fatorização ILU(1) por exemplo, retém todos os elementos cujo nível de *"fill"* não excede 1.

Nesse processo, a idéia é que o nível de *"fill"* seja uma indicação do valor absoluto do elemento: quanto maior o nível de *"fill"*, menor o valor do elemento. Dessa forma atribuindo o valor ϵ^k, com $\epsilon < 1$, a qualquer elemento cujo nível de *"fill"* é k, a atualização de a_{ij} na linha 5 do algoritmo 5.2 pela fórmula

$$a_{ij} = a_{ij} - a_{ik} \times a_{kj}, \tag{5.4}$$

produz um elemento cujo valor deve ser

$$a_{ij} = \epsilon^{level_{ij}} - \epsilon^{level_{ik}} \times \epsilon^{level_{kj}} = \epsilon^{level_{ij}} - \epsilon^{level_{ik}+level_{kj}},$$

onde $level_{ij}$ é o atual nível de *"fill"* do elemento a_{ij}, que inicialmente é definido

como

$$level_{ij} = \begin{cases} 1, & \text{se } a_{i,j} \neq 0, \\ \infty, & \text{se } a_{ij} = 0. \end{cases} \tag{5.5}$$

Como resultado, podemos assumir uma estimativa para o tamanho de a_{ij} como o máximo entre $\epsilon^{level_{ij}}$ e $\epsilon^{level_{ik}+level_{kj}}$, e naturalmente definir o novo nível de *"fill"* como,

$$level_{ij} = \min\{level_{ij}, level_{ik} + level_{kj}\}$$

No entanto, para se adequar a definição comum usada na literatura (veja [23]), devemos fazer inicialmente

$$level_{ij} = \begin{cases} 0, & \text{se } a_{ij} \neq 0, \\ \infty, & \text{caso contrário.} \end{cases} \tag{5.6}$$

Conseqüentemente, a fórmula de atualização do nível de *"fill"* se transforma em

$$level_{ij} = \min\{level_{ij}, level_{ik} + level_{kj} + 1\}. \tag{5.7}$$

Essas definições acima permitem definir a estratégia dos pré-condicionadores baseados na fatorização ILU(p), que retém todos os elementos cujo nível de *"fill"* não excedem p.

ALGORITMO 5.3: Fatorização ILU(p).

1. Para todos os elementos não nulos a_{ij} da matriz A defina $level_{ij} = 0$
2. Para $i = 2, \ldots, n$ faça:
3. Para cada $k = 1, \ldots, i-1$ e para $level_{ik} \leq p$ faça:
4. Calcule $a_{ik} = a_{ik}/a_{kk}$
5. Para $j = k+1, \ldots, n$ faça:
6. $a_{ij} = a_{ij} - a_{ik} \times a_{kj}$
7. Atualize os $level_{ij}$'s usando (5.7)
8. fim
9. fim
10. Substitua qualquer elemento na linha i com $level_{ij} > p$ por zero

11. fim.

Na prática existem várias formas possíveis para implementar o algoritmo ILU(p). Um procedimento comum é dividir o processo em duas etapas. A primeira etapa é conhecida como a fase simbólica na qual a estrutura dos fatores L e U são determinados. Nesta fase nenhum cálculo numérico é realizado. A segunda etapa se incumbe então de realizar todos os cálculos, armazenando os *"fill-in"* aceitos em suas respectivas posições, já criadas na etapa anterior.

Outro procedimento usual, definido pelo algoritmo 5.3, consiste em aumentar a matriz dinamicamente à medida que um novo elemento é aceito. Esse procedimento pode ser um pouco mais lento pois implica no uso de uma rotina que insira cada novo elemento na estrutura esparsa da matriz. Uma alternativa a esse procedimento é efetuar todas as alterações de uma linha i em um vetor cheio, isto é, em um vetor que armazena todos os elementos da linha inclusive os elementos nulos. Dessa forma a atualização dos fatores L e U pode ser feita linha por linha reduzindo o tempo computacional gasto na construção do pré-condicionador. Neste trabalho, a implementação computacional do ILU(p) foi feita adotando esse último procedimento.

5.1.3 Fatorização ILUT

Nas técnicas de pré-condicionamento descritas até agora a estratégia usada para controlar o *"fill-in"* nas matrizes L e U são baseadas unicamente na estrutura esparsa da matriz original. Esta estratégia é eficiente em alguns casos, porém, em muitos problemas reais pode causar algumas dificuldades. No caso específico do ILU(p) o nível de *"fill"* pode não ser um bom indicador do tamanho do elemento, fazendo com que o método retenha elementos pequenos e elimine elementos maiores. Como conseqüência a precisão da aproximação fica comprometida.

Para remediar essa dificuldade foi desenvolvida uma nova estratégia que é

semelhante ao ILU(0) mas que permite *"fill-in"* baseado no valor do elemento. Essa estratégia é chamada ILUT (ILU com *"Threshold"*) [24] e é apresentada agora usando a versão ikj da eliminação Gaussiana.

ALGORITMO 5.4: Fatorização ILUT

1. Para $i = 1, \ldots, n$ faça:
2. $w = a_{i*}$
3. Para $k = 1, \ldots, i-1$, e quando $w_k \neq 0$ faça:
4. $w_k = w_k / a_{kk}$
5. Aplique uma regra de eliminação em w_k
6. Se $w_k \neq 0$, então
7. Para $j = k+1, \ldots, n$ faça:
8. $w_j = w_j - w_k \times u_{kj}$
9. Fim
10. Fim
11. Aplique uma regra de eliminação na linha w
12. $l_{i,j} = w_j$, para $j = 1, \ldots, i-1$
13. $u_{i,j} = w_j$, para $j = i, \ldots, n$
14. $w = 0$
15. Fim

A notação a_{i*} utilizada na linha 2 do algoritmo acima significa a i-ésima linha da matriz A. Isto mostra que este algoritmo utiliza o último procedimento descrito na subseção anterior. Na prática, como dissemos, os fatores L e U são armazenados em uma única matriz. Como conseqüência a atualização desses fatores pode ser feita linha por linha. Nesse processo, os elementos não nulos em w são transferidos para um outro vetor, enquanto é criado um vetor que contém as posições das colunas de cada elemento não nulo dessa linha. Isto é

ilustrado na figura 5.1.

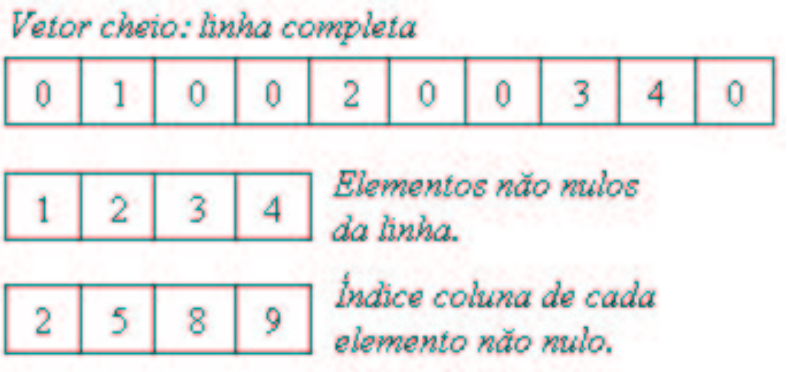


Figura 5.1: *Estrutura de dados usada para os fatores L e U em ILUT.*

Como resultado da utilização da estrutura ilustrada acima e levando em conta que as matrizes L e U são armazenadas juntas, podemos substituir as operações de atualização definidas nas linhas 12 e 13 do algoritmo 5.4 por

$$linha_j = \bar{w} \tag{5.8}$$

onde $\bar{w}$ denota o vetor onde apenas os elementos não nulos da linha são armazenados.

Com relação à regra de eliminação, duas estratégias são utilizadas. A primeira regra, aplicada na linha 5 do algoritmo, elimina um elemento w_k se este for menor que a tolerância relativa τ_i que é obtida multiplicando o número τ, que é um valor real passado pelo usuário, pela norma da $i-$ésima linha da matriz original. A outra regra, usada na linha 11 do algoritmo, usa dois procedimentos: primeiro elimina qualquer elemento da linha cujo valor absoluto é menor que τ_i. Então armazena somente os p maiores elementos na parte da linha que pertence a L e os p maiores elementos na parte da linha que pertence a U, além dos elementos diagonais que sempre são armazenados. O valor de p também é passado pelo usuário e assim como τ pode variar de acordo com o problema.

5.2 Deficiências dos pré-condicionadores ILU

Para matrizes esparsas gerais, que surgem em muitas aplicações reais, a fatorização ILU pode encontrar muitos problemas e até mesmo não existir. Isto pode ocorrer, por exemplo, quando a matriz possui elementos nulos, ou muito pequenos, na sua diagonal principal. No contexto dos pré-condicionadores ILU uma alternativa é o uso da estratégia do pivoteamento [16] de modo que o pivô seja o maior elemento, em valor absoluto, da coluna que está sendo trabalhada. No caso dos pré-condicionadores ILUT, por exemplo, basta combinar o algoritmo 5.4 com um procedimento de troca de linhas na matriz original. Uma sistemática para obter um pré-condicionador ILUT com pivoteamento (ILUTp) é apresentada pelo algoritmo 5.5.

ALGORITMO 5.5: ILUT com pivoteamento-ILUTp

1. Para $k = 1, \ldots, n-1$ Faça:
2. $\beta = \mid a_{kk} \mid$
3. Para $j = k+1, \ldots, n$ faça:
4. Se $|a_{jk}| > \beta$ então
5. $\beta = \mid a_{jk} \mid$ e $r = j$
6. Troca as linhas k e r,
7. Fim
8. Para $i = k+1, \ldots, n$ e se $a_{ik} \neq 0$ faça:
9. $a_{ik} = a_{ik}/a_{kk}$
10. Aplique uma regra de eliminação em a_{ik}
11. Se $a_{ik} \neq 0$ então
12. Para $j = k+1, \ldots, n$ faça:
13. $a_{ij} = a_{ij} - a_{ik} \times a_{kj}$
14. Fim
15. Aplique uma regra de eliminação na linha i
16. Fim

17. Fim.

Entretanto, existem situações comuns onde a matriz A é indefinida, para as quais apenas a adoção da estratégia de pivoteamento não é suficiente para resolver o problema da fatorização ILU. Nesses casos, além do problema de encontrar um pivô nulo durante o processo de fatorização, os fatores L e U obtidos podem ser muito instáveis, gerando pré-condicionadores muito pobres. Além disso, a natureza recursiva da fatorização ILU dificulta um eficiente uso dessas técnicas em implementações em paralelo ([3],[7]).

Para estas situações uma possível solução é encontrar um pré-condicionador que evite um processo de fatorização e que não exija a solução de um sistema linear. Isto é possível pela aproximação direta da inversa da matriz A.

5.3 Pré-condicionadores explícitos

Uma das técnicas mais simples para encontrar a inversa de uma matriz esparsa arbitrária é tentar encontrar uma matriz esparsa M que minimiza a norma da matriz residual $I - AM$,

$$F(M) = \|I - AM\|_F^2 \tag{5.9}$$

A matriz M que minimiza a função definida em (5.9) será uma inversa aproximada à direita de A. Da mesma forma podemos definir inversa aproximada à esquerda usando a função

$$\| I - MA \|_F^2, \tag{5.10}$$

ou um par de matrizes L e U obtidos pela minimização de

$$\| I - LAU \|_F^2 \ . \tag{5.11}$$

Nesses três casos, $\| \cdot \|_F$ denota a norma de Frobenius de uma matriz que é

dada por *

$$\|A\|_F \equiv \sqrt{\sum_{i,j} a_{i,j}^2}. \tag{5.12}$$

O uso da norma de Frobenius é especialmente útil para implementações em paralelo. Note que em razão da utilização dessa norma a função (5.9) pode ser decomposta em n subproblemas independentes envolvendo as colunas da matriz residual $I - AM$ da seguinte forma,

$$\begin{aligned} F(M) &= \| I - AM \|_F^2 \\ &= \| e_1 - Am_1, \ \ldots, \ e_n - Am_n \|_F^2 \\ &= \sum_{i,j} r_{i,j}^2, \end{aligned} \tag{5.13}$$

onde $r_j = e_j - Am_j$.

Mas,

$$\sum_{i=1}^{n} r_j^2 = \| r_j \|_2^2,$$

e portanto,

$$\| I - AM \|_F^2 = \sum_{j=1}^{n} \| e_j - Am_j \|_2^2, \tag{5.14}$$

onde e_j e m_j são as j-ésimas colunas da matriz identidade e da matriz M, respectivamente.

Como resultado dos cálculos acima, a função (5.9) pode ser minimizada de duas maneiras: ou como uma função da matriz esparsa M, denotada por função global, usando um método iterativo qualquer, ou de uma forma alternativa minimizando individualmente cada termo do somatório no segundo membro de (5.14).

*A norma de Frobenius de uma matriz A é definida também como $\sqrt{tr(A^T A)}$ onde $tr(\cdot)$ é a função traço dada por $tr(A) = \sum_i a_{ii}$

5.3.1 Minimização da função global

Esta técnica trata a matriz esparsa M como uma incógnita e usa um método tipo descida para minimizar a função objetivo (5.9). Trata-se de uma função quadrática no espaço das matrizes $n \times n$, vistas com objetos no $\mathbb{R}^{n^2}$. Neste espaço de matrizes, o produto interno ao qual a função (5.9) está associado é

$$\langle X, Y \rangle = tr(Y^T X), \tag{5.15}$$

onde $tr(\cdot)$ é a função traço.

Aqui usamos uma representação vetorial de um objeto $X \in \mathbb{R}^{n^2}$que significa a matriz $n \times n$ cujos vetores coluna são os sucessivos n-vetores de X.

Utilizando então um método de descida, temos que uma nova aproximação M_{new} é obtida a partir da aproximação atual, tomando-se um passo ao longo de uma direção selecionada G, isto é,

$$M_{new} = M + \alpha G, \tag{5.16}$$

onde G é a direção de descida e α é o tamanho do passo dado ao longo da direção G, escolhido de modo a minimizar a função objetivo

$$F(M_{new}) = \| I - AM_{new} \|_F^2.$$

Assim, definindo $R_{new} = I - AM_{new}$, e da mesma forma $R = I - AM$,

$$\begin{aligned} R_{new} &= I - AM_{new} \\ &= I - AM - \alpha AG \\ &= R - \alpha AG, \end{aligned} \tag{5.17}$$

vem que minimizar $F(M_{new})$ significa escolher um valor de α tal que R_{new} seja ortogonal a AG com respeito ao produto interno associado, definido em (5.15).

Impondo essa condição obtém-se,

$$\begin{aligned} 0 &= \langle R_{new}, AG \rangle \\ &= \langle R - \alpha AG, AG \rangle \\ &= \langle R, AG \rangle - \alpha \langle AG, AG \rangle. \end{aligned}$$

Como resultado,

$$\alpha = \frac{\langle R, AG \rangle}{\langle AG, AG \rangle} = \frac{tr(R^T AG)}{tr((AG)^T AG)}, \tag{5.18}$$

Onde o denominador pode ser computado como

$$\| AG \|_F^2.$$

O algoritmo 5.6 apresentado a seguir, descreve um procedimento geral de minimização global.

ALGORITMO 5.6: Procedimento geral de minimização da função global.

1. Escolha uma estimativa inicial para M.
2. Até convergência faça:
3. Escolha uma direção de descida G
4. Calcule α usando (5.18)
5. Calcule $M = M + \alpha G$
6. Aplique uma regra de eliminação em M
7. Fim.

Existem alguns importantes aspectos a serem discutidos com relação ao algoritmo 5.6. O primeiro diz respeito à escolha da direção G no passo 3 do algoritmo. Usualmente existem duas possibilidades. A escolha mais simples consiste em tomar $G = R = I - AM$, onde M é a aproximação no passo atual. Se desconsiderarmos a regra de eliminação essa escolha produz o algoritmo do resíduo mínimo (MR) descrito no capítulo 2 deste trabalho. A segunda escolha possível é tomar G como sendo a direção oposta ao gradiente da função (5.9) com respeito a matriz M. Usando novamente uma representação vetorial das matrizes, temos que este gradiente é dado por $\nabla F = -2A^T R$ [10]. Essa escolha de G reproduz o algoritmo da descida máxima descrito no capítulo 2 que pode ser muito lento em algumas situações[10].

Independente da escolha feita, a matriz G precisa ser armazenada explicitamente. Os escalares $\| AG \|_F^2$ e $tr(R^T AG)$ utilizados no cálculo de α podem ser calculados de forma progressiva utilizando cada coluna de AG. Assim, a matriz AG não precisa ser armazenada.

Outro aspecto importante do algoritmo 5.6 diz respeito à regra de eliminação aplicada à matriz M. Como é usual (veja [8], [26]) um valor de tolerância é selecionado e todo elemento de M menor, em valor absoluto, que esse valor de tolerância, é descartado (substituído por zero). No entanto, quando essa regra é aplicada à matriz M a propriedade de descida pode ser perdida, ou seja, não se pode mais garantir que $F(M_{new}) \leq F(M)$. Uma alternativa usual [23] é aplicar a regra de eliminação na matriz G antes de calcular o passo α. Neste caso, a quantidade de *"fill-in"* na matriz M não pode ser controlada.

Por fim, vale comentar o desempenho do algoritmo 5.6 com relação ao custo computacional. Note que o algoritmo envolve operações matriz-matriz, que são operações BLAS [11] nível 3 que, em função do alto custo computacional, devem ser evitadas.

5.3.2 Algoritmo orientado por coluna

Este algoritmo minimiza a função (5.9) através dos n subproblemas lineares definidos em (5.14). Essa minimização é feita tomando uma estimativa inicial para cada coluna m_j da matriz M e resolvendo aproximadamente os subproblemas

$$Am_j = e_j, \quad j = 1, 2, \ldots, n, \tag{5.19}$$

com alguns passos de um método tipo descida como, por exemplo, o método MR. Uma sistemática para isso é apresentada pelo algoritmo a seguir, usando n_i iterações para resolver (5.19) para cada coluna da inversa aproximada da matriz A. Nesse processo, cada m_j inicial é obtido das colunas de uma matriz M_0 que representa uma estimativa inicial para a inversa aproximada de A.

ALGORITMO 5.7: Minimização por Coluna via algorítmo MR.

1. Escolha uma estimativa inicial para M.
2. Para cada coluna $j = 1, \ldots, n$ faça:
3. Defina $m_j = Me_j$
4. Para $i = 1, \ldots, n_i$ faça:
5. $r_j = e_j - Am_j$
6. $\alpha_j = \frac{(r_j, Ar_j)}{(Ar_j, Ar_j)}$
7. $m_j = m_j + \alpha_j r_j$
8. Aplique uma regra de eliminação em m_j
9. Fim
10. Fim.

A escolha da matriz M_0 na linha inicial do algoritmo 5.7 pode ser feita de várias maneiras. Existem, no entanto, duas escolhas mais naturais que são: $M_0 = \alpha I$ e $M_0 = \alpha A^T$, onde o fator α é escolhido de modo a minimizar a norma de $I - AM_0$. A primeira escolha para M_0 é mais barata em termos computacionais porém, $M_0 = \alpha A^T$ produz em muitos casos uma estimativa inicial muito mais eficiente [23].

Outro aspecto importante do algoritmo 5.7 diz respeito a forma de armazenamento utilizada na implementação. Observe que a matriz inversa aproximada é calculada por coluna, isso sugere que essa matriz deve ser armazenada também por coluna, permitindo uma atualização muito mais eficiente. Após aplicar uma regra de eliminação no vetor-coluna m_j cria-se uma estrutura de dados similar aquela ilustrada na figura 5.1. Então os elementos não nulos de m_j são armazenados nessa estrutura juntamente com os índices das linhas de cada elemento não nulo.

Após os n passos, obtém-se uma inversa aproximada esparsa da matriz A que pode ser usada como um pré-condicionador explícito para um sistema linear envolvendo A.

Uma variante do procedimento descrito acima pode ser obtido acrescen-

tando um laço externo ao algoritmo 5.7. Nesse novo procedimento a matriz M vai sendo atualizada em cada passo e é então utilizada para pré-condicionar o sistema no passo seguinte. Este procedimento é conhecido como MR com auto-precondicionamento [10] e fornece, em muitos casos, um pré-condicionador muito mais confiável.

5.3.3 Inversa aproximada de matrizes simétricas

As técnicas de pré-condicionamento descritas até agora, baseadas na minimização da norma de Frobenius da matriz residual, são desenvolvidas para o tratamento de matrizes não simétricas. Quando a matriz A é simétrica existem outras técnicas de aproximação da inversa que são mais adequadas uma vez que exploram a simetria da matriz, produzindo inversas aproximadas que também são simétricas.

Um exemplo dessas técnicas são as que calculam inversas aproximadas triangulares preservando a simetria no sistema pré-condicionado.

Além de explorar a simetria da matriz outra vantagem dessas técnicas é que, em função de calcularem a inversa aproxima decomposta em fatores triangulares, fica fácil detectar com antecipação a não singularidade da inversa obtida.

Nesse sentido Kolotilina e Yeremin desenvolveram um método que calcula uma inversa aproximada fatorizada. Nesse método, uma matriz triangular G é calculada como uma inversa aproximada da matriz L, onde $A = LL^T$ é a fatorização de Cholesky [20] da matriz A. Este algoritmo não requer qualquer informação sobre a matriz L e trabalha exclusivamente com a matriz A [18]. Por produzirem sistemas pré-condicionados simétricos definidos positivos, quando aplicados a matrizes com estas características, esse método pode ser eficientemente usado com o método dos Gradientes Conjugados [2].

Existem ainda outras técnicas para aproximar a inversa de matrizes simétricas, das quais destacam-se duas. A primeira delas, proposta por Benzi

e Tuma e denotada por AINV [4], calcula uma inversa aproximada fatorizada com base num processo de conjugação (A-ortogonalização)dos vetores base unitários. Para preservar a esparsidade da matriz inversa aproximada esse processo de conjugação é realizado incompletamente, e o padrão de esparsidade da inversa aproximada é determinado dinamicamente à medida em que cada vetor é gerado. A seguir esse processo será detalhado.

Assuma que a matriz $A \in I\!R^{n\times n}$ é simétrica definida positiva (SDP). O algoritmo AINV constrói uma inversa aproximada fatorizada da forma

$$M = ZD^{-1}Z^T \approx A^{-1}, \tag{5.20}$$

onde Z é uma matriz triangular superior, com os elementos da diagonal principal iguais a 1, e D uma matriz diagonal.

A matriz Z é uma inversa aproximada esparsa do fator L^T da decomposição LDL^T de A [2]. Este procedimento é descrito pelo algoritmo 5.8, onde a_i^T denota a $i-$ésima linha da matriz A e e_i denota o $i-$ésimo vetor base unitário.

ALGORITMO 5.8: AINV:Inversa Aproximada fatorada simétrica.

1. Faça $z_i^{(0)} = e_i$ $(1 \leq i \leq n)$.
2. Para $i = 1, 2, \ldots, n$ faça:
3. Para $j = i, i+1, \ldots, n$ faça:
4. $p_j^{(i-1)} = a_i^T z_j^{(i-1)}$
5. Fim
6. Se $i = n$
7. Faça $z_i = z_i^{(i-1)}$ e $p_i = p_i^{(i-1)}$ para $1 \leq i \leq n$. Retorne
8. Para $j = i+1, \ldots, n$ faça:
9. $z_j^{(i)} = z_j^{(i-1)} - (\frac{p_j^{(i-1)}}{p_i^{(i-1)}}) z_i^{(i-1)}$
10. Fim
11. Fim
12. $Z = [z_1, z_2, \ldots, z_n]$ e $D = diag(p_1, p_2, \ldots, p_n)$.

Na prática, a esparsidade da matriz Z é preservada aplicando uma regra de eliminação nos vetores z_i. Note que, assim como no algoritmo 5.7, é mais adequado que a matriz inversa aproximada seja armazenada por coluna, permitindo uma atualização mais eficiente da matriz Z. Observe ainda que, de forma semelhante aos pré-condicionadores ILU, o processo definido aqui pode falhar em determinados casos. Sempre que o algoritmo encontra um pivô nulo ($p_i = 0$) o processo é interrompido. Na tentativa de resolver esse problema surgiram versões estabilizadas desse procedimento que exploram a relação $Z^T AZ = D$ para desenvolver algoritmos sem situações de *breakdowns* [5].

A segunda técnica para matrizes simétricas tratada neste trabalho, calcula um fator L chamado *Fator Inverso Aproximado*. Na prática, o método consiste em buscar uma matriz triangular superior L, com os elementos da diagonal principal iguais a 1, tal que:

$$L^T AL \approx D, \tag{5.21}$$

onde D é uma matriz diagonal desconhecida.

Assumindo disponível uma seqüência de matrizes

$$A_{k+1} = \begin{pmatrix} A_k & v_k \\ v_k^T & \alpha_{k+1} \end{pmatrix} \tag{5.22}$$

na qual $A_n \equiv A$, e supondo que o fator L_k está disponível para a matriz A_k, isto é, supondo

$$L_k^T A_k L_k = D_k, \tag{5.23}$$

podemos então encontrar facilmente o fator L_{k+1} para a matriz A_{k+1}.

Para isso escrevemos

$$\begin{pmatrix} L_k^T & 0 \\ -z_k^T & 1 \end{pmatrix} \begin{pmatrix} A_k & v_k \\ v_k^T & \alpha_{k+1} \end{pmatrix} \begin{pmatrix} L_k & -z_k \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} D_k & 0 \\ 0 & \delta_{k+1} \end{pmatrix}, \tag{5.24}$$

que, aplicando a definição do produto de matrizes, produz

$$\begin{pmatrix} L_k^T A_k L_k & -L_k^T A_k z_k + L_k^T v_k \\ -z_k^T A_k L_k + v_k^T L_k & z_K^T A_k z_k - v_k^T z_k - z_k^T v_k + \alpha_{k+1} \end{pmatrix} = \begin{pmatrix} D_k & 0 \\ 0 & \delta_{k+1} \end{pmatrix}. \tag{5.25}$$

Como resultado

$$A_k z_k = v_k \tag{5.26}$$

$$\delta_{k+1} = z_k^T A z_k - 2v_k^T z_k + \alpha_{k+1}. \tag{5.27}$$

Assim, no passo k do processo obtemos a coluna $k+1$ do fator L_{k+1}, resolvendo um sistema linear com A_k, e obtemos o elemento $k+1$ da matriz diagonal D através da relação (5.27). Além disso, se a matriz D é não singular, então L é chamado fator inverso de A pois nesse caso $A^{-1} = L^T D^{-1} L$. Conseqüentemente, se D_k for não singular, a matriz L_k poderá ser usada para pré-condicionar o sistema linear que envolve a matriz A_k em cada passo.

Todo esse processo é formalizado pelo algoritmo a seguir.

ALGORITMO 5.9: Fator inverso aproximado.

1. Para $k = 1, \ldots, n-1$ Faça:
2. resolva $A_k z_k = v_k$
3. calcule $\delta_{k+1} = z_k^T A z_k - 2v_k^T z_k + \alpha_{k+1}$
4. aplique uma regra de eliminação no vetor z_k
5. Para $i = 1, \ldots, k$ e se $z_k(i) \neq 0$ faça:
6. $l_{i,k+1} = -z_k(i)$

7 fim

8 $l_{k+1,k+1} = 1$

9. $d_{k+1,k+1} = \delta_{k+1}$

10. Fim.

As duas técnicas apresentadas para calcular inversas aproximadas de matrizes simétricas podem ser generalizadas para tratar casos não simétricos. No caso de pré-condicionador AINV a inversa aproximada fatorizada é obtida como

$$M = Z D^{-1} W^T \approx A^{-1}.$$

O algoritmo então calcula duas seqüências de vetores bi-ortogonais $\{z_i\}_{\{i=1,\ldots,n\}}$,

$\{w_i\}_{\{i=1,\ldots,n\}}$ e pode ser obtido do algoritmo 5.8 acrescentando,

$$q_j^{(i-1)} = p_j^{(i-1)} \text{ na linha 3,}$$

e

$$w_j^{(i)} = w_j^{(i-1)} - (\frac{q_j^{(i-1)}}{q_i^{(i-1)}})w_i^{(i-1)} \text{ na linha 9.}$$

Os vetores w_i gerados correspondem as colunas da matriz W e, assim como os vetores z_i devem ser submetidos a uma regra de eliminação para preservar a esparsidade da matriz.

No caso do algoritmo do fator inverso para matrizes não simétricas, são obtidos dois fatores triangulares L e U, inferior e superior respectivamente, tais que

$$LAU \approx D,$$

onde D é uma matriz diagonal.

Como no caso simétrico, supõe-se uma seqüência de matrizes,

$$A_{k+1} = \begin{pmatrix} A_k & v_k \\ w_k & \alpha_{k+1} \end{pmatrix},$$

na qual $A_n \equiv A$, e que os fatores L_k, U_k estão disponíveis para matriz A_k. Então, escreve-se

$$L_{k+1} = \begin{pmatrix} L_k & 0 \\ -y_k & 1 \end{pmatrix},$$

$$U_{k+1} = \begin{pmatrix} U_k & -z_k \\ O & 1 \end{pmatrix},$$

$$D_{k+1} = \begin{pmatrix} D_k & 0 \\ 0 & \delta_{k+1} \end{pmatrix},$$

onde, em função da relação $L_{k+1}A_{k+1}U_{k+1} = D_{k+1}$, z_k, y_k e δ_{k+1} são tais que

$$A_k z_k = v_k \tag{5.28}$$

$$y_k A_k = w_k \tag{5.29}$$

$$\delta_{k+1} = \alpha_{k+1} - w_k z_k - y_k v_k + y_k A z_k \tag{5.30}$$

Diferentemente do caso simétrico, agora é necessário resolver dois sistemas lineares por iteração, um deles envolvendo a matriz A^T. Como resultado, o método BiCG é indicado para este caso.

Capítulo 6

Aspectos da implementação computacional

6.1 Técnicas de armazenamento

A limitação de memória computacional é uma das causas da dificuldade no tratamento de sistemas lineares esparsos por meio de métodos de solução direta, tais como eliminação Gaussiana. Isso acontece porque, como dito anteriormente, as matrizes que surgem de problemas reais são freqüentemente muito grandes.

Assim sendo, os métodos iterativos se consolidaram como uma boa alternativa para solucionar esses sistemas lineares.

No entanto, é importante destacar que a utilização de métodos iterativos também não seria possível se as matrizes, originalmente esparsas, fossem tratadas como matrizes densas, no que diz respeito às formas de armazenamento. Isso parece claro, pois se fosse possível armazenar todos os elementos da matriz, e manipulá-la dessa forma, não faria nenhuma diferença se o elemento é nulo ou não.

Assim, é primordial que as estratégias de armazenamento utilizadas levem em conta a natureza esparsa da matriz, e se aproveitem dessa esparsidade

para economizar memória computacional. Nesse sentido, vários formatos para armazenar matrizes esparsas foram desenvolvidos e estão disponíveis na literatura especializada [1].

Merecem destaque as técnicas de armazenamento comprimidas por linha CSR (*"Compressed Sparse Row"*) e comprimidas por coluna CSC (*"Compressed Sparse Column"*), ambas utilizadas neste trabalho.

6.1.1 Formatos CSR e CSC

A idéia básica dos esquemas de armazenamento de matrizes esparsas é armazenar somente os elementos não nulos da matriz, mantendo alguma informação adicional com relação à posição dos elementos. A maneira mais simples e óbvia de manter essas informações adicionais é guardar as coordenadas de cada elemento não nulo armazenado. Neste esquema, chamado formato de coordenadas, a matriz

$$A = \begin{pmatrix} 1 & 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 4 & 0 \\ 0 & 5 & 6 & 0 & 7 \\ 0 & 0 & 8 & 9 & 0 \\ 10 & 0 & 0 & 11 & 12 \end{pmatrix}$$

é representada por

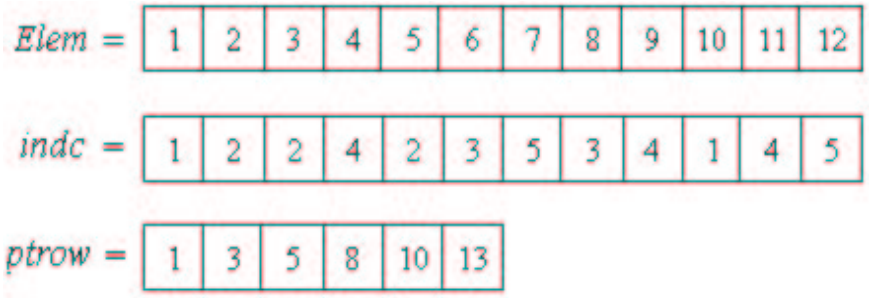
Elem =	2	4	6	1	7	3	9	11	10	5	12	8
indr =	1	2	3	1	3	2	4	5	5	3	5	4
indc =	2	4	3	1	5	2	4	4	1	2	5	3

Observe-se que o armazenamento de matrizes esparsas no formato de coordenada é feito independentemente da ordem dos elementos. Na verdade, se

os elementos foram armazenados linha por linha ou coluna por coluna, um dos vetores que armazenam as posições dos elementos conterá informações redundantes.

No formato CSR, os elementos são armazenados linha por linha e o vetor *indr* é substituído por um vetor de ponteiros chamado *ptrow*, que aponta para o início de cada linha no vetor *Elem*. Na nova estrutura de dados, o vetor de ponteiros terá um comprimento $n+1$ onde n é a dimensão da matriz. Nesse vetor, as n primeiras posições armazenarão as posições nos vetores *Elem* e *indc* onde começam as n linhas da matriz. A posição $n+1$ conterá o número $1+nz$ onde nz é o número de elementos não zeros da matriz, isto é, conterá o endereço em *Elem* e *indc* do início de uma linha $n+1$ fictícia.

Usando esse formato, a estrutura para a matriz A do exemplo acima torna-se

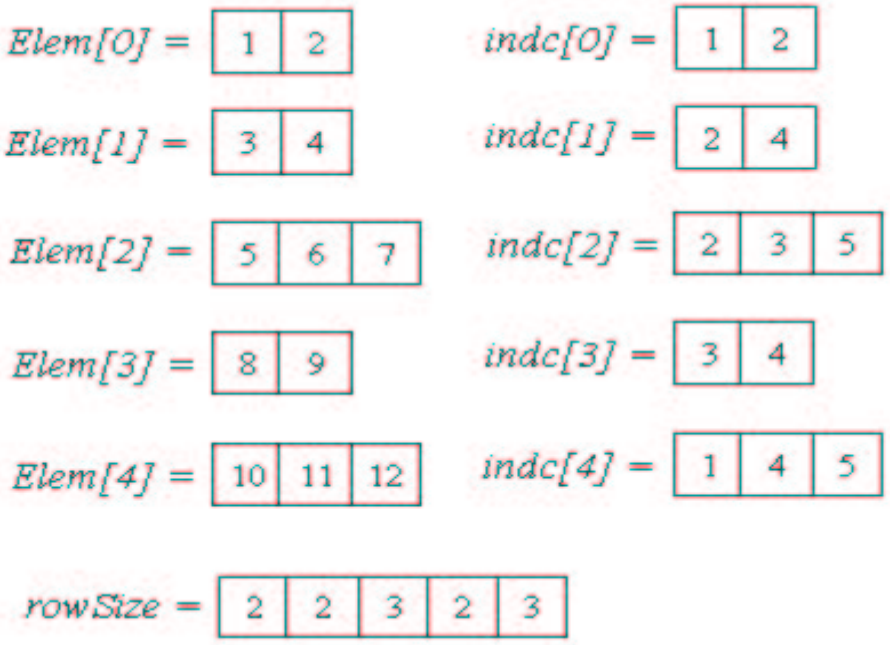


Este esquema de armazenamento é muito útil e, provavelmente é o mais utilizado. Porém, em alguns casos pode apresentar algumas dificuldades. Observe que para atualizar uma linha da matriz, operação muito comum da construção dos pré-condicionadores ILU descritos no capítulo 5, é preciso trabalhar com um vetor que contém todos os elementos não nulos da matriz. Diante disso, foi feita uma reformulação nesse esquema de armazenamento de forma que os elementos não nulos de cada linha fossem armazenados em vetores separados.

Dessa forma, os vetores *Elem* e *indc* que eram de comprimento nz são substituídos por n novos vetores tais que os vetores $Elem[i-1]$ e $indc[i-1]$ tenham comprimentos iguais ao número de elementos não nulos da linha i da

matriz. Um outro vetor, chamado $rowSize$, armazenará o número de elementos não nulos de cada linha da matriz A.

Assim, voltando ao exemplo anterior tem-se



Este esquema foi utilizado neste trabalho para os formatos CSR e CSC. No formato CSC a única diferença é que os elementos são armazenados por coluna e os índices da linha de cada elemento não nulo são armazenados. Existem ainda vários outros formatos que na prática são muito pouco utilizados [23].

6.2 Implementação computacional

No que diz respeito à implementação computacional vale destacar que a possibilidade de manipular diferentes estruturas de dados para matrizes esparsas, vetores e pré-condicionadores é importante para poder tratar de forma eficiente os vários problemas existentes. Em alguns casos, é preciso armazenar a matriz de um pré-condicionador no formato CSR, como por exemplo nos pré-condicionadores implícitos sem pivoteamento como ILUT, e em outros, pode ser conveniente utilizar o formato CSC, como no caso do pré-condicionador

explícito calculado por coluna. Em todos os casos um mesmo método iterativo deve estar preparado para lidar com esta variação.

Além disso, é desejável que para o usuário do programa computacional essas alterações passem despercebidas, no sentido de que ele não tenha que se adaptar a elas, e possa manusear o programa de uma forma padronizada, independente da estrutura usada internamente para armazenar e manipular os dados numéricos. Para garantir essa flexibilidade ao código, adotou-se o paradigma da programação orientada a objeto, e para sua implementação a linguagem C++ [30].

Assumindo, portanto, um estilo de programação orientada a objeto, seguiu-se a implementação de todas as classes necessárias para formar um código eficiente e flexível. Além das classes para matrizes esparsas nos dois formatos apresentados, foi necessária a criação de uma classe para matrizes densas, chamada `Dmatrix`, também dotada de todas as operações necessárias a uma eficiente interface com outros objetos. Também foram construídas uma classe para vetores, classes para todos os métodos iterativos apresentados nos capítulos anteriores, e uma classe para pré-condicionadores da qual derivam todos os pré-condicionadores, tanto os implícitos quanto os explícitos, tratados neste trabalho.

6.2.1 Classes de matrizes esparsas

Seguindo as estruturas de armazenamento descritas na subseção 6.1.1, foram implementadas funções que realizam todas as operações que envolve uma matriz esparsa e um outro objeto, ou um escalar, necessárias na solução de um sistema linear com os métodos apresentados. Dentre essas funções estão aquelas que realizam operações básicas, como `Mult(x)` que realiza o produto de uma matriz esparsa por um vetor x e retorna um vetor resultado y, até funções pouco utilizadas como as que realizam operações BLAS nível 3 [11]. As principais funções das classes `CSRMatrix` e `CSCMatrix` estão descritas na tabela

6.1.

Função	Operação realizada
MultAdd(x,y)	$y = y + Ax$
Mult(x,y)	$y = Ax$
TmultAdd(x,y)	$y = A^T x$
SetVal(i, j, α)	$A(i, j) = \alpha$
AddVal(i, j, α)	$A(i, j) = A(i, j) + \alpha$
Transpor(A)	$B = A^T$
Traco()	$\alpha = \sum_i A(i, i)$
Frobenius_norma()	$\alpha = (\sum_{ij} A(i,j)^2)^{1/2} = \|A\|_F$

Tabela 6.1: *Algumas funções das classes de matrizes esparsas.*

Para aproximar o uso de algumas das funções definidas na tabela 6.1 da linguagem matemática usual, foram realizadas sobrecargas de alguns operadores como $+, -, *, +=, *=$, tanto para o tratamento entre matrizes esparsas em um mesmo formato, quanto para o tratamento entre matrizes esparsas e vetores, e matrizes esparsas e escalares. Assim a função `Mult(x)`, por exemplo, pode ser substituída por $y = A * x$.

6.2.2 Classe de vetores

As operações de manipulação vetorial também foram definidas em uma classe própria, chamada `Vector`. Estas operações são as que envolvem unicamente vetores, ou vetores e escalares como, por exemplo, cálculo da norma 2 do vetor, `norma2( )`; o cálculo do produto interno entre dois vetores, `dot(x,y)`; e a multiplicação de um vetor por uma constante. Também aqui, alguns operadores forma sobrecarregados visando facilitar a utilização do código. As funções e as sobrecargas de operadores definidas são apresentadas na Tabela 6.2.

Função	Operação	Operadores
norma2()	$\|x\|_2$	-
normmax()	$\|x\|_\infty$	-
dot(x, y)	$x^T y$	-
-	$x = x + y$	x += y
-	$x = x - y$	x -= y
-	$x = xy$	x *= y
-	$x = yz$	x = y * z
-	$x = \alpha y$	x = α * y
-	$x = y + z$	x = y + z
-	$x = y - z$	x = y – z

Tabela 6.2: *Algumas funções da classe* `Vector`

6.2.3 Classes dos pré-condicionadores

Cada um dos pré-condicionadores apresentados neste trabalho é definido em uma classe própria, porém, todos eles são derivados de uma classe base chamada `Precondicionador`. Além da função que constrói o pré-condicionador, todas as classes possuem apenas duas outras funções, `Apply(x)` e `TApply(x)` que aplicam o pré-condicionador a um vetor. Na prática essas duas funções são chamadas de dentro do código do método iterativo pré-condicionado. Sempre que no código do método iterativo sem pré-condicionamento existir um produto da matriz A por um vetor x, a versão pré-condicionada do método deve primeiro aplicar o pré-condicionador e depois aplicar a matriz A ao vetor resultante.* Neste caso, a operação $y = A * x$ torna-se

$$\begin{aligned} z &= P.Apply(x) \\ y &= A * z, \end{aligned}$$

*Tem-se nesse caso um pré-condicionamento pela esquerda $AM^{-1}x = M^{-1}b$. Se o pré-condicionador for aplicado após a matriz A tem-se um pré-condicionamento pela direita.

onde P é o pré-condicionador e x, y, z são vetores.

Todas as classes de pré-condicionadores possuem uma matriz esparsa M como um membro privado, onde o pré-condicionador em si é armazenado. Dessa forma, as funções `Apply(x)` e `TApply(x)` são, na verdade, operações envolvendo matriz esparsa e vetor. No caso dos pré-condicionadores implícitos, essas operações correspondem à solução de sistemas lineares triangulares envolvendo os fatores L e U que são armazenados juntos na matriz M. Nesse caso, o código das funções `Apply(x)` e `TApply(x)` é definido dentro da própria classe do pré-condicionador. Por outro lado, nos pré-condicionadores explícitos, essas funções correspondem aos produtos da matriz esparsa M e M^T, respectivamente, pelo vetor x, e usam as funções correspondentes definidas nas classes de matrizes esparsas.

No que diz respeito à função construtora, cada classe de pré-condicionador difere uma da outra apenas quanto aos argumentos passados pelo usuário.

6.2.4 Classes dos métodos iterativos

As classes dos métodos iterativos é relativamente simples. Todas elas possuem uma única função que executa todo o procedimento do método. De um modo geral esta função recebe como argumentos a matriz original do sistema A, um vetor x com a estimativa inicial que também armazena a solução aproximada ao final do processo, o vetor do lado direito, b, um pré-condicionador P, o número máximo de iterações permitidas max_iter, e uma tolerância tol para a norma do vetor resíduo. Em resumo, a interface de um método, como, por exemplo, BiCGSTAB, tem a forma:

```
int  BiCGSTAB(A, x, b, P, max_iter, tol);
```

Visando malhorar a performance dos códigos implementados neste trabalho, principalmente na contrução dos pré-condicionadores, os elementos das matrizes esparsas são acessados diretamente na estrutura de dados utilizada

no armazenamento. Consequentemente, cada formato de armazenamento determina uma forma de manipular os elementos da matriz.

Como os métodos implementados estão preparados para receber somente matrizes no formato CSR, sempre que a matriz original passada para o método estiver armazenada no formato CSC é necessário uma conversão para o formato CSR. Essa conversão é feita através de uma função definida nas classes de matrizes esparsas.

6.2.5 Classe de matrizes densas

A última classe implementada neste trabalho é a que define as operações que envolvem matrizes densas. Nesta classe estão definidas várias operações importantes, como, por exemplo, decomposição LU e inversão de uma matriz densa. A estrutura de dados utilizada é composta por dois *"arrays"*. O primeiro de comprimento n^2, que armazena os elementos linha por linha, e o segundo de comprimento $n + 1$ contendo ponteiros para o início de cada linha, onde n é a dimensão da matriz. A última posição do *"array"* de ponteiros indica a posição de uma linha fictícia $n + 1$. Essa estrutura é ilustrada na figura 6.1.

As funções membros desta classe estão apresentadas na tabela 6.3, e sua interface com vetores permite resolver sistemas lineares de pequeno porte através de métodos diretos. No entanto, quando se trata de métodos iterativos como os que são tratados neste trabalho a classe `Dmatrix` é muito pouco utilizada. Para ser mais exato, apenas o método GMRES usa um objeto matriz densa para armazenar a matriz de Hessenberg.

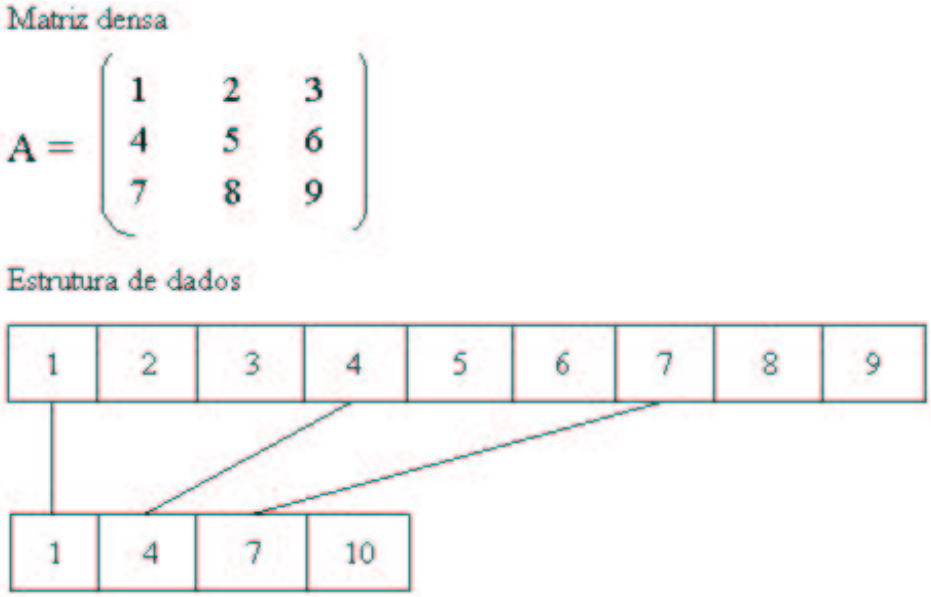


Figura 6.1: *Estrutura de dados usada para classe* `DMatrix`.

Função	Operação	Operadores
mult(x, y)	y = D y	y = D * x
Transpor()	$D = D^T$	-
Invert(A)	$B = D^{-1}$	-
LUdecomp(A)	D = LU	-
LUdecomp( )	D = LU	-
SolveLUx(A, x)	x = LUx	-
-	$D = D/\alpha$	D /= α
-	$D = \alpha D$	D *= α
-	C = D + E	C = D + E
-	C = D - E	C = D - E
-	C = D E	C = D * E
-	$C = \alpha D$	C = D * α
-	$C = D/\alpha$	$C = D/\alpha$

Tabela 6.3: *Funções da classe* `DMatrix`.

Capítulo 7

Resultados das simulações computacionais

7.1 Validação

Os algoritmos apresentados neste trabalho foram implementados e testados com matrizes extraídas da coleção "Matrix Market"[21]. A idéia principal desses testes é comprovar a validade da implementação dos métodos. As matrizes utilizadas nesses testes iniciais são bem conhecidas, e resultados semelhantes são encontrados facilmente na literatura especializada ([10],[8],[3]).

As matrizes utilizadas estão listadas na tabela 7.1.

Foi utilizada uma tolerância de 10^{-7} para a o resíduo produzido pela solução aproximada, isto é, os cálculos foram interrompidos sempre que a norma 2 do vetor resíduo era reduzida a um fator da ordem de 10^{-7}. Foi utilizado 500 como o número máximo de iterações permitidas, e o vetor do lado direito foi escolhido de modo que a solução fosse um vetor com todas as coordenadas iguais a 1. Além disso, a estimativa inicial utilizada foi o vetor nulo.

O método GMRES foi sempre utilizado com reinício a cada 20 iterações.

Matriz	Ordem	Não zeros
PORES2	1244	9613
PORES3	532	3474
SHERMAN3	5005	20033
GRE1107	1107	5664
GRE216B	216	876
NNC261	261	1500
NNC666	666	4044

Tabela 7.1: *Matrizes testes extraídas da coleção Matrix Market.*

Os resultados obtidos mostraram que para as três primeiras matrizes, PORES2,PORES3 E SHERMAN3, os pré-condicionadores ILU fornecem os melhores resultados. Nesses casos, até mesmo o pré-condicionador ILU(0) resolve sem maiores dificuldades. A tabela 7.2 apresenta os resultados obtidos resolvendo um sistema com a matriz PORES2 combinando os pré-condicionadores ILU(0) e ILU(1) com diversos métodos.

Na tabela, t representa o tempo de construção do pré-condicionador, n^o *iter.* e *tempo iter.* denotam, respectivamente, o número de iterações e o tempo em segundos gasto em cada uma delas.

O tempo total, denotado por *tempo tot.*, representa a soma do tempo de construção do pré-condicionador com o tempo das iterações.

Todos os testes foram realizados em um Pentium III, 1GHz com 256Mb de memória RAM.

Método	ILU(1): t=0.29s, nz=18.953		
iterativo	n^o iter.	tempo iter.	tempo tot.
BiCGSTAB	11	0.0110	0.41
CGS	11	0.0127	0.43
TFBiCG	19	0.0158	0.59
GMRES	18	0.0080	0.43
DQGMRES	20	0.0100	0.49
Método	ILU(0): t=0.23s, nz=9.613		
iterativo	n^o iter.	tempo iter.	tempo tot.
BiCGSTAB	23	0.0074	0.40
CGS	26	0.0077	0.43
TFBiCG	46	0.0111	0.74
GMRES	51	0.0055	0.51
DQGMRES	41	0.0132	0.77

Tabela 7.2: *Matriz PORES2 com pré-condicionadores ILU(0) e ILU(1).*

Pode-se observar que apesar de consumir um tempo maior para sua construção e em cada iteração dos métodos, o pré-condicionador ILU(1) gasta, neste caso, um tempo menor para resolver totalmente o problema. Isso ocorre em função da grande redução no número de iterações necessárias para a convergência de cada método iterativo.

Com relação à estratégia utilizada para controlar a quantidade de *"fill in"* na matriz do pré-condicionador, foi possível comprovar que para os casos testados a estratégia utilizada pelo pré-condicionador ILUT é mais eficiente se comparada ao ILU(0) e ILU(1). O gráfico 7.1 ilustra essa afirmação para o caso onde a matriz PORES3 foi resolvida com o método GMRES(20). O parâmetro s é a densidade do pré-condicionador e representa a razão entre o número de não zeros da matriz do pré-condicionador e o número de não zeros da matriz original.

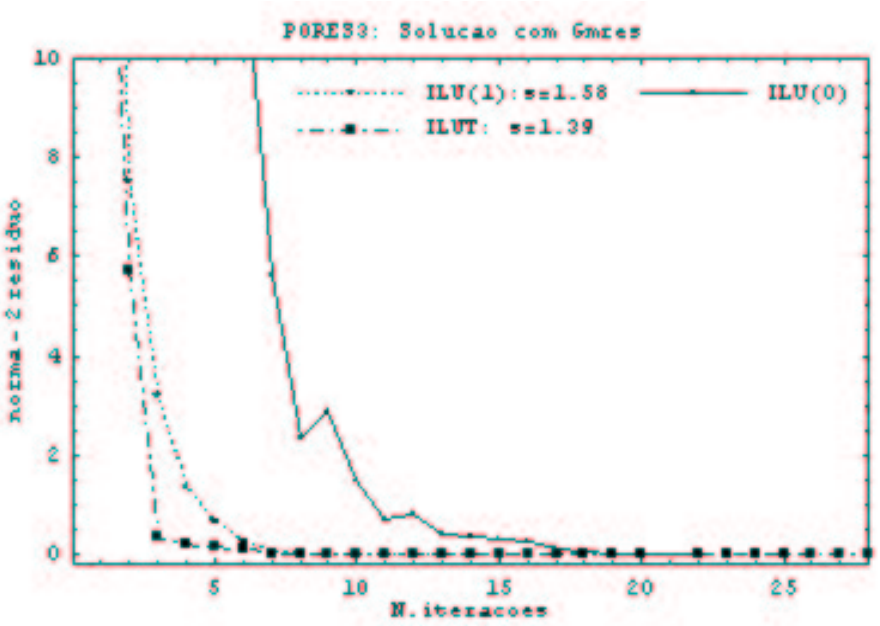


Gráfico 7.1: *Pré-condicionadores ILU com GMRES.*

Mesmo possuindo um número menor de elementos não nulos, o pré-condicionador ILUT fornece melhores resultados para esse caso. Isso também pode ser observado para o sistema envolvendo a matriz SHERMAN2 como mostra o gráfico 7.2.

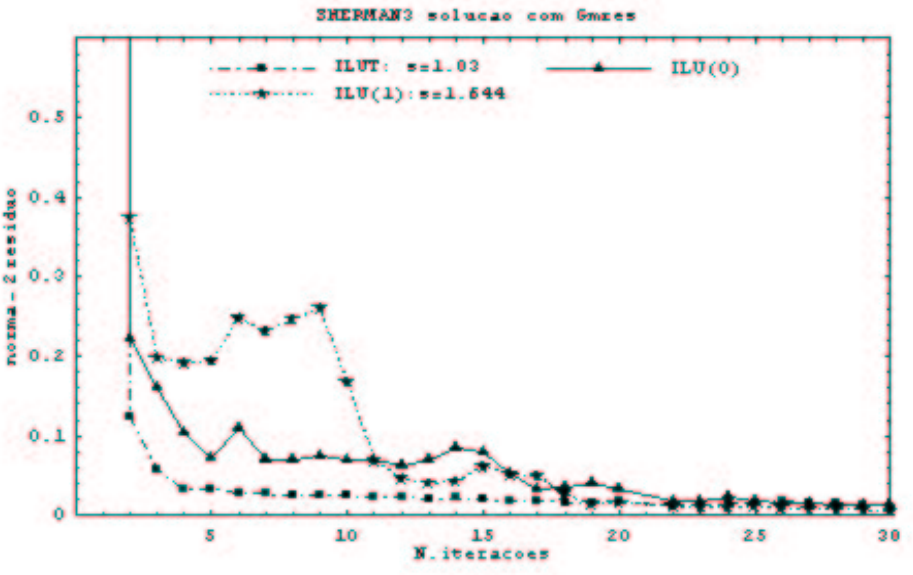


Gráfico 7.2: *Pré-condicionadores ILU com GMRES.*

É importante observar que, em alguns casos, não adianta aumentar excessivamente o número de elementos não nulos no pré-condicionador ILUT. O gráfico 7.3 mostra que isso não garante uma melhora no comportamento de convergência do método iterativo.

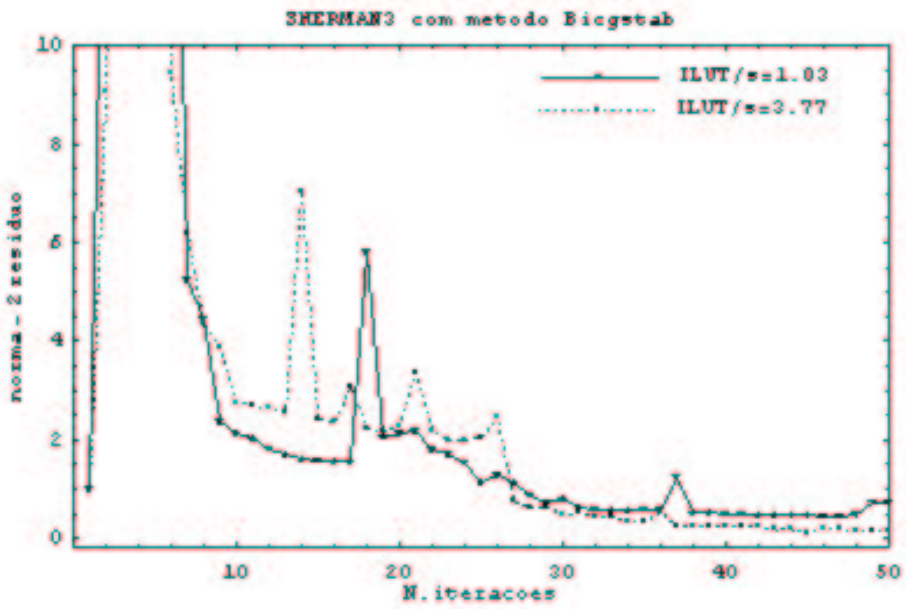


Gráfico 7.3: *ILUT com BiCGSTAB.*

Observe que mesmo aumentando quase quatro vezes o número de não zeros no pré-condicionador, a convergência praticamente não se alterou.

Com relação ao desempenho dos métodos iterativos, o BiCGSTAB e o GMRES(20) produziram os melhores resultados para a maioria dos casos testados. Os gráficos 7.4 e 7.5 mostram os resultados obtidos por alguns métodos associados aos pré-condicionadores ILUT e ILU(1).

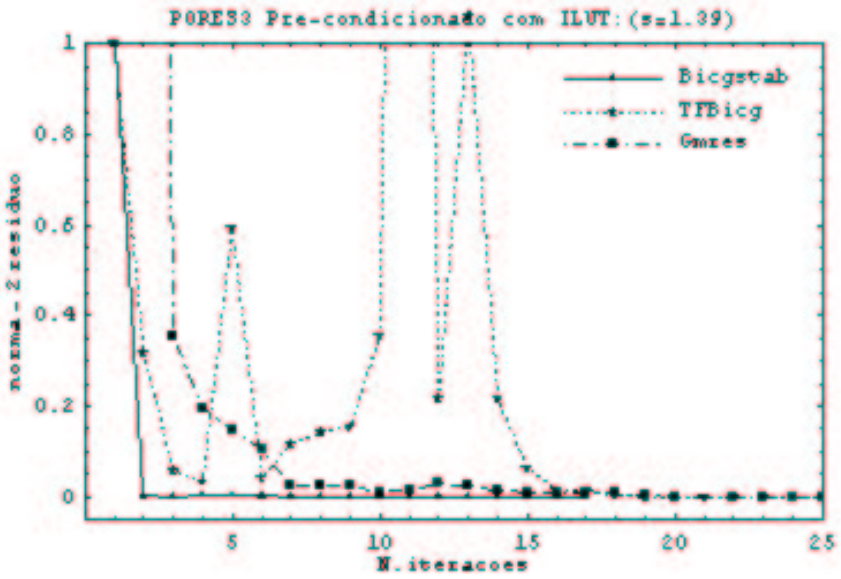


Gráfico 7.4: *Métodos iterativos com ILUT.*

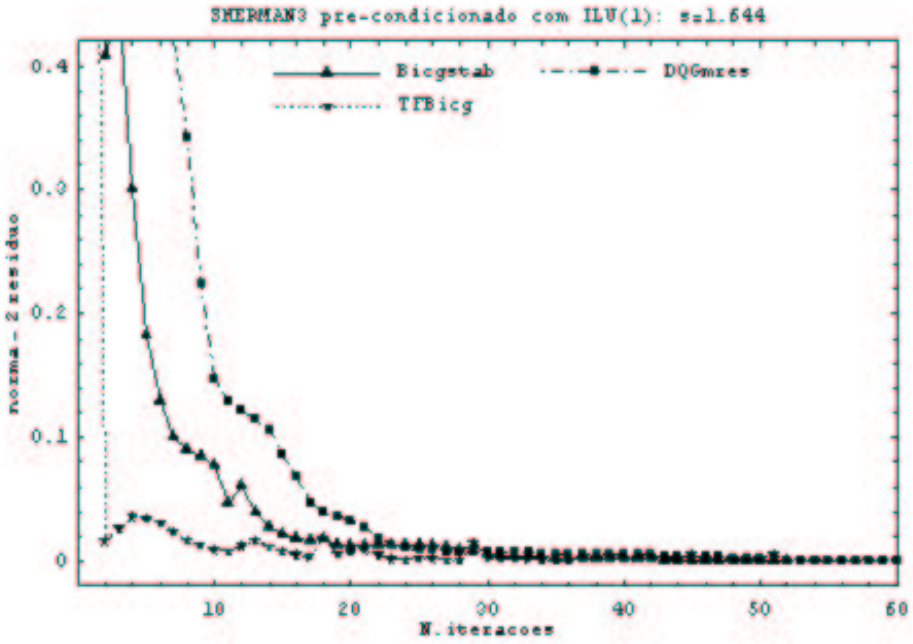


Gráfico 7.5: *Métodos iterativos com ILU(1).*

É interessante observar que para a matriz PORES3 o método TFBiCG associado ao pré-condicionador ILUT apresentou um comportamento de convergência muito oscilatório. No entanto, para a matriz SHERMAN3 esse mesmo método associado ao pré-condicionador ILU(1) apresentou resultados muito bons.

Para essas três primeiras matrizes tratadas aqui, os pré-condicionadores da família ILU tiveram um desempenho muito superior aos pré-condicionadores explícitos. No caso da matriz PORES3 por exemplo, foi preciso uma inversa aproximada demasiadamente densa para aproximar a convergência obtida com o uso do pré-condicionador ILUT. Esses resultados são apresentados no gráfico 7.6, onde a inversa esparsa aproximada (SPAI) foi construída utilizando o algoritmo orientado por coluna apresentado na seção 5.3.

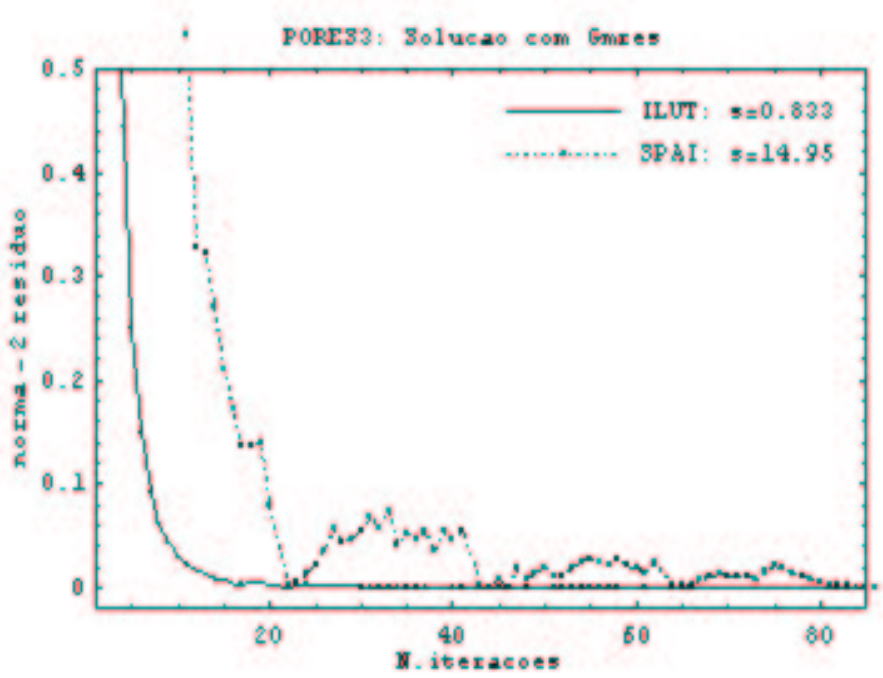


Gráfico 7.6: *ILUT* × *SPAI.*

No entanto, para as demais matrizes apresentadas na tabela 7.1, GRE1107, GRE216B, NNC261, e NNC666, os pré-condicionadores implícitos não obtém

sucesso, nem mesmo utilizando uma estratégia de pivoteamento. Tratam-se de matrizes muito mal condicionadas, todas elas contendo muitos elementos iguais a zero na diagonal principal. Nesses casos, uma alternativa é usar um pré-condicionador que aproxima diretamente a inversa da matriz (SPAI). Neste trabalho isso foi feito usando o algoritmo orientado por coluna.

Apesar de ter resolvido os sistemas lineares envolvendo todas as quatro matrizes, o pré-condicionador SPAI encontrou grandes dificuldades. Em todos os quatro casos foi preciso realizar um escalonamento das colunas da matriz original na tentativa de facilitar a solução do sistema, como feito em [10]. Além disso, a tolerância *tol* para o resíduo precisou ser aumentada para 10^{-15} para melhorar a qualidade da solução, e o número máximo de iterações foi aumentado para 3000. Com relação ao pré-condicionador, foi usado $M = \alpha I$ como estimativa inicial, numa versão com auto pré-condicionamento.

Alguns resultados são apresentados na tabela 7.3. As duas primeiras colunas representam, respectivamente, o número de iterações internas e externas utilizadas na construção do pré-condicionador. O parâmetro s é a densidade do pré-condicionador, e *vdrop* é o valor usado na regra de eliminação. A quarta coluna da tabela indica o número de iterações gastas pelo método GMRES com reinício a cada 20 iterações.

	inner	outer	s	n^0 iter.	tol.	vdrop
NNC666	2	1	0.38	-	10^{-15}	10^{-5}
	2	2	3.45	627	10^{-15}	10^{-5}
	2	3	6.02	658	10^{-15}	10^{-5}
	2	4	8.75	665	10^{-15}	10^{-5}
NNC261	3	1	1.23	501	10^{-15}	10^{-4}
	3	1	0.76	503	10^{-15}	10^{-3}
	4	1	0.56	515	10^{-15}	10^{-2}
	3	2	3.72	525	10^{-15}	10^{-4}
GRE216B	1	1	1.27	434	10^{-15}	10^{-6}
	1	2	1.37	-	10^{-15}	10^{-4}
	2	1	1.08	446	10^{-15}	10^{-4}
	3	1	1.05	457	10^{-15}	10^{-4}
GRE1107	2	1	6.82	2247	10^{-15}	10^{-4}
	2	1	6.82	-	10^{-10}	10^{-2}
	3	1	1.77	2228	10^{-15}	10^{-2}
	3	1	1.77	-	10^{-10}	10^{-2}

Tabela 7.3: *GMRES(20) pré-condicionado com SPAI.*

Na prática, a maior dificuldade foi controlar a quantidade de elementos não nulos na inversa aproximada. Na matriz NNC666 por exemplo, o simples acréscimo do número de iterações externas de 1 para 2 fez a quantidade de elementos não nulos do pré-condicionador aumentar quase dez vezes. Essa dificuldade surge porque o código implementado aplica uma regra numérica baseada apenas no valor do elemento, e na prática não se tem uma estimativa razoável do valor dos elementos da matriz inversa. Uma solução para esse problema seria acrescentar mais um parâmetro que funcionasse como o parâmetro p do pré-condicionador ILUT, de modo a se reter os p maiores elementos de cada coluna.

Observe que para matriz GRE216B os resultados de convergência foram

melhorando à medida que a densidade do pré-condicionador foi aumentando. No entanto, assim como aconteceu no caso do ILUT, um pré-condicionador mais denso nem sempre significou melhores resultados. Para a matriz NNC666 por exemplo, os melhores resultados foram obtidos quando a densidade do pré-condicionador foi menor.

7.2 Aplicação de pré-condicionadores do tipo ILU ao fluxo magnetohidrodinâmico (MHD)

Nesta seção, o desempenho dos pré-condicionadores do tipo ILU é investigado aplicando-os à solução do sistema de equações algébricas obtido quando se revolve o sistema de equações diferenciais parciais que descreve o fluxo magnetohidrodinâmico (MHD) em canais, através do Método de Elementos Finitos.

7.2.1 Descrição do Problema

Seja um canal de altura $2d$ e comprimento total $L_F + L_E + L_R$ apresentado, esquematicamente na figura 7.1.

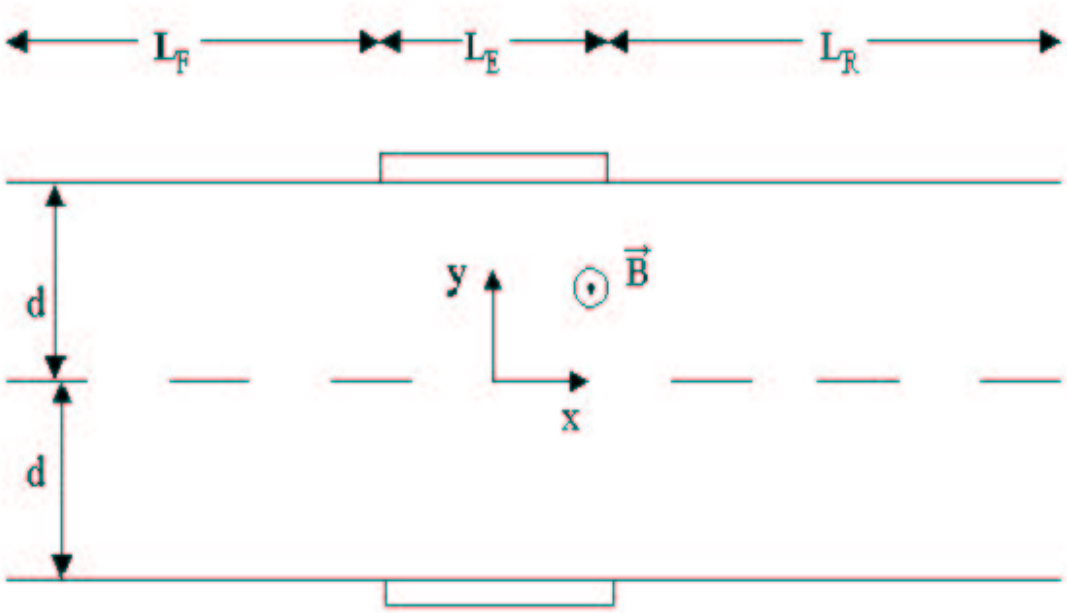


Figura 7.1: *Esquema do canal.*

Nas paredes do canal, o comprimento L_E corresponde a um eletrodo considerado como condutor perfeito e os comprimentos L_F e L_R correspondem a materiais isolantes. Um fluido condutor de eletricidade (metal líquido) movimenta-se na direção x, da esquerda para a direita, sob a ação de um campo magnético externo, $\overrightarrow{B}$, perpendicular à direção do fluxo. Este campo magnético é modelado através da seguinte expressão matemática:

$$\overrightarrow{B} = B_z(x)\overrightarrow{k}$$

$$B_z(x) = \begin{cases} B_0 e^{x+L_E/2}, & x < -L_E/2 \\ 1, & -L_E/2 \leq x \leq /2 \\ B_0 e^{-(x-L_E/2)}, & x > L_E/2 \end{cases}$$

Assumindo que o número de Reynolds magnético é muito pequeno, as equações que governam o escoamento estacionário são obtidos das equações de Maxwell e de Navier-Stokes [17]:

$$\begin{aligned} \rho(\overrightarrow{u}\cdot\nabla)\overrightarrow{u} &= -\nabla p + \mu\nabla^2\overrightarrow{u} + \overrightarrow{J}\times\overrightarrow{B} \\ \nabla\cdot\overrightarrow{u} &= 0 \\ \overrightarrow{J} &= \sigma(\overrightarrow{E} + u\times\overrightarrow{B}) \\ \nabla\cdot\overrightarrow{J} &= 0 \\ \overrightarrow{E} &= -\nabla\phi, \end{aligned}$$

onde ρ é a densidade, $\overrightarrow{u} = (u, v)$ é o vetor velocidade cujas componentes axial e transversal são u e v, respectivamente, p é a pressão, μ é a viscosidade dinâmica, $\overrightarrow{J}$ é a densidade de corrente, $\overrightarrow{B}$ é o campo magnético aplicado, σ é a condutividade elétrica do fluido, $\overrightarrow{E}$ é o campo elétrico e ϕ é o potencial elétrico.

As equações acima estão sujeitas às seguintes condições de contorno:

$$\begin{aligned} u(-\infty, y) &= 1.5\ u_0[1 - (\frac{y}{d})^2] \\ \frac{\partial u}{\partial x}(\infty, y) &= 0 \end{aligned}$$

$$
\begin{aligned}
v(-\infty, y) &= v(\infty, y) = 0 \\
p(\infty, y) &= 0 \\
\phi(-\infty, y) &= \phi(\infty, y) = 0 \\
u(x, d) &= v(x, d) = 0 \\
u(x, -d) &= v(x, -d) = 0 \\
\phi(x, d) &= \phi_E \quad |x| \leq L_E/2 \\
\phi(x, -d) &= -\phi_E \quad |x| \leq L_E/2 \\
\frac{\partial \phi}{\partial y}(x, d) &= \frac{\partial \phi}{\partial y}(x, -d) \quad |x| > L_E/2
\end{aligned}
$$

7.2.2 Formulação pelo Método de Elementos Finitos

O problema descrito no item anterior pode ser resolvido numericamente através do Método de Elementos Finitos utilizando-se uma formulação fraca de Galerkin convencional [22].

Contudo, esta abordagem possui dificuldades intrínsecas sérias no tocante à precisão numérica dos resultados e à convergência de solução para situações em que o campo magnético aplicado é intenso. O chamado número de Hartmann

$$M = \frac{B_0 d}{\sqrt{\sigma/\mu}}$$

e o número de Reynolds

$$Re = \frac{\rho u_0 d}{\mu},$$

são amplamente utilizados, respectivamente, para descrever a intensidade das forças de Lorentz e das forças viscosas atuando sobre o fluido.

Para se contornar as dificuldades mencionadas é usual introduzir um procedimento de estabilização na formulação de Galerkin conduzindo ao que é conhecido na literatura como Método de Elementos Finitos Estabilizados.

No caso aqui considerado, emprega-se o procedimento conhecido por SUPG/PSPG (*"Streamline Upwind Petrov-Galerkin/Pressure Stabilizing Petrov-Galerkin"*)[32]. A extensão do método SUPG/PSPG para o fluxo MHD conduz à seguinte formulação pelo Método de Elementos Finitos:

Se $(\overrightarrow{v}, q, w)$ denotam um vetor de funções de teste, encontrar $(\overrightarrow{u}, p, \phi)$ tal que

$$\rho((\overrightarrow{u}\cdot\nabla)\overrightarrow{u}+\nabla p,\overrightarrow{v})+\mu(\nabla\overrightarrow{u},\nabla\overrightarrow{v})+$$
$$\sum_{T\in\Im}\{((\overrightarrow{u}\cdot\nabla)\overrightarrow{u}+\nabla p,\tau(\overrightarrow{u}\cdot\nabla)\overrightarrow{v})\}_T-\sigma(\nabla\phi\times\overrightarrow{B}+(\overrightarrow{u}\times\overrightarrow{B})\times\overrightarrow{B},\overrightarrow{v})-$$
$$\sum_{T\in\Im}\{\sigma(\nabla\phi\times\overrightarrow{B}+(\overrightarrow{u}\times\overrightarrow{B})\times\overrightarrow{B},\tau(\overrightarrow{u}\cdot\nabla)\overrightarrow{v})\}_T=0, \tag{7.1}$$

$$-(\overrightarrow{u},\nabla q) \quad + \quad \sum_{T\in\Im}\{(\overrightarrow{u}\cdot\nabla)\overrightarrow{u}+\nabla p,\tau\nabla q\}_T-$$
$$\sum_{T\in\Im}\{\sigma(\nabla\phi\times\overrightarrow{B}+(\overrightarrow{u}\times\overrightarrow{B})\overrightarrow{B},\tau\nabla q\}_T=0, \tag{7.2}$$

$$(\nabla\phi,\nabla\omega)+(\nabla\omega,\overrightarrow{u}\times\overrightarrow{B})=0, \tag{7.3}$$

Nas expressões acima, (u, v), denota o produto escalar L_2, isto é,

$$(u,v)=\int_\Omega u\cdot v dx,$$

e $(u, v)_T$ é a restrição do produto escalar sobre um elemento T pertencente a uma triangulação $\Im$ do domínio. O parâmetro τ é um parâmetro ajustável conhecido como tempo intrínseco.

Como se trata de um problema não linear aplica-se o método de Newton-Raphson que conduz ao sistema de equações algébricas, resolvido através de métodos iterativos associados a pré-condicionadores ILU.

7.2.3 Resultados numéricos

Foram feitas simulações usando os métodos iterativos BiCGSTAB e GMRES associados aos pré-condicionadores da família ILU. Porém, quando foi utilizado o pré-condicionador ILU(0) o método de Newton-Raphson não convergiu. Para os pré-condicionadores ILUT, com parâmetros $p = 150$ e $\tau = 0.001$, o método convergiu em duas iterações com o BiCGSTAB, e em três iterações com o

GMRES. Quando foi usado ILU(1) a solução foi obtida em duas iterações de Newton-Raphson para os dois métodos iterativos utilizados.

O gráfico 7.7 mostra os resultados obtidos na primeira iteração de Newton-Raphson, onde a matriz original é de ordem 13.725 e possui 1.518.211 elementos não nulos. A densidade s do pré-condicionador vale $s = 0.88$ para o ILUT e $s = 1.78$ para o ILU(1).

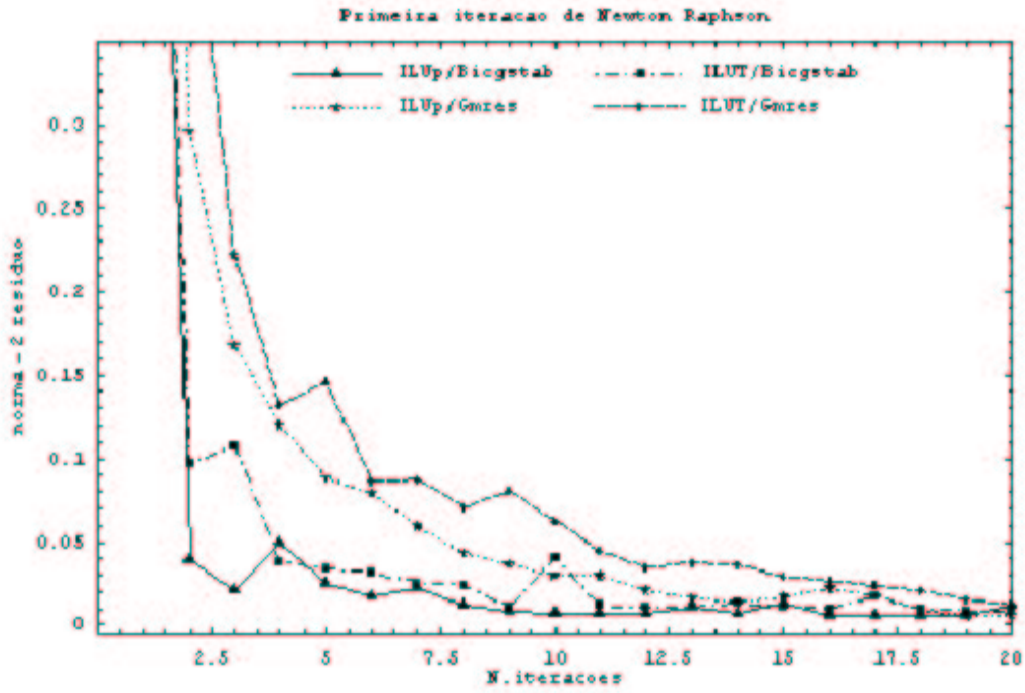


Gráfico 7.7: *Primeira iteração de Newton Raphson.*

Observe que mesmo possuindo um número bem menor de elementos não nulos que o ILU(1), o pré-condicionador ILUT também convergiu em apenas duas iterações de Newton-Raphson quando associado ao BiCGSTAB. Além disso, pode ser observado no gráfico anterior que o comportamento de convergência na primeira iteração é muito parecido para os dois pré-condicionadores.

Com relação ao método iterativo, note que nas primeiras iterações o método BiCGSTAB leva uma pequena vantagem sobre o GMRES. No entanto, como mostram os gráficos 7.8 e 7.9, a partir da iteração 20 o BiCGSTAB passa a apresentar um comportamento de convergência irregular enquanto o GMRES converge de modo mais suave.

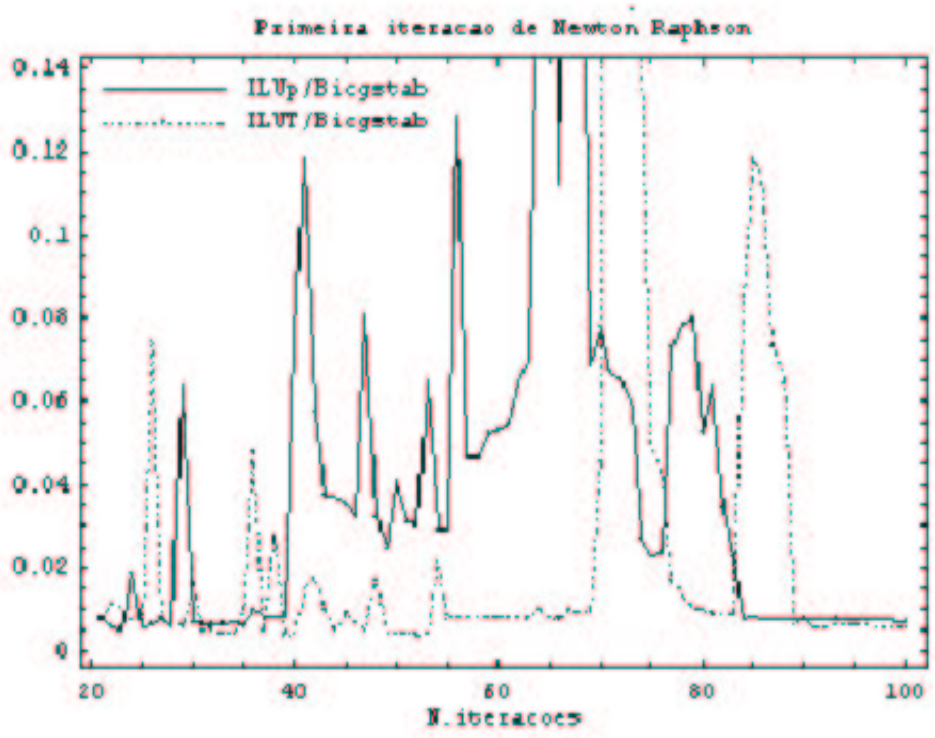


Gráfico 7.8: *Comportamento de convergência do BiCGSTAB.*

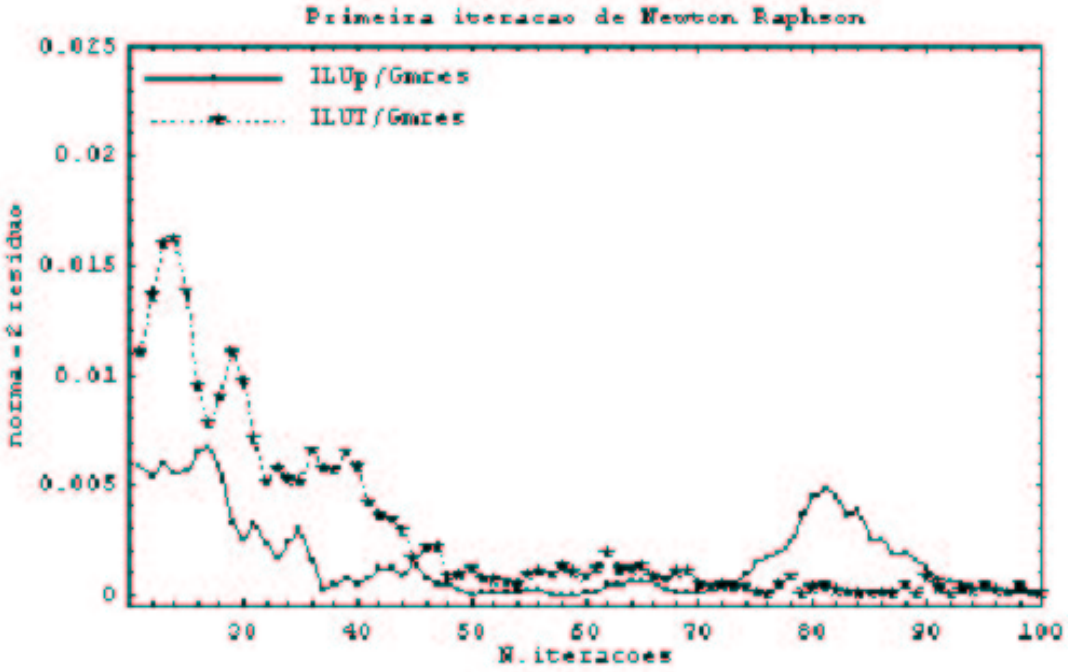


Gráfico 7.9: *Comportamento de convergência do GMRES.*

O gráfico 7.10 mostra os resultados obtidos na segunda iteração de Newton-Raphson. Neste caso, a matriz original possui 1.524.600 não zeros, porém os valores de densidade permanecem os mesmos para os dois pré-condicionadores. Novamente pode ser observado um comportamento de convergência ligeiramente irregular por parte do BiCGSTAB, que é mais acentuado quando esse método está associado ao pré-condicionador ILU(1). Para o método GMRES o gráfico mostra uma convergência suave e muito semelhante para os dois pré-condicionadores.

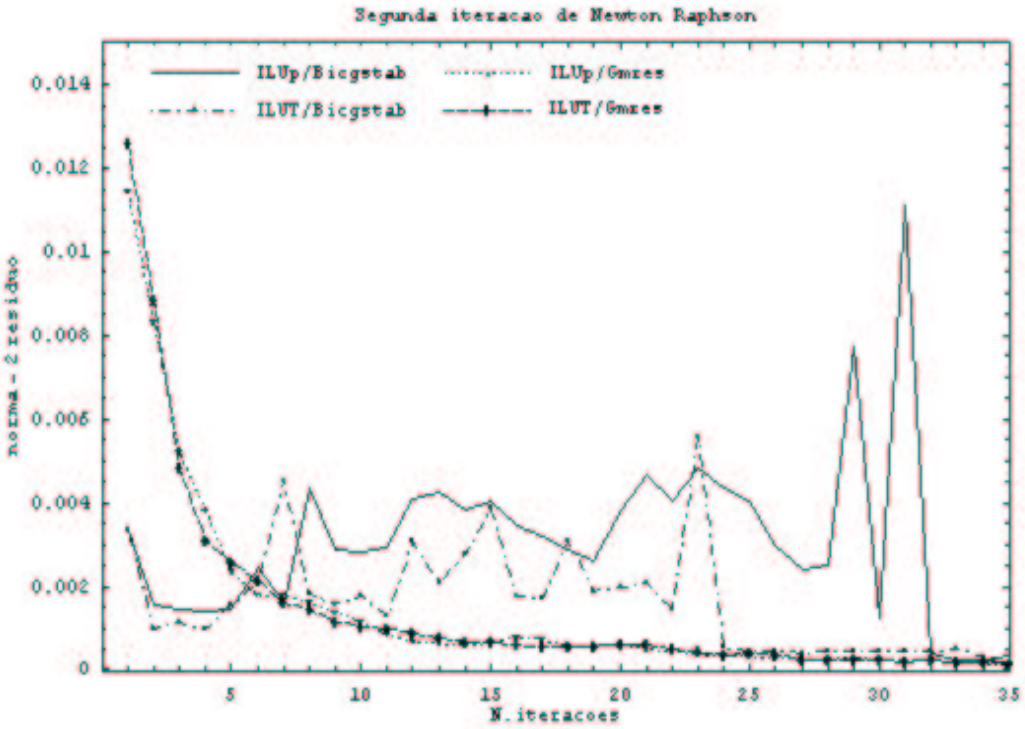


Gráfico 7.10: *Segunda iteração de Newton Raphson*

Capítulo 8

Conclusões

Nos casos onde as condições da matriz permitem a construção dos pré-condicionadores ILU, esses métodos têm se mostrado muito eficientes superando em muito o desempenho dos pré-condicionadores explícitos. Apesar do pré-condicionador ILU(0) não ser suficiente em alguns casos, como no problema do fluxo magnetohidrodinâmico apresentado, pode-se admitir algum *fill-in* e resolver eficientemente o problema usando o ILU(p) ou o ILUT, com um pequeno favorecimento para esta segunda técnica.

Para os casos onde não é possível construir os pré-condicionadores ILU, as técnicas de aproximação da inversa da matriz representam uma boa alternativa para pré-condicionar o sistema. No entanto, cada caso deve ser trabalhado cuidadosamente, no sentido de superar as dificuldades apresentadas por estes métodos, principalmente no que diz respeito à escolha do padrão de esparsidade ideal para a matriz inversa aproximada.

Com relação aos métodos iterativos, foi possível comprovar a maior robustez do BiCGSTAB e do GMRES em relação aos demais métodos tratados. Neste aspecto, o método GMRES apresenta uma pequena superioridade apesar de apresentar um maior custo computacional.

Feito esse estudo inicial, a intenção agora é realizar um estudo mais definitivo a respeito do comportamento desses métodos. Nesse sentido, a idéia

é melhorar a implementação dos pré-condicionadores explícitos, especialmente no que diz respeito à performance computacional, e aplicá-los, juntamente com as outras técnicas, ao problema do fluxo magnetohidrodinâmico em canais. Existe ainda a possibilidade de estudar a paralelização dos códigos de alguns dos métodos tratados neste trabalho.

Referências Bibliográficas

[1] Barrett, B., Berry, M., Chan, T.F., Demmel, J.,Donato, J.,Dongarra, J.,Eijkhout, V.,Pozo, R.,Romine, C.,Van der Vorst, H.A., *Templates for Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM Publications, Philadelphia, PA, (1993).

[2] Benzi, M., Cullum, J.K. e Tuma, M., *Robust approximate Inverse Preconditioning for the Conjugate Gradiente Method*, SIAM J. Sci. Comput., vol. 22, n. 4, pp. 1318-1332 (2000).

[3] Benzi, M., Haws, J.C. e Tuma, M., *Preconditioning Highly Indefinite and Nonsymmetric Matrices,* SIAM J. Sci.Comput., vol. 22, pp. 1333-1353 (2000).

[4] Benzi, M., Meyer, C.D. e Tuma, M., *A sparse approximate inverse preconditioner for the conjugate gradient method,* SIAM J. Sci. Comput., vol. 17, pp. 1135-1149 (1996).

[5] Benzi, M. e Tuma, M., *Approximate Inverse Preconditioning of Iterative Methods for Nonsymmetric Linear Systems*, Research Report n.653, Institute of Computer Science, Czech Academy of Sciences, Prague, Czech Republic (1995).

[6] Benzi, M. e Tuma, M., *Orderings for Factorized Sparse Approximate Inverse Preconditioners,* SIAM J. Sci. Comput., vol. 21, n. 5, pp. 1851-1868 (2000).

[7] Benzi, M. e Tuma, M., *A Sparse Approximate Inverse Preconditioner for Nonsymmetric Linear Systems,* SIAM J. Sci. comput., vol. 19, n. 3, pp. 968-994 (1998).

[8] Benzi, M. e Tuma, M., *Numerical Experiments with two Approximate Inverse Preconditioners,* BIT, vol. 38, pp. 234-241 (1998).

[9] Chan, T.F.,Pillis, L. e Van der Vorst, H.A., *Transpose-Free Formulations of Lanczos-Type Methods for Nonsymmetric Linear Systems,* Numerical algorithms, vol. 17, pp. 51-66 (1998).

[10] Chow, E.T., *Robust Preconditioning for Sparse Linear Systems*, Graduate School of the University of Minnesota, doctoral thesis (1997).

[11] Dongarra, J.J., Du Croz, J., Hammarling, S., Duff, I., A Set of level 3 basis linear algebra subprograms, *ACM Trans. Math. Software*, vol. 16, n. 1, pp. 1-17 (1990).

[12] Dongarra, J.J. e Eijkhout, V., *Numerical Linear Algebra Algorithms and Software,* J. Comput. Appl. Math., vol. 123, pp. 489-514 (2000).

[13] Freund, R.W., Golub,G.H., e Nachtigal, N., *Iterative Solution of Linear Systems*, Acta Numerica 1, pp. 57-100 (1992).

[14] Freund, R., Nachtigal, N., *QMR: A quasi-minimal residual method for non-Hermitian linear systems,* Numer. Math., vol. 60, pp. 315-339 (1991).

[15] Freund, R.W., Gutknecht, M., Nachtigal, N., *An Implementation of the Look-Ahead Lanczos Algorithm for Non-Hermitian Matrices,* SIAM J. Sci. and Stat. Comp., 14, pp. 137-158 (1993).

[16] Golub, G.H.,Vam Loan, C.F.,*Matrix Computations*, the Johns Hopkins University Press, 3^a ed., Baltimore, (1996).

[17] Hughes, W.F e Young, F.J., *The Eletromagnetodynamics of Fluids*, John Wiley and Sons, New York, 1966.

[18] Kolotilina, L.Y., Yeremin, A.Y., *Factorized Sparse Approximate Inverse Preconditioning I: Theory,* SIAM J. Matrix Anal. Applic. 14, 45-58 (1993).

[19] Lanczos, C., *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators,* citado em Saad, Y. e Van der Vorst, H.A., *Iterative Solution of Linear Systems in the 20th Centure,* J. Comput. Appl. Math., vol. 123, pp. 1-33 (2000).

[20] Lin, C.-J. e Moré, J.J., *Incomplete Cholesky Factorizations with Limited Memory,* SIAM J. Sci. Comput. ,vol. 21, n. 1, pp. 24-45 (1999).

[21] National Institute of Standards, *Matrix Market*, http://math.nist.gov/MatrixMarket (1999).

[22] Ramos, J.I e Winowich, N.S, *"Finite Difference and Finite Element Methods for MHD Channel Flows",* Inst. J. Numer. Meth. Fluids 11, pp. 907-934 (1990).

[23] Saad, Y., *Iterative Methods for Sparse Linear Systems*, PWS Publishing, New York (1996).

[24] Saad, Y., *ILUT: A Dual Threshold Incomplete LU Factorization*, Tecnical Report 92 (38), Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, (1984).

[25] Saad, Y. e Schultz, M.H., *GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems,* SIAM J. Sci. Stat. Comput., vol. 7, n. 3, pp. 856-869 (1986).

[26] Saad, Y. e Van der Vorst, H.A., *Iterative Solution of Linear Systems in the 20th Centure,* J. Comput. Appl. Math., vol. 123, pp. 1-33 (2000).

[27] Saad, Y., Wu, K.S., *DQGMRES: A Direct Quasi-Minimal Residual Algorithm based on Incomplete Orthogonalization,* Numer. Lin. Alg. with Appl., vol. 3, n. 4, pp. 329-343 (1996).

[28] Shewchuk, J.R., *An Introduction to the Conjugate Gradiente Method Without the Agonizing Pain*, university Pittsburgh (1994).

[29] Sonneveld, P., *CGS, a Fast Lanczos-Type Solver for Nonsymetric linear Systems,* SIAM J. Sci. Stat. Comput., vol. 10, n. 1, pp. 36-52 (1989).

[30] Stroustrup, B., *The C++ Programing Language*, 3ª ed., Addison Wesley, (1997).

[31] Tang, W.P., *Toward an Effective Sparse Approximate Inverse Preconditioner,* SIAM J. Matrix Anal. Appl., vol. 20, n. 4, pp. 970-986 (1999).

[32] Tezduyar, T.E., Mittal, L.S; Ray, S.E. e Shii, R., *"Incompressible flow computations with stabilized bilinear and linear equal-order interpolation velocity-pressure elements",* Comput. Methods Appl. Mech. Engrg. 95, pp. 221-242 (1992).

[33] Van der Vorst, H.A., *BiCGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems,* SIAM J. Sci. Stat. Comput., vol. 13, n. 2, pp. 631-644 (1992).

[34] Van der Vorst, H.A., *Iterative Methods for Large Linear Systems*, Utrecht University, Utrecht (2001).

[35] Van der Vorst, H.A., *The convergence behaviour of preconditioned CG and CG-S in the presence of rouding errors*, Lecture notes in Math., 1457, pp. 126-136 (1990).

[36] Zhang, J., *Preconditioned Krylov Subspace Methods for Solving Nonsymmetric Matrices from CFD Applications,*Comput. Methods Appl. Mech. engrg., vol. 189, pp. 825-840 (2000).

[37] Zhang, S., *GPBi-CG:Generalized Product-Type Methods Based on Bi-CG for Solving Nonsimmetric Linear Systems,* SIAM J. Sci. Comput., vol. 18,n. 2, pp. 537-551 (1997).