UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE MATEMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MATEMÁTICA APLICADA

Um Estudo da Fatoração Incompleta LU e Cholesky como Pré-Condicionadores nos Métodos Iterativos

por

Hélia Valério Thibes

Dissertação submetida como requisito parcial
para a obtenção do grau de

Mestre em Matemática Aplicada

Prof. Dr. Rudnei Dias da Cunha, Ph.D.
Orientador

Profa. Dra. Maria Paula Gonçalves Fachin
Co-orientadora

Porto Alegre, Dezembro de 2002.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Thibes, Hélia Valério

Um Estudo da Fatoração Incompleta LU e Cholesky como Pré-Condicionadores nos Métodos Iterativos / Hélia Valério Thibes.—Porto Alegre: PPGMAp da UFRGS, 2002.

108 p.: il.

Dissertação (mestrado) —Universidade Federal do Rio Grande do Sul, Programa de Pós-Graduação em Matemática Aplicada, Porto Alegre, 2002.
Orientador: da Cunha, Rudnei Dias; Co-orientadora: Fachin, Maria Paula Gonçalves

Dissertação: Matemática Aplicada
Modelo, Dissertação

Um Estudo da Fatoração Incompleta LU e Cholesky como Pré-Condicionadores nos Métodos Iterativos

por

Hélia Valério Thibes

Dissertação submetida ao Programa de Pós-Graduação em Matemática Aplicada do Instituto de Matemática da Universidade Federal do Rio Grande do Sul, como requisito parcial para a obtenção do grau de

Mestre em Matemática Aplicada

Linha de Pesquisa: Algoritmos Numéricos e Algébricos

Orientador: Prof. Dr. Rudnei Dias da Cunha, Ph.D.

Co-orientadora: Profa. Dra. Maria Paula Gonçalves Fachin

Banca examinadora:

Prof. Dr. Philippe Olivier Navaux.
Instituto de Informática/UFRGS

Profa. Dra. Liliane Basso Barichelo.
PPGMAp/IM/UFRGS

Prof. Dr. Vilmar Trevisan.
PPGMAp/IM/UFRGS

Dissertação apresentada e aprovada em
04 de Dezembro de 2002.

Prof. Dr. Vilmar Trevisan
Coordenador

AGRADECIMENTO

Gostaria de agradecer, primeiramente a Deus, pois é a Ele que recorremos nos momentos mais difíceis de nossas vidas, aqueles que nos angustiam e nos fazem refletir sobre o que realmente vale a pena. Aos professores do Programa de Pós-Graduação em Matemática Aplicada pela oportunidade, amizade e experiência que nos transmitiram.

À meus filhos Marcelo e Christiane e a meu esposo Celso, pelo carinho incentivo e apoio, mesmo que por tantas vezes tenha lhes dedicado tão pouca atenção. Ao meu orientador e co-orientadora pelas sugestões, paciência e atenção que sempre me dedicaram, mas principalmente por confiarem e acreditarem em mim, aceitando a responsabilidade de orientar-me.

Em especial aos grandes amigos, Anne, Carlos e Dulcenéia pela amizade, ajuda, carinho e incentivo, que com certeza contribuiu para que eu continuasse acreditando em meu potencial e pudesse chegar até aqui. E por fim, a todos que de uma forma ou de outra, contribuiram para que este momento fosse possível.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Neste trabalho procuramos analisar alguns métodos iterativos e os processos de aceleração na solução de sistemas lineares grandes e esparsos, associando o uso de alguns pré-condicionadores, tais como os métodos de fatoração incompleta. De forma mais específica, nos detivemos no estudo dos métodos de fatoração incompleta LU, ou ILU, e o método de Cholesky incompleto. Para isso procuramos antes definir algumas especificidades sobre esses métodos, tais como, critérios de existência, limitação de memória e alguns problemas encontrados em sua implementação. Alguns autores analisam tais problemas e sugerem algumas técnicas de conserto, ou seja, algumas maneiras de eliminar tais falhas para que os métodos de iteração possam ser utilizados para determinar soluções mais próximas da solução real. Procedemos a uma revisão teórica de alguns dos métodos iterativos, dos pré-condicionadores: Jacobi, fatoração incompleta LU e fatoração incompleta de Cholesky e a sua associação com os métodos iterativos GMRES e Gradiente Conjugado. Utilizando os pré-condicionadores associados aos métodos iterativos citados e fixando alguns parâmetros de parada, aplicamos alguns testes. Os resultados e a análise dos mesmos encontram-se neste trabalho.

ABSTRACT

In this work, we analysed iterative methods and acceleration schemes to solve large sparse linear systems of equations using incomplete factorisations methods as preconditioners. The studied involving incomplete LU (ILU) and incomplete Cholesky factorisation methods. In this sense, we first define some properties of the methods such as existence criteria, memory limitations and problems on the implementation. Some authors have analysed this problems and suggest repair techniques, i.e., some ways to avoid breakdowns and allow the use of iterative methods to obtain the most real solutions. Afterwards, we conduct a theoretical revision, including some iterative methods, Jacobi preconditioner, incomplete LU and incomplete Cholesky factorisation preconditioners, as well as the use of these preconditioners with GMRES and Conjugate Gradient (CG) iterative methods. Results are presented for some problems using a fixed stopping criteria.

1 INTRODUÇÃO

Os métodos iterativos para a solução de grandes e esparsos sistemas lineares, alcançaram muita popularidade em diversas áreas de computação científica. Até recentemente, os métodos diretos de solução eram preferidos aos métodos iterativos em aplicações reais, devido à sua robustez e seu comportamento previsível. Estimativas apontam que a maioria dos problemas de simulação em matemática, se convertem em solução de sistemas de equações. Como exemplo, podemos citar a solução de equações diferenciais pelos métodos de discretização, como diferenças finitas ou elementos finitos. Em geral tais sistemas envolvem um número muito grande de equações, com características de esparsidade que podem auxiliar no processo de solução numérica.

Neste trabalho, procuramos fazer uma análise dos diversos métodos de resolução de sistemas lineares, sua aplicabilidade aos diversos tipos de sistemas, mas especialmente aos que possuem uma certa esparsidade e um número n de equações relativamente grande.

Inicialmente, discorremos sobre alguns métodos que se destinam à solução numérica de sistemas lineares. Tais métodos podem ser classificados em dois grupos.

Métodos Diretos: são os que conduzem à solução exata, após um número finito de passos.

Métodos Iterativos: nestes métodos usamos uma aproximação inicial da solução e a melhoramos quantas vezes sejam necessárias para chegarmos a uma exatidão satisfatória. Gropp, et al, descreve estes métodos em [24, p.14].

A escolha do método depende de cada problema a ser resolvido. Se os métodos diretos têm a vantagem de fornecer a solução exata após um certo número de passos e não dependem de condições de convergência, têm a desvantagem de se

tornarem inviáveis se o sistema for muito grande ou apresentar a característica de mal-condicionamento.

Segundo Cunha [10], o número de operações realizadas nos métodos diretos cresce na ordem de n^3, o que compromete os cálculos devido ao acúmulo de erros.

Para a notação, adotaremos a forma usual da Álgebra Linear: um sistema com n equações e n incógnitas será representado por

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n &= b_2 \\ \vdots &= \vdots \\ a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n &= b_n \end{aligned} \tag{1.1}$$

Os coeficientes das incógnitas formam uma matriz quadrada com n linhas e n colunas, $A = (a_{ij})$:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ . & . & \cdots & . \\ . & . & \cdots & . \\ a_{n1} & a_{22} & \cdots & a_{nn} \end{bmatrix}_{n \times n}$$

Assim, definindo os vetores colunas $x = (x_1\, x_2\, ...\, x_n)^T$, e $b = (b_1\, b_2\, ...\, b_n)^T$, podemos representar o sistema (1.1) na forma matricial

$$Ax = b.$$

Se admitirmos que A é não singular, e portanto inversível, então o sistema possui solução única $x = A^{-1}b$, onde A^{-1} representa a inversa da matriz A.

A solução eficiente de um sistema de equações lineares depende na maioria das vezes da escolha formal do método iterativo. Porém, deve-se levar em consideração para esse bom desempenho, o tipo de arquitetura que será utilizada, e nesse particular as arquiteturas paralelas tem um papel relevante.

No capítulo 2 estaremos analisando alguns métodos básicos para a solução de sistemas lineares, mais especificamente os métodos iterativos. Estes são subdivididos em duas classes: os métodos iterativos estacionários e os não estacionários. Sobre os métodos iterativos estacionários, trataremos especialmente dos métodos de Jacobi, Gauss-Seidel, Sobre-relaxação Sucessiva (SOR) e Sobre-relaxação Sucessiva Simétrica(SSOR)[4][10]. Quanto aos métodos iterativos não estacionários, destacamos o Gradiente Conjugado (GC)[2][10] e o Resíduo Mínimo Generalizado (GMRES)[12][13].

No capítulo 3 analisaremos a utilização dos pré-condicionadores, associados aos métodos iterativos citados, as vantagens de sua aplicação e alguns problemas encontrados em sua implementação.

No capítulo 4 estaremos analisando métodos de pré-condicionamento associados à fatoração incompleta da matriz de coeficientes, com o objetivo de acelerar e facilitar a resolução de um sistema linear, com um número menor de iterações e ganho de tempo computacional. Em particular serão analisadas algumas técnicas de fatoração incompleta, tais como, a fatoração incompleta LU (ILU), que baseia-se na eliminação Gaussiana e a fatoração incompleta de Cholesky (IC), considerado um dos pré-condicionadores mais importantes e comumente usados em métodos iterativos para resolver sistemas lineares grandes e esparsos.

No capítulo 5 apresentaremos os experimentos utilizando os pré-condicionadores obtidos das fatorações incompletas, LUINC e CHOLINC e o pré-condicionador de Jacobi, aplicados aos métodos do Gradiente Conjugado e o RGMRES, ou seja, o GMRES reinicializado. Fazemos uma análise de seu desempenho em relação ao número de iterações e tempo de processamento.

E finalmente, no capítulo 6 apresentamos as conclusões finais sobre o trabalho e sugerimos alguns ítens para trabalhos futuros.

2 MÉTODOS BÁSICOS DE SOLUÇÃO

Os primeiros métodos usados na solução de grandes sistemas lineares baseavam-se na "relaxação de coordenadas". Tais métodos assumem inicialmente uma solução aproximada, e modificam gradativamente as componentes através de aproximações que seguem uma certa ordem até atingir a convergência. Cada uma dessas modificações, chamadas de passos de relaxação, destinam-se a eliminar gradativamente as componentes do vetor resíduo. Essas técnicas raramente são usadas separadamente, e quando combinadas com um método eficiente, geralmente algum pré-condicionador, produzem bons resultados. Sobre estes métodos de pré-condicionamento e áreas de aplicação que são bastante populares, podemos consultar Saad[40].

Inicialmente, analisamos alguns métodos que se destinam à solução numérica de sistemas lineares. Tais métodos podem ser classificados em dois grupos: métodos diretos e métodos iterativos.

Métodos Diretos: Estes métodos são apropriados para resolver sistemas de equações lineares relativamente pequenos (um número de equações relativamente grande pode incorrer em muitos erros de arredondamento), e para os casos onde os métodos iterativos não são muito eficientes, como por exemplo em sistemas densos. São baseados na fatoração LU ou Cholesky, da matriz de coeficientes A.

Para os sistemas maiores, o problema causado por erros de arredondamento pode ser evitado ou diminuído, utilizando outros métodos que garantem, em certos casos, uma melhor exatidão. Estes métodos são chamados métodos iterativos.

Métodos iterativos: Este termo se refere a um variado número de técnicas, que utilizando aproximações sucessivas a cada passo, buscam obter soluções mais precisas para um sistema linear. Os métodos iterativos são os rivais fortes dos métodos diretos, principalmente porque eles são menos propensos aos primeiros dois problemas, preenchimentos e precisão, problemas estes que serão melhor explici-

tados no decorrer do trabalho. Porém, os métodos iterativos têm suas próprias falhas. Entre estas falhas está a dificuldade de predizer o comportamento preciso de convergência dos mesmos em uma gama extensa de problemas. Assim, um determinado método iterativo pode dar resultados excelentes em alguns problemas, dentro de uma determinada classe, mas apresentar uma execução pobre em outros problemas dentro daquela mesma classe.

Segundo Barret et al [4], há essencialmente duas classes de métodos iterativos: os métodos estacionários e os não estacionários. Os primeiros são mais simples de entender e de implementar, mas tipicamente com taxas de convergência lentas. Os não estacionários, de desenvolvimento relativamente recente, são normalmente de análise um pouco mais complicada, porém, geralmente bem mais eficientes que os primeiros.

Os métodos estacionários incluem Jacobi, Gauss-Seidel, Sobre-relaxação Sucessiva (SOR) e Sobre-relaxação Sucessiva Simétrica (SSOR). Os métodos não estacionários incluem aqueles baseados no subespaço de Krylov, como o método do Gradiente Conjugado (GC) e o método do Resíduo Mínimo Generalizado (GMRES).

Como a taxa de convergência de um método iterativo depende, em grande parte, do espectro[1] da matriz de coeficientes, este normalmente envolve uma segunda matriz que vem a ser a transformação da matriz original em outra de espectro mais favorável. Tal matriz é chamada de "pré-condicionador". Sua finalidade é melhorar consideravelmente a convergência do método iterativo, de tal forma que possa superar o custo extra de sua construção, implementação e aplicação. Em alguns casos o método iterativo pode não convergir sem o uso de tal matriz "pré-condicionadora".

Métodos estacionários: Métodos iterativos que podem ser expressos na forma simples

$$x^{(k)} = Bx^{(k-1)} + c \tag{2.1}$$

[1] O conjunto de todos os autovalores de A é chamado de espectro de A e é denotado por $\sigma(A)$.

onde nem B nem c dependem do cálculo da iteração k, são chamados métodos iterativos estacionários. Na seqüência discutiremos os principais métodos iterativos estacionários: o método de Jacobi, o método de Gauss-Seidel, o método de Sobre-Relaxação Sucessiva (SOR) e o método de Sobre-Relaxação Sucessiva Simétrica (SSOR). Em cada caso tentaremos resumir o seu comportamento de convergência, eficiência e quando devem ser usados.

Métodos não estacionários: Os métodos iterativos não estacionários diferem dos estacionários pois sua computação envolve informações que variam a cada passo da iteração. Tipicamente, as constantes que são computadas resultam de produtos internos de vetores, ou outros resíduos que surgem do método iterativo. Alguns métodos serão a priori analisados, tais como, o método do Gradiente Conjugado (GC) e o método do Resíduo Mínimo Generalizado (GMRES).

2.1 Método de eliminação Gaussiana

O método direto mais popular é a eliminação Gaussiana. Sua idéia principal é utilizar as operações básicas[2], que não alteram a solução do sistema. Se no processo de eliminação um dos pivôs for zero, devemos trocar as posições das linhas, de modo que tomemos sempre os elementos não nulos como pivô. A sistemática da eliminação Gaussiana, fica descrita pelo algoritmo 2.1.

[2]Aqui nos referimos a operações elementares sobre linhas da matriz representativa do sistema, de modo a triangularizá-la, para, usando retro-substituições, determinar os valores das incógnitas de modo simples.

Algoritmo 2.1 *Eliminação Gaussiana*

1. *Dada a matriz* $A_{n\,n+1}$ *(coluna* $n+1$ *representa b)*
2. *Para* $k = 1, 2, ...(n-1)$
3. *encontre* $i \geq k$ *tal que* $a_{ik} \neq 0$
4. *se* $a_{ii} = 0$ *para todo* $i \geq k$ *pare. A não inversível*
5. *troque a linha k com a linha i*
6. *Para* $i = k+1, k+2, ..., n$
7. $m = m_{ik} = a_{ik}/a_{kk}$
8. *Para* $j = k+1, k+2, ..., n+1$
9. $a_{ij} = a_{ij} - ma_{kj}$.
10. *Fim*
11. *Fim*
12. *Fim*

Para avaliarmos o custo computacional de um algoritmo numérico, devemos levar em conta o volume de operações necessárias à sua utilização, ou seja, medir seu esforço computacional. No algoritmo acima para cada j do terceiro laço $(j = k+1, ...n+1)$ são realizadas duas operações: uma multiplicação e uma adição. Assim, neste laço são necessárias

$$\sum_{j=k+1}^{n+1} 2 = 2(n-k+1) \tag{2.2}$$

operações. No segundo laço, além das operações contabilizadas acima, para cada i realizamos uma divisão; assim no laço correspondente a i temos

$$\sum_{j=k+1}^{n} [1 + 2(n-k+1)] = [1 + 2(n-k+1)](n-k). \tag{2.3}$$

Finalmente, para obtermos o número total de operações fazemos a soma em k, que corresponde ao laço externo:

$$\sum_{k=1}^{n-1}(n-k) + 2(n-k+1)(n-k),$$

ou

$$\sum_{k=1}^{n-1}(n-k) + 2\sum_{k=1}^{n-1}(n-k+1)(n-k),$$

que é igual a

$$\frac{1}{2}(n(n-1)) + \frac{2}{3}(n^3 - n) = \frac{2}{3}n^3 + \frac{n^2}{2} - \frac{7}{6}n.$$

Nos cálculos anteriores usou-se uma soma de progressão aritmética e

$$\sum_{k=1}^{n-1} k^2 = \frac{(n-1)n(2n-1)}{6}. \tag{2.7}$$

Para contabilizar o número total de operações deste método, temos que somar o número de operações para obter e resolver um sistema triangular. Deste modo para aplicar o método de eliminação em um sistema de n equações e n incógnitas o número total de operações será

$$\frac{2}{3}n^3 + \frac{n^2}{2} - \frac{7n}{6} + \frac{n^2}{2} - \frac{n}{2} = \frac{2}{3}n^3 + n^2 - \frac{5}{3}n, \tag{2.8}$$

que é um número próximo de $\frac{2}{3}n^3$ para n grande.

O peso maior no esforço computacional do método deve-se às operações realizadas na eliminação, pois este processo é responsável pelo termo cúbico no custo computacional. Isso deve ser explorado, quando se tornar necessário resolver mais de um sistema com a mesma matriz de coeficientes.

2.2 Eliminação de Gauss com pivotamento

Na resolução de sistemas lineares, a solução encontrada, pode muitas vezes apenas aproximar a solução real e não encontrá-la exatamente. Isto acontece, pois erros numéricos aparecem no processo de cálculo, devido a arredondamentos e armazenamento das variáveis envolvidas. Este processo introduz o "erro de arredondamento absoluto" (valor real menos o valor encontrado)

$$e = x - x'.$$

(onde x' é o valor aproximado). Uma medida melhor do efeito possível do erro é fornecida pela razão entre o erro de arredondamento absoluto e o valor verdadeiro - o "erro relativo"

$$\frac{x - x'}{x}.$$

Os erros de arredondamento podem se acumular durante a execução de operações aritméticas repetitivas. Em uma computação numérica extensa, que envolva um número elevado de operações de adições e multiplicações, esse erro de arredondamento acumulado pode ser tão grande a ponto de o resultado final se afastar muito do esperado e não ter significado. Além disso, o processo de eliminação Gaussiana é particularmente sensível a erros de arredondamento, mesmo quando apenas uma quantidade modesta de cálculos for envolvida.

O método de eliminação de Gauss com pivotamento é projetado para minimizar a acumulação de erros de arredondamento ao resolver sistemas lineares. A idéia principal do pivotamento é escolher, em cada estágio, o pivô de maior grandeza possível [36].

2.3 Sistemas mal-condicionados

Todos os programas computacionais de eliminação de Gauss projetados com cuidado empregam pivotamento para tentar controlar a acumulação de erros de arredondamento. No entanto, existem alguns sistemas lineares que são extremamente sensíveis a diminutos erros que ocorrem mesmo com as estratégias mais cuidadosas. Estes sistemas são ditos mal-condicionados, e nestes, tanto pequenos erros nos elementos originais de A e b como os pequenos erros de arredondamento em passos computacionais intermediários, podem conduzir a grandes erros nos resultados.

Observamos que neste caso, pequenas mudanças em b (da ordem de erros de arredondamento), podem resultar em mudanças bastante significativas na solução. A questão levantada é: se o vetor constante b no sistema $Ax = b$ for mudado para $b + \Delta b$, o que se pode dizer sobre a mudança resultante Δx no vetor solução x? Observe que se

$$Ax = b \quad e \quad A(x + \Delta x) = b + \Delta b, \tag{2.9}$$

então, subtraindo em (2.9) a primeira equação da segunda, encontramos

$$A\Delta x = \Delta b \tag{2.10}$$

logo

$$\Delta x = A^{-1}\Delta b. \tag{2.11}$$

Portanto, o ponto central é, de quanto é aumentado o módulo de um vetor, quando multiplicado por uma dada matriz. Esta questão é mais fácil de ser respondida quando a matriz A $n \times n$, for simétrica e todos seus autovalores forem não-nulos. Assim, sejam os autovalores, $\lambda_1, \lambda_2, ..., \lambda_n$, ordenados de forma que

$$0 \leq |\lambda_1| \leq |\lambda_2| \leq ... \leq |\lambda_n|, \tag{2.12}$$

e $v_1, v_2, ..., v_n$, os autovetores associados de A. Como

$$|Av_i| = |\lambda_i v_i| = |\lambda_i||v_i| \tag{2.13}$$

para cada i, $1 \leq i \leq n$, segue-se de (2.13)

$$|\lambda_1||v_i| \leq |Av_i| \leq |\lambda_n||v_i| \tag{2.14}$$

para cada autovetor v_i de A. Como todo vetor x do R^n pode ser expresso como uma combinação linear dos autovetores mutuamente ortogonais $v_1, v_2, ..., v_n$, verifica-se as desigualdades para todo x

$$|\lambda_1||x| \leq |Ax| \leq |\lambda_n||x|. \tag{2.15}$$

Ou seja, quando o vetor x é multiplicado pela matriz A, seu módulo $|x|$ é ampliado no mínimo pelo fator $|\lambda_1|$ e no máximo pelo fator $|\lambda_n|$.

Com isto pode-se começar a pensar em responder de quanto o vetor solução x do sistema $Ax = b$ será "perturbado" quando o vetor constante b for "perturbado" por Δb.

Teorema 2.1 *As Mudanças Relativas em b e x.*
Seja a matriz simétrica não-singular A_1 $n \times n$ com seus autovalores arranjados de tal forma que:

$$0 < |\lambda_1| \leq |\lambda_2| \leq ... \leq |\lambda_n|,$$

e o sistema linear $Ax = b$ *com* $b \neq 0$. *Se* b *for substituído por* $b + \Delta b$, *então a mudança relativa* $\frac{|\Delta b|}{|b|}$ *em* b, *e a mudança relativa* $\frac{|\Delta x|}{|x|}$ *no vetor solução são relacionadas como segue:*

$$\frac{|\Delta x|}{|x|} \leq \frac{|\lambda_n|}{|\lambda_1|} \cdot \frac{|\Delta b|}{|b|}. \tag{2.16}$$

Portanto, se for cometido um erro Δb *no valor de* b, *o produto à direita da equação (2.16), fornece um limite superior do erro relativo resultante na solução.*

Prova Como $Ax = b$, a segunda desigualdade em (2.15) fornece

$$|b| \leq |\lambda_n||x|$$

da mesma forma para $(A\Delta x = \Delta b)$, a primeira desigualdade em (2.15) conduz a

$$|\lambda_1||\Delta x| \leq |\Delta b|.$$

Logo, a multiplicação das últimas desigualdades dá

$$|\lambda_1||\Delta x||b| \leq |\lambda_n||x||\Delta b|$$

e finalmente, dividindo por $|\lambda_1||b||x|$, temos a desigualdade desejada em (2.16).
Note que a matriz A entra na desigualdade (2.16), apenas na forma da relação $\frac{|\lambda_n|}{|\lambda_1|}$ da razão das magnitudes de seu maior e menor autovalores. O número

$$c(A) = \frac{|\lambda_n|}{|\lambda_1|} \tag{2.17}$$

é chamado número de condição da matriz simétrica não-singular A que possui os autovalores maior e menor, λ_n e λ_1, respectivamente.
Em termos de $c(A)$, a desigualdade (2.16) toma a forma

$$\frac{|\Delta x|}{|x|} \leq c(A)\frac{|\Delta b|}{|b|}. \tag{2.18}$$

Se o número de condição $c(A)$ for pequeno, então (2.18) implica que um pequeno erro relativo em b resulta em um erro pequeno na solução x. Mas se $c(A)$ for muito grande, então mesmo um erro relativo pequeno em b pode resultar em um erro relativo grande na solução x.

Resumindo, dizemos que o sistema $Ax = b$ é mal-condicionado, se pequenos erros em A e b resultarem em grandes erros na solução; caso contrário, diz-se que o sistema é bem-condicionado (bem-posto). Com isso conclui-se que um sistema é bem-condicionado quando sua solução é "estável" com relação a pequenas perturbações em A e b.

Quando um sistema for mal-condicionado, pouco há a fazer para evitar grandes erros na solução numérica, pois os pequenos erros de arredondamento durante o processo de solução podem ter o mesmo efeito que pequenas mudanças no vetor b. Quando isso se verificar em um sistema, revelado por um número de condição grande, o usual é tentar reformular o problema para evitar um sistema mal-condicionado. Vale ressaltar que o número de condição, definido anteriormente, levava em consideração uma matriz A simétrica e não-singular.

2.4 Métodos iterativos estacionários

Métodos iterativos que aparecem na forma simples, analisada anteriormente,

$$x^{(k+1)} = Mx^{(k)} + c \tag{2.19}$$

(onde M e c não dependem do cálculo da iteração k), são chamados de métodos iterativos estacionários. Analisaremos a seguir de forma mais pormenorizada alguns desses métodos, como: o método de Jacobi, o método de Gauss-Seidel, o método da Sobre-relaxação Sucessiva SOR, e o método da Sobre-relaxação Sucessiva Simétrica SSOR. Serão analisados o comportamento de convergência de alguns desses métodos, sua eficiência e condições de uso.

2.4.1 O método de Jacobi

Segundo Barret [4] este método é extremamente simples, mas nos casos em que a matriz não é diagonalmente dominante[3], deve-se considerá-lo apenas como uma introdução a métodos iterativos ou então como um pré-condicionador em um método não estacionário. Este método se presta muito bem para a paralelização.

O método de Jacobi é facilmente derivado, se analisarmos separadamente cada uma de suas n equações. Se na i-ésima equação de $Ax = b$ dada por

$$a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n = b_i$$

ou

$$\sum_{j=1}^{n} a_{ij}x_j = b_i,$$

nós isolamos o valor de x_i, e assumimos que as outras entradas de x permanecem fixas, obtemos

$$x_i = \left(b_i - \sum_{j \neq i} a_{ij}x_j \right) / a_{ii}. \tag{2.20}$$

Isto leva a um método iterativo definido por

$$x_i^{(k)} = \left(b_i - \sum_{j \neq i} a_{ij}x_j^{(k-1)} \right) / a_{ii}, \tag{2.21}$$

que representa o método de Jacobi. Segundo Barret [4], neste método a ordem na qual as equações são examinadas é irrelevante, pois são tratadas independentemente. Por esta razão este método é conhecido como método de deslocamentos simultâneos, desde que a princípio as atualizações sejam feitas simultaneamente.

Considerando o sistema linear $Ax = b$, com a matriz A decomposta de modo que

$$A = D - E - F,$$

[3]Uma matriz é diagonalmente dominante se satisfaz as condições seguintes $|a_{ii}| \geq \sum_{j=i,j\neq 1}^{n} |a_{ij}|, i = 1, 2, ..., n.$

onde D é uma matriz diagonal e E e F são matrizes triangulares inferior e superior, respectivamente, com diagonais nulas. Assim, o sistema pode ser escrito na forma

$$(D - E - F)x = b \rightarrow Dx = (E + F)x + b.$$

Essa igualdade pode ser convertida em processo iterativo formando a recorrência

$$x^{(k+1)} = (D^{-1}(E + F))x^{(k)} + D^{-1}b \rightarrow x^{(k+1)} = Jx^{(k)} + c, \tag{2.22}$$

tal que, por (2.19), a matriz $J = D^{-1}(E + F)$ é a matriz de iteração do método de Jacobi, segundo Campos[7].

2.4.2 O método de Gauss-Seidel

Considere as equações em (2.20). Se assumirmos que as equações são examinadas separadamente, uma de cada vez, seqüencialmente, e os resultados utilizados tão logo estejam disponíveis, obtemos o método de Gauss-Seidel:

$$x_i^{(k)} = \left(b_i - \sum_{j<i} a_{ij} x_j{}^{(k)} - \sum_{j>i} a_{ij} x_j^{(k-1)} \right) / a_{ii}. \tag{2.23}$$

Sobre o método de Gauss-Seidel dois fatos importantes devem ser analisados. Primeiro, os cálculos em (2.23) parecem estar em série. Pelo fato de que cada componente da nova iteração depende de todas as outras previamente calculadas, as atualizações não podem ser feitas simultaneamente.

Segundo, o $x^{(k)}$ da nova iteração depende da ordem na qual as equações são examinadas. Por essa razão, o método de Gauss-Seidel é chamado de *método de deslocamentos sucessivos*, pois indica a dependência das iterações ao ordenar. Com essas ordenações, não só as componentes da nova iteração são mudadas, mas também a sua ordem.

Segundo Barret et al[4], estes dois pontos são importantes, pois se A é esparsa, a dependência entre cada componente da nova iteração e a componente anterior não é absoluta. A presença de zeros na matriz pode remover a influência

de alguns dos componentes prévios. Usando uma ordenação criteriosa das equações, pode ser possível reduzir tal dependência, restabelecendo assim a habilidade para fazer atualizações a grupos de componentes em paralelo. De qualquer modo, reordenando as equações se pode atingir a taxa na qual o método de Gauss-Seidel converge. Uma escolha inadequada de reordenamento pode diminuir a taxa de convergência; uma boa escolha pode aumentar a taxa de convergência.

Considerando o sistema linear como no método anterior, $Ax = b$, com a matriz A decomposta como $A = D - E - F$, também segundo Campos[7], onde D é matriz diagonal e E e F são matrizes triangulares inferior e superior, respectivamente, com diagonais nulas. O sistema linear pode ser escrito

$$(D - E - F)x = b \rightarrow (D - E)x = Fx + b$$

ou na forma de iteração obtida da recorrência

$$x^{(k+1)} = ((D - E)^{-1}F)x^{(k)} + (D - E)^{-1}b \rightarrow x^{(k+1)} = Sx^{(k)} + d). \qquad (2.24)$$

Por (2.19), a matriz $(D - E)^{-1}F$ é a matriz de iteração do método de Gauss-Seidel.

Algoritmo 2.2 *Método de Gauss-Seidel*

1. Escolha um valor inicial $x^{(0)}$ *para a solução* x.
2. Para $k = 1, 2, ...$
3. *Para* $i = 1, 2, ..., n$
4. $\sigma = 0$
5. *Para* $j = 1, 2, ..., i - 1$
6. $\sigma = \sigma + a_{i,j}x_j^{(k)}$
7. *Fim*
8. *Para* $j = 1 + i, ..., n$
90. $\sigma = \sigma + a_{i,j}x_j^{(k-1)}$
10. *Fim*
11. $x_i^{(k)} = (b_i - \sigma)/a_{i,i}$
12. *Fim*
13. Cheque a convergência; continue se necessário
14. Fim

2.4.3 Convergência dos métodos de Jacobi e Gauss-Seidel

O método de Gauss-Seidel usa atualizações imediatas dos dados e por esse motivo, poderíamos esperar que a convergência fosse mais rápida ao utilizá-lo. Embora com certa freqüência isto ocorra, não se pode generalizar tal fato, pois em certos casos, o método de Jacobi converge e o de Gauss-Seidel diverge. O método de Jacobi, para o momento atual, tem sido mais atrativo, devido ao grande uso de arquiteturas em paralelo para computação.

A análise de convergência dos métodos em questão, pode ser feita colocando-os na forma geral dos métodos iterativos:

$$x^{(k+1)} = Bx^{(k)} + c, \tag{2.25}$$

onde B é a matriz de iteração. As matrizes de iteração de Jacobi e Gauss-Seidel são, respectivamente:

$$B^J = D^{-1}(E+F) \quad e \quad B^{GS} = (D-E)^{-1}F. \tag{2.26}$$

Nos dois métodos iterativos, a convergência é estudada através dos autovalores de B (dizemos que $\lambda_i,\ i = 1:n$ é autovalor da matriz B se $Bv = \lambda v$ para algum $v \neq 0$). O teorema abaixo, conforme Cunha [10], estabelece a condição necessária e suficiente para a convergência.

Teorema 2.2 *O método iterativo $x^{(k+1)} = Bx^{(k)} + c$, converge se e somente se $max_{1\leq i\leq n}|\lambda_i| < 1$.*

No entanto, como a determinação dos autovalores de uma matriz, pode ser uma tarefa mais trabalhosa que a solução do sistema linear, a afirmação anterior tem mais interesse teórico que prático. Porém, com ela podemos estabelecer algumas condições de convergência mais simples de serem verificadas.

Os métodos iterativos de Jacobi e Gauss-Seidel convergem se uma das condições a seguir for satisfeita:

i. a matriz de coeficientes A, em $Ax = b$, é diagonalmente dominante;

ii. a matriz A é uma matriz positiva definida (isto é $x^T Ax > 0 \ \ \forall x > 0$).

Em alguns casos, mesmo que a matriz não seja diagonalmente dominante, o método de Gauss-Seidel pode, mesmo assim convergir, desde que a condição referente aos autovalores da matriz de iteração B seja satisfeita. Em alguns casos, reordenando adequadamente as equações, pode-se obter uma matriz diagonalmente dominante.

2.4.4 O método de Sobre-Relaxação Sucessiva (SOR)

O método de sobre-relaxação sucessiva, ou SOR é uma técnica de aceleração da convergência dos métodos iterativos. Segundo Barret [4] o SOR é utilizado aplicando extrapolação ao método de Gauss-Seidel. Esta extrapolação leva a forma de uma média entre o valor $x^{(k)}$ obtido na iteração (k) e o valor $x^{(k+1)}$, que seria obtido pelo método de Gauss-Seidel. As iterações associadas ao parâmetro ω, chamado de parâmetro de sobre-relaxação, são definidas por

$$x_i^{(k)} = \omega \bar{x}_i^{(k)} + (1 - \omega) x_i^{(k-1)}, \tag{2.27}$$

onde $\bar{x}$ denota a iteração de Gauss-Seidel. A idéia é escolher um valor para ω que acelerará a taxa de convergência do método iterativo para a solução. O pseudo-código para o algoritmo SOR, segundo [4], é dado pelo algoritmo 2.3:

Algoritmo 2.3 *O método SOR*

1. *Escolher um valor inicial* $x^{(0)}$ *como solução* x
2. *Para* $k = 1, 2, ...$
3. *Para* $i = 1, 2, \ldots, n$
4. $\sigma = 0$
5. *Para* $j = 1, 2, ..., i-1$
6. $\sigma = \sigma + a_{ij} x_j^{(k)}$
7. *Fim*
8. *Para* $j = i+1, ...n$
9. $\sigma = \sigma + a_{ij} x_j^{(k-1)}$
10. *Fim*
11. $\sigma = (b_i - \sigma / a_{ii}$
12. $x_i^{(k)} = x_i^{(k-1)} + \omega(\sigma - x_i^{(k-1)})$
13. *Fim*
14. *Cheque a convergência, continue se necessário*
15. *Fim*

A escolha do valor de ω. Quando o parâmetro de sobre-relaxação ω é igual a 1, o método SOR é simplesmente o método de Gauss-Seidel. Barret [4], usando o teorema de Kahan, mostra que o SOR não converge se ω está fora do intervalo $(0, 2)$. A escolha $1 < \omega < 2$, caracteriza métodos de sobre-relaxação, ao passo que os métodos de sub-relaxação são obtidos com valores de $0 < \omega < 1$.

Pela prática, os melhores resultados são obtidos pela sobre-relaxação. Em geral, não é possível calcular antecipadamente um valor de ω que seja ótimo com respeito ao raio de convergência do SOR. Mesmo que fosse possível calcular esse valor ótimo de ω, o custo computacional se tornaria proibitivo.

Se a matriz de coeficientes A, é simétrica e positiva definida, pode-se garantir que a iteração SOR converge para qualquer valor de ω entre 0 e 2. No entanto, a escolha de ω afeta significativamente a taxa de convergência do SOR. Uma implementação mais sofisticada do algoritmo SOR (como por exemplo a encontrada no ITPACK [28]), emprega uma estimativa mais apropriada do parâmetro ω para melhorar a taxa de convergência da iteração.

Para matrizes de coeficientes A de uma classe especial chamada de *consistentemente ordenadas* (veja [4, p.12]), que inclui certo ordenamento de matrizes surgidas de discretizações de equações diferenciais parciais (EDP) elípticas, há uma relação direta entre os espectros das matrizes de iteração de Jacobi e SOR. A princípio, determinado o raio espectral[4] ρ da matriz de iteração de Jacobi, pode-se determinar a priori, teoricamente, um valor ótimo de ω para o SOR

$$\omega_{opt} = \frac{2}{1+\sqrt{1-\rho^2}}. \tag{2.28}$$

O valor ótimo de ω raramente é calculado, pois, para calcular o raio espectral da matriz de Jacobi é necessário uma quantidade impraticável de cálculos computacionais. Porém, podemos conseguir estimativas razoáveis para o valor ótimo de ρ usando métodos menos sofisticados e de menor custo (por exemplo, os métodos de Golub e Van Loan [23, p.351]).

2.4.5 O método de Sobre-Relaxação Sucessiva Simétrica (SSOR)

Se assumirmos que a matriz de coeficientes A é simétrica, então o método de sobre-relaxação sucessiva simétrica (SSOR) combina duas varreduras conjuntas do SOR de tal modo, que a matriz de iteração resultante é semelhante a uma matriz simétrica. Especificamente, a primeira varredura do SOR é realizada como em (2.23), mas na segunda varredura do SOR os elementos são atualizados na ordem inversa. Isto é, o SSOR representa uma varredura de SOR para frente, seguida de outra varredura SOR para trás. A semelhança da matriz de iteração SSOR com uma matriz simétrica, permite sua aplicação como um pré-condicionador, para outros métodos iterativos em matrizes simétricas. De fato esta é a principal motivação para seu uso, já que sua taxa de convergência, com um valor ótimo de ω tem normalmente taxa de convergência mais lenta que do SOR, também com um ótimo ω. Em termos matriciais, o SSOR é expressado como segue:

$$x^{(k)} = B_1 B_2 x^{(k-1)} + \omega(2-\omega)(D-\omega U)^{-1} D (D-\omega L)^{-1} b, \tag{2.29}$$

[4] O maior valor absoluto dos autovalores de uma matriz A é chamado raio espectral e é denotado por $\rho(A)$: $\rho(A) = \max_{\lambda \in \sigma(A)} |\lambda|$.

onde

$$B_1 = (D - \omega U)^{-1}(\omega L + (1 - \omega)D),$$

e

$$B_2 = (D - \omega L)^{-1}(\omega U + (1 - \omega)D).$$

Note que B_2 é a iteração simples para SOR em (2.23) e que B_1 é similar, apenas com os papéis de L e U invertidos. O algoritmo 2.4 é o pseudo-código do SSOR.

Algoritmo 2.4 *O método SSOR*

Escolher um valor inicial $x^{(0)}$ *como solução* x
Faça $x^{(\frac{1}{2})} = x^{(0)}$
Para $k = 1, 2, ...$
    Para $i = 1, 2, \ldots, n$
        $\sigma = 0$
        Para $j = 1, 2, ..., i-1$
            $\sigma = \sigma + a_{i,j} x_j^{(k-\frac{1}{2})}$
        Fim
        Para $j = i+1, ...n$
            $\sigma = \sigma + a_{i,j} x_j^{(k-1)}$
        Fim
        $\sigma = (b_i - \sigma)/a_{i,i}$
        $x_i^{(k-\frac{1}{2})} = x_i^{(k-1)} + \omega(\sigma - x_i^{(k-1)})$
    Fim
    Para $i = n, n-1, ..., 1$
        $\sigma = 0$
        Para $j = 1, 2, ..., i-1$
            $\sigma = \sigma + a_{ij} x_j^{(k-\frac{1}{2})}$
        Fim
        Para $j = i+1, ..., n$
            $\sigma = \sigma + a_{ij} x_j^{(k)}$
        Fim
        $x_i^{(k)} = x_i^{(k-\frac{1}{2})} + \omega\left(\sigma - x_i^{(k-\frac{1}{2})}\right)$
    Fim
Cheque a convergência, continue se necessário
Fim.

O tratamento moderno de dados nos métodos iterativos, baseia-se no método de relaxamento de Southwell [4, p.12], que foi o precursor do método SSOR.

Entretanto, a ordem na qual as aproximações das incógnitas eram relaxadas variava durante a computação. Especificamente, a próxima incógnita era escolhida baseada em estimativas de localização do maior erro na aproximação atual. Por essa causa, o método de relaxamento de Southwell foi considerado não muito prático para a computação automatizada. É interessante notar como a introdução de múltiplas instruções, múltiplos dados (MIMD[5]) em computação paralela, reacenderam o interesse em métodos iterativos, denominados assíncronos ou caóticos (ver Chazan e Miranker , Baudet e Elkin [4, p.13]), que são relacionados de perto ao método original de relaxamento de Southwell. Em métodos caóticos, a ordem de relaxação é livre e elimina assim o custo de sincronização dos processadores. Entretanto, é difícil predizer este efeito na convergência.

A noção de aceleração e de convergência de um método iterativo através da extrapolação, antecede ao desenvolvimento do SOR. Realmente, Southwell usou a sobre-relaxação para acelerar a convergência do método de relaxamento original. Mais recentemente, o método SOR *ad hoc*, no qual um fator ω de relaxamento é usado para atualizar cada variável, tem obtido bons resultados para alguns tipos de problemas [21].

2.5 Métodos iterativos não estacionários

Analisaremos agora outros tipos de métodos iterativos, que são classificados como métodos não estacionários, pois diferem dos estacionários, devido ao fato de que sua computação envolve troca de informações a cada iteração. Tipicamente, as constantes são calculadas através do produto interno dos resíduos, ou por outros vetores, gerados a partir do método iterativo.

[5]do inglês *Multiple Instruction Multiple Data* que representa um dos modelos de computação paralela.

2.5.1 Método Gradiente Conjugado

A busca da solução de sistemas, provenientes da discretização de equações diferenciais parciais, tem motivado o uso dos métodos gradientes conjugados. Estes vêm sendo estudados já há algumas décadas, viabilizados pela utilização de computadores. Vários autores, entre eles Cunha [10], citam o trabalho de Hestenes e Stiefel, como precursores de tal estudo. Para apresentar o Método Gradiente Conjugado, denotado por GC, admitiremos que a matriz dos coeficientes do sistema linear é simétrica e positiva definida.

Métodos iterativos como o GC, são normalmente utilizados em matrizes esparsas. Isto porque, sendo A uma matriz densa, o trabalho para fatorá-la e resolver o sistema por retro-substituições, é bem mais econômico do ponto de vista do fator tempo, do que resolvê-lo iterativamente.

Os métodos tipo gradiente, para resolver sistemas de equações $Ax = b$, têm como idéia básica minimizar a função de $x = (x_1, x_2, ..., x_n)$

$$F(x) = \frac{1}{2}x^T Ax - b^T x \tag{2.32}$$

onde, denotamos o produto escalar de vetores

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

Quando abrirmos a equação matricial, para torná-la mais simples, usaremos $n = 2$, pois seu desenvolvimento é o mesmo para um número arbitrário de componentes. Por exemplo, para verificarmos que $F(x)$ é uma função quadrática nas componentes do vetor x, tomamos $n = 2$ em (2.32) e obtemos

$$F(x) = \frac{1}{2}(a_{11}x_1^2 + 2a_{12}x_1x_2 + a_{22}x_2^2) - b_1x_1 - b_2x_2,$$

usando a simetria da matriz A.

As curvas de nível da função $F(x)$ são definidas por $F(x_1, x_2)$= constante e, portanto, são elipses. Assim, o gráfico de $F(x)$ é um parabolóide e esta função

tem um mínimo, que é único. Da expressão (2.32), vemos que como F é uma função definida para vetores $x = (x_1, x_2, ..., x_n)$, com n componentes, o resultado é um número real. Se A é uma matriz simétrica e positiva definida, pode-se mostrar que F é uma função quadrática, das variáveis $x_1, x_2, ..., x_n$ (para efeito de visualização, pode-se fazer uma analogia a um parabolóide no espaço de dimensão n). Por outro lado, sabemos do Cálculo Diferencial que o ponto de mínimo, que não esteja no limite da região de busca, é aquele que anula o vetor gradiente de F. Como por definição as componentes do vetor gradiente são as derivadas parciais de F, o ponto de mínimo é atingido no vetor x que anula o gradiente da função,

$$gradF(x_1, x_2) = \begin{bmatrix} \frac{\delta F}{\delta x_1} \\ \frac{\delta F}{\delta x_2} \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 - b_1 \\ a_{21}x_1 + a_{22}x_2 - b_2 \end{bmatrix} = Ax - b. \tag{2.34}$$

Com isso, temos que $gradF(x) = 0$ significa $Ax = b$, ou, a solução do sistema de equações lineares minimiza a função quadrática e vice-versa. O vetor gradiente aponta a direção de crescimento máximo da função, portanto, é natural que, na busca do mínimo, se caminhe na direção contrária ao gradiente, ou seja,:

$$x^{(k+1)} = x^{(k)} - s_k gradF(x^{(k)}). \tag{2.35}$$

Assim, a partir da aproximação num passo k, é calculada a aproximação no passo $(k+1)$, na direção contrária ao gradiente. O parâmetro real s_k regula o tamanho do passo na k-ésima iteração. Usando a equação (2.34), temos a direção de descida como, $-gradF(x^{(k)}) = b - Ax^{(k)} = r^{(k)}$, que é o resíduo associado à aproximação $x^{(k)}$.

O parâmetro s_k em (2.35), tamanho do passo, será usado na minimização do resíduo associado à aproximação que está sendo calculada. Assim, para calcularmos s que minimiza

$$F(x + sr) = \frac{1}{2}(x + sr)^T A(x + sr) - b^T(x + sr), \tag{2.36}$$

uma vez fixado x, diferencia-se a função com relação à variável s, aplicando as regras da cadeia e diferenciação do produto:

$$\frac{dF}{ds}(x + sr) = \frac{1}{2}r^T A(x + sr) + \frac{1}{2}(x + sr)^T Ar - b^T r.$$

Usando a distributividade nas operações e as propriedades de transposição de vetores e matrizes temos que $x^T Ar = (Ar)^T x = r^T Ax$, uma vez que A é simétrica, e portanto

$$\frac{dF}{ds}(x + sr) = sr^T Ar + r^T(Ax - b) = sr^T Ar - r^T r.$$

E finalmente, igualando a zero a derivada, obtemos o valor que minimiza a função

$$s = \frac{r^T r}{r^T Ar}.$$

Com este valor de s em (2.36) e como $-gradF(x) = r^{(k)}$, o Método Gradiente pode ser resumido no Algoritmo (2.5).

Algoritmo 2.5 *Método Gradiente*

1. *Dados* A, b, max, tol
2. $x^{(0)} = 0$
3. *Para* $k = 0 : max$, *faça*
4. $\quad r = b - Ax$
5. $\quad s = \frac{r^T r}{r^T Ar}$
6. $\quad x^{(k+1)} = x^{(k)} + sr$
7. $\quad$ *Se* $r^T r < tol$
8. $\quad\quad$ *Saída com* $sol = x^{(k+1)}$

2.5.1.1 Método dos Gradientes Conjugados

O método do tipo gradiente mais utilizado é o Método dos Gradientes Conjugados. Lembrando que trabalhamos com uma matriz A, simétrica e positiva definida, temos como base do método, o resultado a seguir.

Propriedade: É possível escolher n direções linearmente independentes, $v_1, v_2, .., v_n$ e, por meio da minimização da função $F(x^{(k)} + sv_k)$, em cada uma das direções separadamente, construir uma seqüência de aproximações que forneça o mínimo da função (2.32) após n passos (n é o número de equações do sistema). Por este método, cada novo vetor v_j é gerado após um ciclo completo de minimização.

Detalhando mais o processo, a partir de uma aproximação inicial $x^{(0)}$, tomamos a primeira direção $v_1 = grad(F(x^{(0)}))$.
Supondo conhecidas as direções $v_1, v_2, ..., v_j$ e as aproximações $x^{(1)}, ..., x^{(j)}$, tomamos:

1. v_{j+1} tal que $v_j^T A v_{j+1} = 0$.
2. s_{j+1} é o número real que minimiza $F(x^{(j)} + s v_{j+1})$, isto é, o mínimo de F ao longo da reta que passa por $x^{(j)}$ e tem a direção de v_{j+1}.
3. $x^{(j+1)} = x^{(j)} + s_{j+1} v_{j+1}$.

Existem várias maneiras de construir vetores satisfazendo o primeiro item. No item 2 procede-se da mesma forma que para minimizar (2.36) e mostra-se que

$$s_{j+1} = \frac{r_j^T v_{j+1}}{v_{j+1}^T A v_{j+1}} \tag{2.37}$$

onde, $r_j = b - Ax^{(j)}$.

Algoritmo 2.6 *Método Gradiente Conjugado*

1. Dados A, b, max, tol
2. $x^{(0)} = 0,\ r = b,\quad v = b,\ \ aux = r^T r$
3. Para $k = 0$*:* $\quad max,\quad$ *faça*
4. $\qquad z = Av$
5. $\qquad s = \frac{aux}{v^T z}$
6. $\qquad x^{(k+1)} = x^{(k)} + sv$
7. $\qquad r = r - sz$
8. $\qquad aux1 = r^T r$
9. $\qquad$ *Se* $aux1 < tol,\quad$ *então*
11 $\qquad\quad$ *Saída com* $sol = x^{(k+1)}$
12. $\qquad m = aux1/aux;\ \ aux = aux1$
13. $\qquad v = r + mv$
14.Fim

Assim, o algoritmo 2.6, representa uma das formas do método dos gradientes conjugados. O custo computacional em cada ciclo desse método envolve um produto matriz vetor, dois produtos internos e três somas de vetores.

Conforme deixamos claro no início, este método só é aplicável para sistemas cujas matrizes são simétricas e positivas definidas. Para eliminar tal dificuldade, uma saída seria pré-multiplicar, o sistema linear $Ax = b$ por A^T obtendo

$$A^T Ax = A^T b.$$

Assim, como $A^T A$ é uma matriz simétrica e positiva definida, pode ser usado o método GC neste sistema, que é equivalente ao inicial. Porém, esta forma não é recomendável se a matriz for mal condicionada, pois neste caso seus autovalores estariam num intervalo muito grande ou muito próximos de zero. Como os autovalores de $A^T A$ são os mesmos de A elevados ao quadrado, o problema com esse novo sistema tem resolução mais complexa. Para estes casos, existem outros métodos, como GMRES, QMR, entre outros. Não há na literatura especializada, muito consenso em que um determinado método seja melhor que outro, para este ou aquele problema. Pode-se encontrar muitos deles em pacotes computacionais, como o MATLAB por exemplo.

2.5.2 Método do Resíduo Mínimo Generalizado (GMRES)

O método do Resíduo Mínimo Generalizado (GMRES), descrito por Saad e Schultz [41] e da Cunha [12], é um método iterativo para a solução de sistemas lineares, de n equações, não simétricos e não singulares,

$$Ax = b, \tag{2.38}$$

considerado muito eficiente. O GMRES é uma modificação do método de Ortogonalização Completa [37], que utiliza o processo de Arnoldi. O método de Arnoldi [38, p.147][41], baseia-se no método de ortogonalização de Gram-Schmidt para calcular uma base ortonormal de vetores $\{v_1, v_2, ..., v_k\}$ do subespaço de Krylov $\kappa_k \equiv \{v_1, Av_1, A^2v_1, ..., A^{k-1}v_1\}$.

Becker [5], em sua dissertação de mestrado, o descreve como uma modificação do método de ortogonalização que usa o processo de Arnoldi, citado anteriormente, para calcular uma base ortonormal do subespaço de Krylov $\kappa_k(A, v_1)$. A

solução $x^{(k)}$ é dada por $x^{(0)} + Vy^{(k)}$, onde V é a matriz cujas colunas são os vetores ortonormais v_k calculados pelo processo de Arnoldi. O vetor $y^{(k)}$ é a solução do sistema $H^{(k)}y^{(k)} = \|r^{(0)}\|_2 e_1$, onde $r^{(0)} = b - Ax^{(0)}$, $H^{(k)}$ é a matriz de Hessenberg superior $k \times k$ gerada durante o processo de Arnoldi e e_1 é o vetor da base canônica $(1, 0, 0, ..., 0)^T$ de dimensão k.

O problema que muitas vezes aparece no uso do GMRES, é que o número n de vetores que precisam ser armazenados cresce linearmente com k, e o número de multiplicações cresce quadraticamente, necessitando para isso uma grande quantidade de memória. Em versões reiniciadas do método GMRES, os custos de computação e armazenamento são limitados; este é o método implementado no PIM [12][13]. Em vez de gerar uma base ortonormal de dimensão k, escolhemos um valor c, $c \ll n$, usualmente pequeno ($10 \leq c \leq 50$) e geramos uma aproximação da solução, usando uma base ortonormal de dimensão c. Com isso, podemos estimar a quantidade de armazenamento necessária e escolher c dentro dos limites estabelecidos. Em [11] (apêndice A) encontra-se uma descrição detalhada do GMRES(c) e [5], descreve o método iterativo GMRES em blocos dinâmicos para a solução de sistemas lineares em computadores paralelos, em sua forma não reiniciada.

Embora a versão reiniciada do GMRES não pare [41] (no sentido em que não há divisão por zero), dependendo do sistema e do valor de c, pode produzir uma seqüência estacionária de resíduos, e assim não ocorrer convergência.

Em relação às observações anteriores, Becker [5], cita em seu trabalho, quatro características importantes do GMRES:

1. O GMRES nunca pára, a não ser que tenha convergido [41, p.865, proposição 2]; ou que tenha sido satisfeito um critério de parada.

2. Converge em no máximo n iterações [41, p.865, corolário 3];

3. A convergência é monotônica, i. e. $\|r^{(k+1)}\|_2 \leq \|r^{(k)}\|_2$. Se o campo de valores de A, definido por $F = \{x^T Ax/x^T x,\ x \in \mathcal{C}^N\}$[35, p. 41], es-

tiver contido no semi-plano direito, é possível que haja monotonicidade estrita.

4. O número de vetores de ordem n que precisam ser armazenados cresce linearmente com k e o de multiplicações cresce quadraticamente.

Este último ítem, justifica o uso da versão reiniciada do GMRES e também a escolha do parâmetro c. Aumentar o valor de c faz com que o raio de convergência aumente ou permaneça constante.

A forma mais popular do GMRES é baseada no procedimento modificado de Gram-Schmidt com a versão reiniciada, para controlar exigências de armazenamento. Se nenhum passo de reinicialização é usado, o GMRES (assim como qualquer método do subespaço de Krylov ortogonalizado), convergirá dentro de não mais do que n passos, assumindo uma aritmética exata. Porém, isto não tem nenhum valor prático quando n for muito grande o que implica num número de iterações também muito grande.

Saad e Schultz [41], provaram vários resultados úteis. Em particular, eles mostram que se a matriz de coeficientes A é real e positiva, o GMRES(c) converge para qualquer $x^{(0)}$ desde que c seja maior que um certo valor mínimo. Encontrar esse valor mínimo envolve a determinação da distribuição dos autovalores de A, e assim é usada somente em casos especiais.

A dificuldade está, pois, em escolher um valor apropriado para c. Se c é muito pequeno, o GMRES(c) pode convergir muito lentamente, ou não convergir completamente. Um valor de c maior que o necessário envolve trabalho excessivo (e usa mais armazenamento). Infelizmente, não há nenhuma regra definida e a escolha de c para reiniciar é uma questão de experiência. No entanto, quando c for igual a n, o GMRES(c) se equivale ao GMRES.

Para uma discussão do GMRES em computadores de memória distribuída e não compartilhada, ver Da Cunha [11][13].

3 TÉCNICAS DE PRÉ-CONDICIONAMENTO

A escolha de um bom pré-condicionador para resolver um sistema linear esparso, é na maioria das vezes, uma combinação de arte e ciência, na visão de Saad [40]. Um pré-condicionador pode ser definido como uma modificação de forma implícita ou explícita, no sistema linear original, para facilitar a solução em um determinado método iterativo. Por exemplo, escalonar todas as filas de um sistema linear para transformar os elementos diagonais em unidade, representa uma forma explícita de pré-condicionamento. Outro exemplo, citado anteriormente, o sistema resultante do método do subespaço de Krylov pode vir a convergir em bem menos passos que o sistema original em algumas vezes. Por outro lado, resolver o sistema linear

$$M^{-1}Ax = M^{-1}b$$

onde, determinar M^{-1} é, em alguns casos, um trabalho complexo, podendo envolver cálculos mais trabalhosos, também representa outra forma de pré-condicionamento. Existe a possibilidade de que M e M^{-1}, não possam ser calculadas explicitamente, porém os processos iterativos operam com A e M^{-1} sempre que houver necessidade.

Um dos modos mais simples e rápidos de se determinar um pré-condicionador é fatorar a matriz original A de forma incompleta. A decomposição é da forma $A = LU - R$, onde L e U têm respectivamente a estrutura das partes inferior e superior de A, e R representa o resíduo ou erro na fatoração. Esta fatoração incompleta, conhecida como $ILU(0)$, normalmente é fácil de calcular e seu custo computacional é pequeno. Por outro lado, conduz freqüentemente a uma aproximação que pode resultar em uma aceleração do subespaço de Krylov que precisa de muitas iterações para convergir.

Geralmente, uma fatoração ILU mais precisa, usada como pré-condicionador associado ao método iterativo leva a uma convergência com um número

menor de iterações, porém seu custo computacional de pré-processamento dos fatores é relativamente alto. Se, no entanto, uma única matriz é usada para resolver vários sistemas esse custo é amortizado.

3.1 Custo de implementação

A utilização de um pré-condicionador em um método iterativo incorre em algum custo extra, pela instalação e a iteração ao aplicá-lo. Há uma troca (ou relação custo-benefício) entre o custo de construir e aplicar o pré-condicionador, e o ganho na velocidade de convergência. Entretanto, alguns pré-condicionadores não apresentam um custo muito significativo em sua fase de construção, como por exemplo, o pré-condicionador SSOR. Mas em outros, tais como a fatoração incompleta, pode haver um trabalho substancial envolvido no processo. Esse custo inicial pode ser amortizado em relação às iterações, ou sobre o uso repetido do mesmo pré-condicionador em sistemas lineares múltiplos.

Para determinar a maioria dos pré-condicionadores o trabalho envolvido é proporcional ao número de variáveis. Isto implica em que o trabalho de cada iteração é multiplicado por um fator constante. Por outro lado, o número de iterações para a convergência, em função do tamanho da matriz, geralmente é melhorado sensivelmente com o uso de pré-condicionadores.

Em máquinas paralelas há uma troca razoável entre a eficácia de um pré-condicionador no sentido clássico, e sua eficiência no método em paralelo. Muitos dos pré-condicionadores tradicionais, conforme Barret et all [4, p. 40], têm um componente seqüencial grande, o que pode prejudicar sua implementação em máquinas paralelas.

3.2 Pré-condicionador de Jacobi e SSOR

O pré-condicionador mais simples é uma matriz diagonal definida por

$$m_{ij} = \begin{cases} \frac{1}{a_{ij}} & se\ i = j \\ 0 & se\ i \neq j \end{cases}$$

que é conhecida como pré-condicionador de Jacobi. É possível usar este pré-condicionador sem nenhum armazenamento extra, além daquele da própria matriz. Entretanto, algumas operações têm, geralmente, um custo relativamente alto. Como dissemos anteriormente, uma iteração para resolver um sistema linear $Ax = b$ tem a forma geral

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b, \tag{3.3}$$

onde M e N representam a partição de A em

$$A = M - N. \tag{3.4}$$

A iteração acima é da forma

$$x^{(k+1)} = Gx^{(k)} + f, \tag{3.5}$$

onde

$$f = M^{-1}b \tag{3.6}$$

e

$$G = M^{-1}N = M^{-1}(M - A) = I - M^{-1}A. \tag{3.7}$$

Para Jacobi e Gauss-Seidel, mostra-se que

$$G_{JA}(A) = I - D^{-1}A \tag{3.8}$$

e

$$G_{GS}(A) = I - (D - E)^{-1}A, \tag{3.9}$$

onde $A = D - E - F$ é a partição de A. A iteração (3.5) tenta resolver

$$(I - G)x = f, \tag{3.10}$$

que devido à expressão (3.7) para G, pode ser reescrita como

$$M^{-1}Ax = M^{-1}b. \tag{3.11}$$

O sistema acima é o sistema pré-condicionado associado à partição $A = M - N$, e a iteração (3.3) é a iteração de ponto fixo nesse sistema pré-condicionado. Para resolver (3.11), pode-se usar uma versão de um pré-condicionador do subespaço de Krylov, por exemplo, o pré-condicionador GMRES. A existência desse pré-condicionador genérico M, ou qualquer versão baseada numa partição geral na qual M é não singular, pode ser usada, sendo que na sua forma ideal M deve se aproximar de A. De qualquer modo, o sistema linear em relação à matriz M deve ser resolvido em cada passo de iteração. Em vista disso, este processo deve ser relativamente econômico.

O pré-condicionador SSOR é definido por

$$M_{SSOR} = (D - \omega E)D^{-1}(D - \omega F).$$

Quando a matriz M acima, é um pré-condicionador, não é necessário uma escolha criteriosa de ω como base de iteração de pontos fixos. Se $\omega = 1$, temos a iteração simétrica de Gauss-Seidel,

$$M_{SGS} = (D - E)D^{-1}(D - F), \tag{3.13}$$

onde podemos observar que $D - E$ é a parte inferior de A, incluindo a diagonal, e $D - F$ é, similarmente, a parte superior de A. Assim,

$$M_{SGS} = LU,$$

com

$$L \equiv (D - E)D^{-1} = I - ED^{-1}, \;\; U \equiv D - F,$$

onde L é matriz triangular inferior e U é matriz triangular superior.

Como podemos observar, a matriz pré-condicionadora SSOR ou SGS é da forma $M = LU$, onde L e U são partes de A. Se a matriz erro $A - LU$ é calculada, então para SGS, por exemplo, encontramos

$$A - LU = D - E - F - (I - ED^{-1})(D - F) = -ED^{-1}F \tag{3.16}$$

O inconveniente é que estes pré-condicionadores (Jacobi e SSOR, citados anteriormente) poderão ser menos efetivos que os métodos de fatoração incompletos, desde que estes existam. A introdução de um parâmetro de relaxamento pode melhorar a eficiência do algoritmo, mas o cálculo do parâmetro de relaxamento ótimo normalmente é não-trivial.

Na tentativa de encontrar L e U que produzam o menor erro possível, desenvolveu-se o estudo da fatoração incompleta, na qual a matriz resíduo $A - LU$ tem elementos zero onde A possui entradas não nulas. Este é o estudo da fatoração $ILU(0)$, que pretendemos desenvolver essencialmente no capítulo 4.

3.3 Pré-condicionador da fatoração incompleta LU - ILU

O processo geral de fatoração incompleta LU (ILU), onde A é uma matriz esparsa de elementos $a_{ij},\ i,j = 1,...,n$, calcula a matriz triangular inferior esparsa L e a matriz triangular superior, também esparsa, U, e gera uma matriz resíduo $R = LU - A$, com entradas nulas em algumas posições. O pré-condicionador ILU (principalmente para M-matrizes[1]) e a discussão da fatoração ILU(0), estarão pormenorizados no próximo capítulo.

3.4 Pré-condicionador Polinomial e o método do Gradiente Conjugado

A associação conveniente de pré-condicionadores no método do Gradiente Conjugado representa, segundo Saad [38], uma poderosa técnica de solução de sistemas de equações lineares esparsos e simétricos positivos definidos. Alguns autores, tais como, Concus, Golub e Meurant[9], Dubois, Greenbaum e Rodrigue [14] e

[1]Há várias definições equivalentes de uma M-matriz. Para nosso propósito, o mais conveniente é que A é uma M-matriz se, sendo positiva definida tem elementos não positivos fora da diagonal. Disto, segue que uma M-matriz tem elementos diagonais positivos.

Johnson, Micchelli e Paul[25], têm trabalhado com algumas possibilidades que tem se mostrado atrativas, com o uso de pré-condicionadores polinomiais.

O princípio do pré-condicionador polinomial (em uma matriz SPD), segundo Rutishauser, citado por Saad [38], consiste em resolver o sistema pré-condicionado

$$s(A)Ax = s(A)b, \tag{3.17}$$

onde s é um polinômio de baixo grau. A escolha do polinômio é de tal forma que a matriz $s(A)A$ tenha uma distribuição de autovalores favorável ao método dos gradientes conjugados, o que possibilita uma rápida convergência. No contexto clássico da computação escalar, pouco se justifica o uso do pré-condicionador polinomial, visto que o método do gradiente conjugado é um processo ótimo. O número total de multiplicações matriz-vetor necessárias aplicadas ao sistema pré-condicionado $s(A)Ax = s(A)b$, em alguns casos, será mais elevado que o para o sistema não pré-condicionado $Ax = b$. Porém, muitas vezes, o custo que incorre do grande número de multiplicações matriz-vetor, compensa o pequeno número de produtos internos requerido quando o pré-condicionador polinomial for usado.

Johnson et al. [25] e Saad [38] apresentam métodos para determinação de polinômios ótimos a serem utilizados como pré-condicionadores no método dos gradientes conjugados. Estes métodos dependem de uma estimativa do maior e menor autovalor da matriz.

Algoritmo 3.1 *Método Gradiente Conjugado Pré-Condicionado*

1. *Dados* A, b, M *(matriz pré-condicionadora)* max, tol
2. $x^{(0)} = 0, \;\; r = b, \;\; v = M^{-1}b, \;\; y = M^{-1}r, \;\; aux = y^T r$
3. *Para* $k = 0 : max, \;\;$ *faça*
4. $\quad z = Av$
5. $\quad s = \frac{aux}{v^T z}$
6. $\quad x^{(k+1)} = x(k) + sv$
7. $\quad r = r - sz$
8. $\quad y = M^{-1}r$
9. $\quad$ *Se* $r^T r < tol$ *então*
   $\quad\quad$ *Saída com* $sol = x^{(k+1)}$
10. $\quad aux1 = y^T r$
11. $\quad m = aux1/aux; \;\; aux = aux1$
12. $\quad\quad = y + mv$
13. *Fim*

3.5 Pré-condicionador da fatoração incompleta de Cholesky

O pré-condicionador obtido com a fatoração incompleta de Cholesky utiliza a fatoração LU da própria matriz A, ou seja, a matriz M correspondente a este método é a fatoração de Cholesky incompleta da matriz A.

Esta fatoração se baseia no método de decomposição de Cholesky para solução de um sistema de equações lineares, onde a matriz A é decomposta da seguinte forma:

$$A = LL^T \tag{3.18}$$

onde L é uma matriz triangular inferior.

A fatoração incompleta de Cholesky proposta por Meijerink e Van der Vorst (1977) [32], representada na equação (3.18) (ver no próximo capítulo), utiliza o algoritmo da decomposição de Cholesky modificado para evitar a utilização de

raiz quadrada para encontrar a fatoração

$$A = LDL^T, \tag{3.19}$$

onde D é uma matriz diagonal. L e D são obtidas da seguinte forma:

$$\begin{aligned} L_{ii} &= A_{ji} - \sum_{k}^{(i-1)} L_{jk} L_{ik} D_{kk} \quad para \quad j = i, i+1, ..., N \\ D_{ii} &= \frac{1}{L_{ii}}. \end{aligned} \tag{3.20}$$

No processo de decomposição, a matriz A perde o padrão de distribuição de zeros, ou seja, elementos nulos podem se tornar não nulos após a decomposição.

Na fatoração incompleta deve-se manter a mesma distribuição de zeros da matriz original. Isto se consegue fazendo com que os elementos nulos na matriz original permaneçam nulos na matriz fatorada.

Diversos autores apresentam estudos sobre os pré-condicionadores das fatorações incompletas, assim como melhoramentos em sua implementação.

Kershaw [27](1978), apresenta uma generalização do método para solução de sistemas de equações diferenciais parciais implícitas. Manteuffel [31] (1980), Munksgaard (1980) e Meijerink e Van der Vorst (1981) apresentam outras variações destes métodos de pré-condicionamento, chamados de fatoração incompleta.

3.6 Características de um bom Pré-condicionador

Embora teoricamente, um algoritmo deva fornecer a solução do sistema após realizadas n iterações, na prática nem sempre isto ocorre, pois se a matriz é mal-condicionada isto pode prejudicar os cálculos, ampliando o efeito dos erros de arredondamento. Isto ocorre quando os autovalores se distribuem num intervalo muito grande ou existem autovalores muito próximos de zero. Segundo Cunha [10], um dos casos em que o método dos gradientes conjugados pode não convergir, é para uma matriz de Hilbert ($a_{ij} = 1/(i+j-1)$), mesmo que sua dimensão não seja

tão grande, pois esta tem um número de condição muito elevado (por exemplo, para uma matriz de dimensão 15, este é da ordem de 10^{17}, o que faz com que se percam muitos algarismos significativos).

Em algumas situações, pode-se escolher uma matriz adequada, que multiplicada ao sistema, pode modificar tal distribuição de autovalores, acelerando a convergência do método GC, como visto anteriormente.

Como antes, o pré-condicionamento consiste em substituir o sistema original $Ax = b$ por um outro equivalente

$$M^{-1}Ax = M^{-1}b,$$

onde M é a matriz pré-condicionadora. Cunha [10], afirma que o bom pré-condicionamento ocorre se M é uma matriz que "aproxima" A e satisfaz as propriedades:

1. M é simétrica positiva definida;
2. $M^{-1}A$ é bem condicionada;
3. a equação $Mx = b$ é de fácil resolução.

As pesquisas referentes ao assunto em questão, vêm sendo realizadas nos últimos anos; entretanto, não encontramos uma metodologia geral para determinar matrizes pré-condicionadoras. Algumas sugestões são: usar $M = D$, no caso dos elementos da diagonal variarem em grande intervalo, usar matrizes bloco-diagonais, fatoração de Cholesky incompleta ou fatoração LU incompleta.

Por outro lado, se no sistema $Ax = b$ a matriz A não for simétrica e positiva definida, os argumentos teóricos aqui apresentados, não são válidos. Isto porque não existe algoritmo do tipo gradiente conjugado, que ao mesmo tempo minimize o resíduo e exija pouco trabalho por iteração, que são as qualidades básicas do método. Algumas alternativas para sanar tais problemas, são sugeridas por Cunha [10]:

i. Trabalhar com o sistema $A^TAx = A^Tb$, que satisfaz a condição de simetria, e assim usar o algoritmo 2.6, ou 3.1;

ii. Procurar algoritmos simples que de alguma forma imitem as idéias dos Gradientes Conjugados, tais como os métodos BCG e CGS.

Uma vez selecionada a matriz pré-condicionadora, Cunha sugere o algoritmo 3.1 para implementar o método.

Axelsson e Lindskog [1], também discutem a relação entre a distribuição dos autovalores de uma matriz e a classe de métodos pré-condicionadores a ser utilizada em cada problema. Os métodos são baseados na fatoração incompleta da matriz A e incluem ambos "pointwise e blockwise" fatoração [1]. O estudo da dependência entre o raio de convergência do pré-condicionador do método do gradiente conjugado, e a distribuição de autovalores de $M^{-1}A$, onde M é a matriz pré-condicionadora, bem como os gráficos referentes aos testes numéricos aplicados aos métodos, podem ser encontrados no artigo citado anteriormente [1].

4 MÉTODOS DE FATORAÇÃO INCOMPLETA

Uma grande classe de pré-condicionadores baseia-se na fatoração incompleta da matriz de coeficientes. Nós chamamos de fatoração incompleta, aquela em que durante o processo de fatoração, as posições em que na matriz original são ocupadas por elementos zeros, continuam desta maneira, ignorando-se qualquer preenchimento que se daria pelos cálculos de atualização. A fatoração obtida ao "descartar" certos elementos durante o processo é chamada de fatoração incompleta. Assim, uma fatoração incompleta LU de uma matriz A, em geral, resulta $A \neq LU$; como espera-se que uma fatoração LU aproxime A e esta é justamente a função do pré-condicionador num método iterativo, temos que procurar fazer uma escolha para que tal objetivo seja alcançado. Para obtermos o pré-condicionador, inicialmente determinamos uma fatoração da forma $M = LU \approx A$ onde L e U são matrizes respectivamente triangular inferior e superior. Barret et al [4], afirma que a eficácia de um pré-condicionador depende de uma boa aproximação M^{-1} de A^{-1}, onde M é a matriz pré-condicionadora, neste caso, construída através de uma fatoração incompleta.

4.1 Implementação da fatoração incompleta

As fatorações incompletas foram os primeiros pré-condicionadores encontrados para os quais há uma fase de criação não-trivial. Segundo Eijkhout [17][18] é necessário, a princípio, definir alguns critérios sobre esses métodos de fatoração, tais como, existência, limitação de memória, e alguns problemas encontrados em sua implementação. Isto porque, segundo Eijkhout, as fatorações incompletas podem apresentar alguns problemas durante o processo, podendo sofrer interrupções (ao tentar dividir pelos pivôs zeros) ou resultarem em matrizes indefinidas (pivôs

negativos), mesmo se a fatoração completa da mesma matriz existe e resulta uma matriz positiva definida.

Se a matriz original é uma M-matriz, pode-se garantir a existência da fatoração incompleta. Isto foi provado originalmente por Meijerink e Van der Vorst [32] e mais tarde por Manteuffel [31].

Nos casos em que os pivôs são zeros ou negativos, algumas estratégias são propostas, como a substituição destes pivôs por um número arbitrário positivo (ver Kershaw [27]), ou reiniciando a fatoração com $A+\alpha I$ para algum valor positivo de α (ver Manteuffel [31]). Também Eijkhout [17],[18], sugere algumas estratégias para minimizar esses problemas e relaciona alguns aspectos da fatoração incompleta.

Uma consideração importante sobre os pré-condicionadores dos métodos de fatoração incompleta é o custo do processo de fatoração. Até mesmo se a fatoração incompleta existir, o número de operações envolvidas para criá-la é pelo menos o mesmo que para resolver um sistema com tal matriz de coeficientes, assim, o custo pode ser igual ao das iterações nos métodos iterativos.

Tais custos podem ser amortizados se o método iterativo leva a muitas iterações, ou mesmo se o pré-condicionador for usado para vários sistemas lineares.

Para uma eficiente solução de sistemas lineares esparsos $Ax = b$ utilizando um método terativo, é importante a escolha de um bom pré-condicionador. Como dissemos anteriormente, tal pré-condicionador é a matriz M, que tenha construção relativamente simples e cuja fatoração se aproxime da matriz original A e para a qual a solução do sistema $Mx = b$ tenha menor custo computacional ou que melhore a taxa de convergência.

Como a matriz de coeficientes A é esparsa, espera-se que a matriz $M = LU \approx A$ também seja esparsa, o que nem sempre ocorre. A matriz M construída como fatoração incompleta, depende de alguns critérios relativos aos índices ou

entradas, quando a atualização

$$a_{ij} \leftarrow a_{ij} - a_{ik} a_{kk}^{-1} a_{kj} \tag{4.1}$$

é feita. O que esperamos de uma fatoração incompleta é que além da mesma existir, possamos obter uma certa exatidão no processo.

4.1.1 Resolução de um sistema com um método de fatoração incompleta

Sobre o processo de resolução de um sistema pelo método de fatoração incompleta, estes podem ser calculados de várias formas. Se $M = LU$ (onde L e U são matrizes triangulares não-singulares), para fatorar A pode-se proceder de modo habitual, como no algoritmo 4.1, mas freqüentemente fatorações incompletas são determinadas como $M = (D + L)D^{-1}(D + U)$ (com D diagonal e L e U matrizes estritamente triangulares inferior e superior, determinadas durante o processo de fatoração). Neste caso, poder-se-ia usar qualquer uma das seguintes formulações equivalentes para $Mx = y$:

$$(D + L)z = y, \quad (I + D^{-1}U)x = z \tag{4.2}$$

ou

$$(I + LD^{-1})z = y, \quad (D + U)x = z. \tag{4.3}$$

Em ambos os casos, os elementos diagonais são usados duas vezes (não três vezes como a fórmula para M levaria a esperar), e como somente produtos por D^{-1} são executadas, armazenar explicitamente D^{-1} é o mais prático a fazer. Podemos armazenar LD^{-1} ou $D^{-1}U$ para diminuir o custo extra de armazenamento. Para resolver o sistema usando a primeira forma ver o algoritmo 4.2. A segunda formulação, segundo Barret et al [4], é mais difícil de implementar.

Algoritmo 4.1 *Pré-condicionador para solução do sistema* $Mx = y$, *onde* $M = LU$

1. *Faça* $M = LU$ *e* y *conhecido*
2. *Para* $i = 1, 2, ...$
3. $\quad z_i = l_{ij}^{-1}(y_i - \sum_{j<i} l_{ij} z_j)$
4. $\quad$ *Para* $i = n, n-1, n-2, ...$
5. $\quad\quad x_i = u_{ii}^{-1}(z_i - \sum_{j>i} u_{ij} x_j)$
6. $\quad$ *Fim*
7. *Fim*

Algoritmo 4.2 *Pré-condicionador para solução do sistema* $Mx = y$, *onde* $M = (D + L)D^{-1}(D + U) = (D + L)(I + D^{-1}U)$

1. *Faça* $M = (D + L)(I + D^{-1}U)$ *e* y *conhecido*
2. *Para* $i = 1, 2, ...$
3. $\quad z_i = d_{ii}^{-1}(y_i - \sum_{j<i} l_{ij} z_j)$
4. $\quad$ *Para* $i = n, n-1, n-2, ...$
5. $\quad\quad x_i = z_i - dii^{-1} \sum_{j>i} u_{ij} x_j$
6. $\quad$ *Fim*
7. *Fim*

4.1.2 Observações sobre os métodos de fatoração incompleta

A forma mais comum de fatoração incompleta consiste em adotar um subconjunto S das posições da matriz, excluindo todas as posições de elementos iguais a zero para proceder a fatoração. A fatoração resultante é incompleta quando preenchimentos são ignorados.

A escolha usual do conjunto S refere-se às posições (i, j) para as quais $a_{ij} \neq 0$. Uma posição que é zero em A mas não o é em uma fatoração exata, é chamada uma posição de preenchimento e se ela estiver fora de S, tal preenchimento deve ser ignorado. Este tipo de fatoração é chamada de fatoração $ILU(0)$: fatoração incompleta LU de nível zero, ou seja, não são permitidos preenchimen-

tos em posições que na matriz original continham elementos zeros. Barret et al [4] descreve a fatoração incompleta da forma a seguir

$$\text{para cada} \quad k,i,j > k : a_{ij} \leftarrow a_{ij} = \begin{cases} a_{ij} - a_{ik}a_{kk}^{-1}a_{kj} & se \;\; (i,j) \in S \\ a_{ij} & no \;\; outro \;\; caso \end{cases}$$

Meijerink e Van der Vorst [32], mostraram que se A é uma M-matriz, a fatoração existe para alguma escolha conveniente de S, e fornece uma matriz simétrica e positiva definida se A é uma matriz simétrica e positiva definida. Algumas normas para estabelecer níveis de preenchimento são dados por Meijerink e Van der Vorst [33].

4.1.3 Estratégias de preenchimento

Dentre as estratégias de aceitação ou descarte de preenchimentos [6] [17], duas são consideradas principais: estrutural e numérica.

A estratégia estrutural é a que aceita preenchimentos até um certo nível. Uma posição zero (i,j) preenchida em A, possui um nível de preenchimento associado
$fill(i,j) = 1 + max\{fill(i,k),\ fill(k,i)\}$. Se $a_{ij} \neq 0$, o nível de preenchimento não é modificado.

A estratégia numérica de preenchimento é a que apresenta "pequena tolerância": ignora preenchimentos se os elementos são muito pequenos, para uma adequada definição de "pequenos". Embora tal definição faça mais sentido matematicamente, ela é mais difícil de implementar na prática, pois a quantidade de armazenamento necessária para a fatoração é difícil de predizer. Para maior discussão sobre formas de armazenamento e uso de pré-condicionadores usando tal tipo de tolerância ver [2][3].

4.2 Casos simples: ILU(0) e ILU-D

Embora no método ILU(0), a fatoração incompleta procure não preencher posições que na matriz original e esparsa A são zeros, alguns preenchimentos podem ser feitos. Mesmo assim o pré-condicionador, no pior caso, ocupa exatamente o mesmo espaço de armazenamento que a matriz original A. Segundo Eisenstat [19], uma simplificação da versão ILU(0), chamada D-ILU ou ILU-D (por Claude Pommerell, em, "Solution of large Unsymmetric Systems of Linear Equations, 1992") requer menos memória.

Particionando a matriz de coeficientes A nas suas partes diagonal, triangular inferior e triangular superior $A = D_A + L_A + U_A$, o pré-condicionador pode ser escrito como $M = (D+L_A)D^{-1}(D+U_A)$ onde D é a matriz diagonal que contém os pivôs gerados ou modificados. A construção do pré-condicionador da fatoração incompleta ILU-D é descrito no algoritmo 4.3, onde são armazenados os inversos dos pivôs.

Algoritmo 4.3 *Pré-condicionador da fatoração incompleta ILU-D*

1. *Faça* S *o conjunto não nulo* $\{(i,j) : a_{ij} \neq 0\}$
2. *Para* $i = 1, 2, ...$
3. $\quad d_{ii} \leftarrow a_{ii}$
4. *Para* $i = 1, 2, ...$
5. $\quad d_{ii} \leftarrow 1/d_{ii}$
6. $\quad$ *Para* $j = i+1, i+2, ...$
7. $\quad\quad$ *Se* $(i,j) \in S$ *e* $(j,i) \in S$, *então*
8. $\quad\quad d_{jj} \leftarrow d_{jj} - a_{ji} d_{jj} a_{ij}$

.

No caso geral, preenchimentos podem ser aceitos ou descartados em qualquer posição. Se um parâmetro de entrada r é usado, por exemplo na operação

$$Se \ \ |a_{ij}| \geq r a_{ii} : \quad a_{ij} \leftarrow a_{ij} - a_{ik} a_{kk}^{-1} a_{kj}, \tag{4.4}$$

chamamos a isto "estratégia numérica de eliminação", e o método de ILU(r).

4.3 O método de Kershaw

Quanto ao surgimento de pivôs zeros ou negativos no método de fatoração incompleta, que podem levar à uma parada ou resultado indesejado (como foi comentado no início desta seção), pode-se tomar algumas medidas, sugeridas por Kershaw [17], substituindo um valor positivo arbitrário nas posições de zeros ou pivôs negativos. Enquanto trivialmente isto possa garantir a existência da fatoração, é provável que esta conduza a um número de condição muito grande para $M^{-1}A$.

A escolha de tal valor arbitrário deve ser de forma "não trivial", ou seja atendendo a um determinado critério pré-estabelecido. Se a escolha for um valor muito pequeno, o processo poderá tornar-se instável, segundo Elman [20]. Assim, mesmo que o pré-condicionador seja SPD, o método iterativo poderá divergir. Superestimando o valor ótimo de conserto, este tornará o pré-condicionador "muito" diagonal dominante, na prática, transformando-o no método de Jacobi. A escolha de Kershaw [17][p.6], para o pivô de conserto, é

$$m_{kk} = \sum_{j<k} |m_{kj}| + \sum_{j>k} |m_{jk}|,$$

onde os m_{ij} são os elementos de M. Uma escolha similar

$$m_{kk} = \sum_{j<k} |m_{kj}| + \sum_{j>k} |m_{kj}|,$$

foi usada por Van der Vorst, motivado por considerações de estabilidade.

4.4 O método de Manteuffel

Desde que o principal problema com respeito à fatoração incompleta sejam pivôs negativos, somando-se um número conveniente à diagonal da matriz, pode-se obter uma fatoração bem definida. Trivialmente, acrescentar tal valor para obter a matriz diagonalmente dominante, é uma condição suficiente, mas é provável que isto conduza a um sistema mal-condicionado. Mauteuffel [22] propôs algumas alternativas para encontrar um pequeno, mas conveniente valor de α, tal que $A+\alpha I$,

possua uma fatoração bem definida de M, e que não resulte em um número de condição muito grande em $M^{-1}A$.

4.5 Fatoração incompleta LU

Um algoritmo geral da fatoração incompleta LU pode ser obtido através da eliminação Gaussiana e substituição de alguns elementos em posições pré-determinadas fora da diagonal. Para analisar este processo e estabelecer a existência da fatoração para M-matrizes, Saad [40, p.270] apresenta o seguinte teorema:

Teorema 4.1 *Seja A uma M-matriz e seja A_1 a matriz que se obtém após o primeiro passo de eliminação Gaussiana. Então A_1 é uma M-matriz.*

Prova Para uma matriz A ser uma M-matriz as seguintes condições são necessárias:

1. $a_{ij} \leq 0 \quad$ para $\quad i \neq j, \quad i, j = 1, ..., n.$

2. A é não singular.

3. $A^{-1} \geq 0.$

4. $a_{ii} > 0 \quad$ para $\quad i = 1, ..., n.$

Saad mostrou [40, teorema 1.17, p.29] que as 3 primeiras condições são suficientes, ou seja, que elas implicam a propriedade 4.

Para mostrarmos que A_1 é uma M-matriz, mostraremos que A_1 satisfaz as propriedades 1, 2 e 3. Os elementos fora da diagonal de A_1 são dados por

$$a_{ij}^1 = a_{ij} - \frac{a_{i1}a_{1j}}{a_{11}}.$$

Como a_{ij}, a_{i1}, a_{1j} são não positivos e a_{11} é positivo, segue que $a_{ij}^1 \leq 0$ para $\quad i \neq j$, o que prova a propriedade 1.

O fato de A_1 ser não singular vem da relação entre A_1 e A.

$$A = L_1 A_1 \text{ onde } \; L_1 = \left[\frac{A_{*1}}{a_{11}}, e_2, e_3, ..., e_n \right]. \tag{4.5}$$

(e portanto $A_1^{-1} = A^{-1}L_1$ existe), o que prova a propriedade 2.
Finalmente, para provar que todos os elementos de A_1^{-1} são não negativos, consideramos as colunas de A_1^{-1}.
A 1^a coluna de A_1^{-1} é dada por

$$A_1^{-1}e_1 = \frac{1}{a_{11}}\, e_1 \geq 0,$$

devido a estrutura de A_1 (que tem como primeira coluna $[a_{11}\, 0 \ldots 0]^T$).
As demais colunas de A_1^{-1} são dadas por

$$A_1^{-1}e_j = A^{-1}L_1e_j = A^{-1}e_j \geq 0.$$

(onde usamos a equação (4.5) e a propriedade 3 para A).

Assim todas as colunas de A_1^{-1} são não negativas e portanto os elementos de A_1^{-1} são não negativos, o que prova a propriedade 3.

Assim pode-se verificar que a matriz $(n-1) \times (n-1)$ obtida de A_1 quando removemos a primeira linha e a primeira coluna também é uma M-matriz.

Assumimos que alguns elementos em posições fora da diagonal, são descartados durante o processo de eliminação Gaussiana. Qualquer elemento descartado é um elemento não positivo que é transformado em zero. Assim a matriz resultante $\tilde{A}_1$ é tal que,

$$\tilde{A}_1 = A_1 + R,$$

onde os elementos de R são tais que, $r_{ii} = 0,\ r_{ij} \geq 0$. Assim,

$$A_1 \leq \tilde{A}_1$$

e os elementos que estão fora da diagonal de $\tilde{A}_1$ são não positivos. Como A_1 é uma M-matriz, $\tilde{A}_1$ também é (ver Saad [40, teorema 1.18 p.30]). Repete-se este processo para a matriz $\tilde{A}(2:n, 2:n)$, até que se obtenha a matriz da fatoração incompleta de A. Deste modo, pode-se verificar que a cada passo a matriz obtida é uma M-matriz e o processo continua sem problemas.

Para a escolha dos elementos a serem eliminados em cada passo do processo, uma das maneiras é estabelecer um padrão de elementos não nulos fora da diagonal. Um padrão de zeros P será um subconjunto de índices

$$P \subset \{(i,j)|i \neq j;\ 1 \leq i,j \leq n\}, \tag{4.6}$$

para os quais preenchimentos não devem ser feitos. A fatoração incompleta LU, ILU_P, pode ser calculada utilizando-se um determinado padrão P, conforme o pseudo-código do algoritmo 4.4.

Algoritmo 4.4 *Padrão Geral ILU*

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

No algoritmo 4.4, o laço da linha 4 é interpretado da seguinte forma: para $j = k+1, ..., n$ e somente para aqueles índices j que não estão em P, execute a próxima linha. Saad afirma que não há necessidade de examinar j de $k+1$ até n, por existir uma forma mais econômica para identificar os elementos que estão no complemento de P. Saad [40], usa estes argumentos para provar o teorema seguinte.

Teorema 4.2 *Seja A uma M-matriz e P um dado padrão de zeros definido como em (4.6). Então o algoritmo 4.4 é possível e produz uma fatoração incompleta,*

$$A = LU - R \tag{4.7}$$

que é uma partição regular de A[1].

[1] *Definimos uma partição regular de A como $A = M - N$, onde, se M é não-singular, M^{-1} e N, são também não-singulares*

A demonstração deste teorema, dada a seguir, encontra-se em [40][p.271].

Prova .

A cada passo do processo, temos

$$\tilde{A}_k = A_k + R_k, \;\; A_k = L_k \tilde{A}_{k-1}$$

onde, usando 0_k para denotar o vetor zero de dimensão k, e $A_{m:n,j}$ denota o vetor de componentes a_{ij}, com $i = m, ..., n$,

$$L_k = I - \frac{1}{a_{kk}^{(k)}} \begin{pmatrix} 0_k \\ A(k+1:n,k) \end{pmatrix} e_k^T.$$

Temos portanto as relações

$$\tilde{A}_k = A_k + R_k = L_k \tilde{A}_{k-1} + R_k.$$

Aplicando recursivamente esta relação, com k variando de $n-1$ até 1, obtemos

$$\tilde{A}_{n-1} = L_{n-1}...L_1 A + L_{n-1}...L_2 R_1 + ... + L_{n-1} R_{n-2} + R_{n-1}. \tag{4.8}$$

Chamando

$$L = (L_{n-1}...L_1)^{-1}, \;\; U = \tilde{A}_{n-1}.$$

Temos,

$$U = L^{-1} A + S$$

com

$$S = L_{n-1}...L_2 R_1 + ... + L_{n-1} R_{n-2} + R_{n-1}.$$

Observe que no estágio k, os elementos são eliminados somente na parte inferior $(n-k) \times (n-k)$ de A_k. Portanto, as primeiras k linhas e colunas de R_k são nulas e

$$L_{n-1}...L_{k+1} R_k = L_{n-1}...L_1 R_k,$$

assim, S pode ser reescrito como

$$S = L_{n-1}...L_2 (R_1 + R_2 + ... + R_{n-1}).$$

e representando por R a matriz

$$R = R_1 + R_2 + ... + R_{n-1},$$

obtemos

$$S = L^{-1}R, \;\; U = L^{-1}A + L^{-1}R, \;\; LU = A + R$$

e portanto

$$A = LU - R,$$

onde $(LU)^{-1} = U^{-1}L^{-1}$ e R são matrizes não negativas.

Considerando alguns aspectos práticos em relação à implementação da fatoração ILU com base no algoritmo 4.4, observa-se que sua implementação não é tão simples, pois a cada passo k, todas as linhas de $k+1$ até n sofrem modificações.

Como a fatoração ILU depende da eliminação Gaussiana usada, verificamos que estas estão relacionadas com a ordem dos três laços associados às três variáveis de controle i, j e k no algoritmo. O algoritmo 4.4 é derivado da variante k, i, j. Para fatorações LU incompletas a ordem i, k, j é a mais utilizada se os dados forem armazenados por linha. Nesta versão a linha i de L e U são geradas simultaneamente (ver Saad [38]).

A adaptação desta variante para matrizes esparsas é obtida sem dificuldades, pois, as linhas de L e U são geradas em sucessão. Tais linhas são calculadas uma de cada vez, e podem ser armazenadas em uma estrutura de dados "linha orientada", como o formato CSR[2]. A fatoração incompleta ILU com as variantes i, k, j nesta ordem aparece no algoritmo 4.5.

[2]para verificar esta forma de armazenamento de matrizes ver Saad [38, p.84]

Algoritmo 4.5 *Fatoração Geral ILU versão IKJ*

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

As duas versões são equivalentes, no entanto, este último algoritmo é de implementação mais fácil.

É útil analisar o resultado de um passo de eliminação incompleta. Denotando por l_{i*}, u_{i*} e a_{i*}, a i-ésima linha de L, U e A, respectivamente, então o laço k, na linha 2 do algoritmo 4.5, pode ser interpretado da seguinte forma. Inicialmente, temos $u_{i*} = a_{i*}$; então, cada passo da eliminação é uma operação da forma

$$u_{i*} = u_{i*} - l_{ik}u_{k*}. \tag{4.9}$$

Como esta operação só é executada no complemento de P, isto significa que, na realidade, o passo de eliminação assume a forma

$$u_{i*} = u_{i*} - l_{ik}u_{k*} + r_{i*}^{(k)}, \tag{4.10}$$

onde, $r_{ij}^{(k)}$ é igual a zero, quando $(i,j) \notin P$ e igual a $l_{ik}u_{kj}$, quando $(i,j) \in P$. Assim, a linha $r_{i*}^{(k)}$ cancela os termos $l_{ik}u_{kj}$ que seriam de outra forma introduzidos no padrão zero. Ao final, obtemos a seguinte relação:

$$u_{i*} = a_{i*} - \sum_{k=1}^{i-1} \left(l_{ik}u_{k*} - r_{i*}^{(k)} \right). \tag{4.11}$$

Observe que $l_{ik} = 0$ para $(i,k) \in P$. Definimos

$$r_{i*} = \sum_{k=1}^{i-1} r_{i*}^{(k)}. \tag{4.12}$$

A linha r_{i*} contém a soma de todos os elementos $r_{i*}^{(k)}$ do padrão P. Usando o fato de que $l_{ii} = 1$, obtemos a relação

$$a_{i*} = \sum_{k=1}^{i-1} l_{ik} u_{k*} - r_{i*}. \tag{4.13}$$

E assim temos a seguinte proposição.

Proposição 4.2. O algoritmo 4.5 produz fatores L e U tais que

$$A = LU - R,$$

na qual $-R$ é a matriz dos elementos que são descartados durante o processo de eliminação incompleto. Quando $(i, j) \in P$, um elemento r_{ij} de R é igual ao valor de $-a_{ij}$ obtido ao final do laço k, no algoritmo 4.5. Caso contrário, r_{ij} é nulo.

4.5.1 A fatoração ILU(0)

A técnica de fatoração incompleta LU sem o preenchimento de posições nulas, denotado por ILU(0), consiste em tomar o padrão de zeros P, como sendo exatamente o padrão de zeros de A. Denotaremos por b_{i*} a i-ésima linha de uma determinada matriz B, e por $NZ(B)$, o conjunto de pares (i, j), $1 \leq i, j \leq n$, tal que $b_{ij} \neq 0$.

A fatoração incompleta ILU(0) foi desenvolvida originalmente para matrizes de 5-pontos e 7-pontos, relacionadas à discretização por diferenças finitas de equações diferenciais parciais (EDPs) [40].

Saad [40][p.275] ilustra este exemplo considerando uma matriz A como da figura 4.1, que representa uma matriz de 5-pontos de tamanho $n = 32$, que corresponde a uma malha $n_x \times n_y = 8 \times 4$. Considera uma matriz L triangular inferior, com a mesma estrutura da parte inferior de A, e uma matriz U, também triangular, com a mesma estrutura da parte superior de A. Ao fazer o produto LU obtém-se a matriz mostrada na parte inferior direita da figura 4.1.

Devido às diagonais extras, em geral o produto LU não fornece exatamente a matriz A, para quaisquer L e U. Os elementos das diagonais extras são

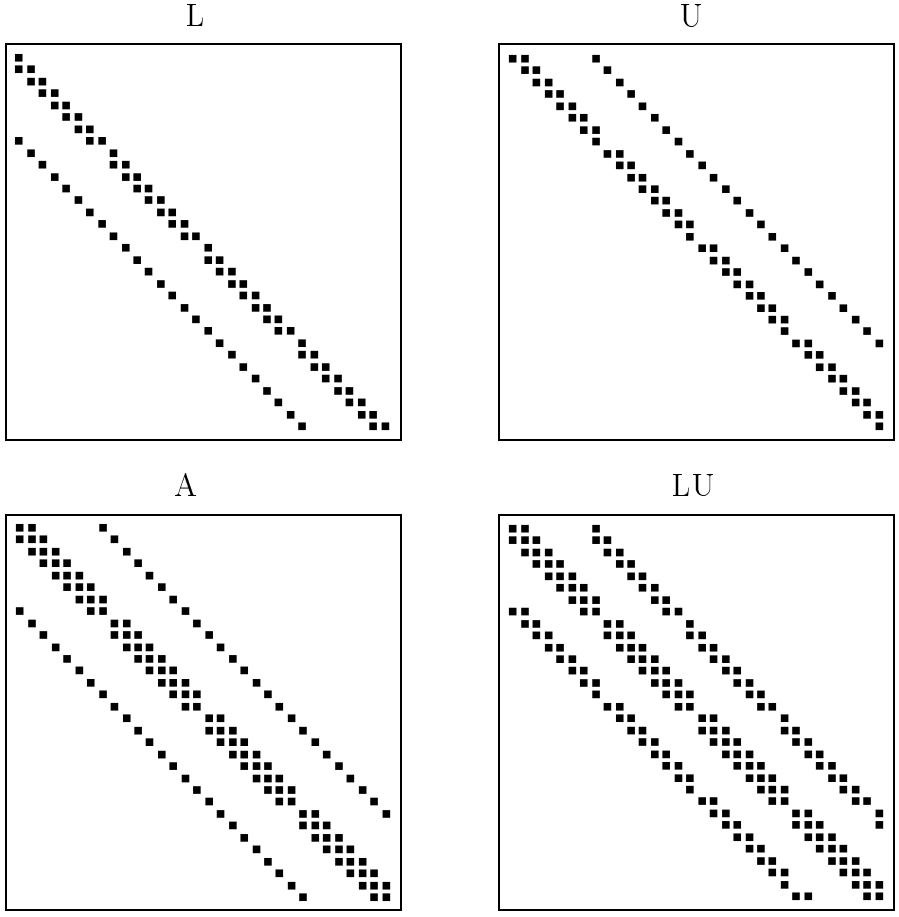

Figura 4.1: *Fatoração ILU(0) para matriz de 5 pontos*

chamados de *elementos de preenchimento*. Porém, pode ser possível achar L e U de forma que o produto se iguale a A, nas outras diagonais, se ignorarmos estes elementos de preenchimento. Teremos uma fatoração ILU(0) se obtivermos um par de matrizes triangulares L (inferior unitária) e U (superior), de tal forma que os elementos de $A - LU$ são nulos nas localizações de $NZ(A)$. Como geralmente podemos determinar infinitos pares de matrizes satisfazendo tais exigências, estas restrições não definem os fatores de ILU(0) unicamente. No entanto podemos definir ILU(0) construtivamente no algoritmo 4.6, que usa o algoritmo 4.5 com o padrão P igual ao padrão de zeros de A.

Algoritmo 4.6 *Fatoração ILU*(0)

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

Em alguns casos, é possível escrever a fatoração ILU(0) na forma

$$M = (D - E)D^{-1}(D - F), \tag{4.14}$$

onde $-E$ e $-F$ são as partes triangulares estritamente inferior e superior de A e D é uma matriz diagonal, geralmente diferente da diagonal de A. Nestes casos é suficiente encontrar uma fórmula recursiva para determinar os elementos de D. Como se requer apenas uma diagonal extra de armazenamento, isto pode ser vantajoso para esta forma de fatoração ILU(0).

Esta forma de fatoração incompleta será equivalente a fatoração incompleta obtida com o algoritmo 4.6, quando o produto das partes estritamente inferior e superior de A, for constituída somente por elementos diagonais e de preenchimento. A matriz D pode ser definida por uma recursão de forma que a diagonal do produto de matrizes (4.14), seja igual à diagonal de A. As matrizes L e U, juntas, em ILU(0), devem ter o mesmo número de elementos não nulos que a matriz original A.

4.5.2 Nível de preenchimento e ILU(p)

Fatorações incompletas que diferem um pouco de ILU(0), por admitirem alguns preenchimentos em posições nulas, podem ser preferidas por serem mais eficientes e mais confiáveis por se aproximarem mais de A, ou por produzirem uma taxa de convergência mais adequada. Uma destas seria a fatoração ILU(1) (Saad [40]), que conserva a "primeira ordem de preenchimentos".

Para ilustrar ILU(p), usando o exemplo da figura 4.1, a fatoração ILU(1) será calculada tomando o padrão de zeros P como sendo do produto LU, onde L e U são obtidos a partir de ILU(0). Este padrão de zeros é mostrado na parte inferior direita da figura 4.1. Os fatores L_1 e U_1 na fatoração ILU(1) são obtidos executando a fatoração ILU(0) na matriz com "padrão aumentado", sendo este padrão chamado $NZ_1(A)$. As posições de preenchimento criadas neste produto pertencem ao padrão "aumentado" $NZ_1(A)$, porém com valores reais nulos. O novo padrão da matriz A é mostrado na figura 4.2, onde os padrões de L_1 e U_1 são ilustrados na parte superior da figura 4.2. Esta nova matriz LU da figura 4.2 tem adicionadas duas diagonais (uma na parte inferior e outra na parte superior).

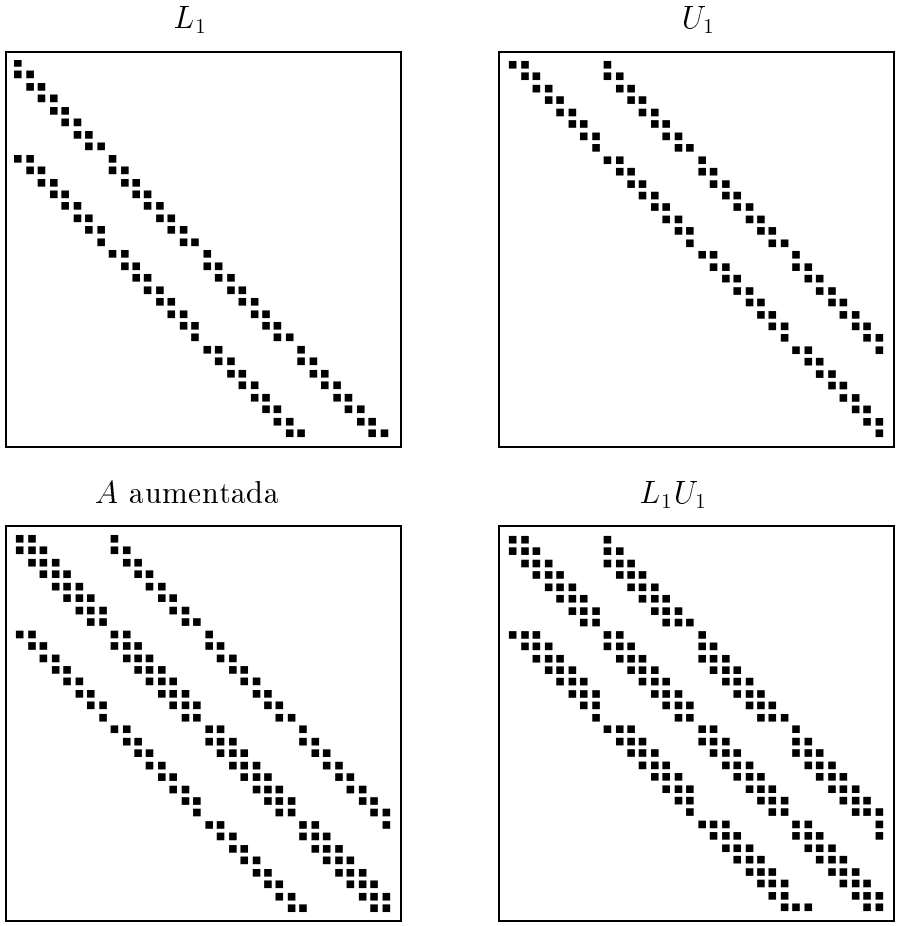

Figura 4.2: *Fatoração ILU(1)*

A construção definida na ilustração 4.2 não se estende para matrizes esparsas em geral, mas este método pode ser generalizado introduzindo o conceito de "nível de preenchimento".

A cada elemento que é processado através da eliminação Gaussiana é atribuído um nível de preenchimento e os descartes são baseadas no valor do nível de preenchimento. Qualquer forma de eliminação Gaussiana pode ser usada. O algoritmo 4.5 é um dos exemplos usados como modelo. O nível de preenchimento indica o tamanho dos elementos: quanto maior o nível, menor a magnitude dos elementos. Como exemplo, Saad [40][p.279], emprega um tamanho ϵ^k é que é atribuído a qualquer elemento cujo nível de preenchimento é k, onde $\epsilon < 1$. Um elemento não-nulo inicia com um nível de preenchimento 1 e um elemento zero tem nível de preenchimento ∞.

No algoritmo 4.5, um elemento a_{ij} na linha 5, é atualizado pela fórmula

$$a_{ij} = a_{ij} - a_{ik} \times a_{kj}. \tag{4.15}$$

Considerando lev_{ij} como o nível atual do elemento a_{ij}, o modelo desenvolvido por Saad [40] diz que o tamanho do elemento atualizado deve ser

$$a_{ij} := \epsilon^{lev_{ij}} - \epsilon^{lev_{ik}} \times \epsilon^{lev_{kj}} = \epsilon^{lev_{ij}} - \epsilon^{lev_{ik}+lev_{kj}}. \tag{4.16}$$

O tamanho de a_{ij}, será o máximo dos dois tamanhos $\epsilon^{lev_{ij}}$ e $\epsilon^{lev_{ik}+lev_{kj}}$, e assim podemos definir o novo nível de preenchimento como,

$$lev_{ij} = min\{lev_{ij}, lev_{ik} + lev_{kj}\}.$$

Na bibliografia especializada, usualmente os níveis de preenchimento são transladados por -1 e assim temos a definição.

Definição: 4.1. O nível inicial de preenchimento de um elemento a_{ij} na matriz esparsa A é definido por

$$lev_{ij} = \begin{cases} 0 & se \;\; a_{ij} \neq 0 \;\; ou \; i = j \\ \infty & em \;\; outro \;\; caso \end{cases}$$

A cada modificação do elemento a_{ij} (linhas do algoritmo) o nível de preenchimento é atualizado por

$$lev_{ij} = min\{lev_{ij}, lev_{ik} + lev_{kj} + 1\}. \tag{4.17}$$

Se $a_{ij} \neq 0$ na matriz original A, então o elemento na posição i, j deverá ter nível de preenchimento igual a zero em todo o processo de eliminação. Esta observação se deve ao fato de que o nível de preenchimento de um elemento nunca aumenta durante a eliminação.

Em ILU(p), todos os elementos cujo nível de preenchimento não excedem p podem ser mantidos. Pela definição anterior, o padrão de zeros para ILU(p) é dado por

$$P_P = \{(i,j)/lev_{ij} > p\}$$

onde lev_{ij} representa o valor do nível de preenchimento após feitas todas as atualizações. O caso em que $p = 0$ representa a fatoração ILU(0).

Para implementar a fatoração ILU(p) é comum separar a fase simbólica, onde as estruturas dos fatores L e U são determinados, da fase numérica, onde os valores numéricos são calculados. Embora em alguns casos, como descrito aqui, a variante descrita não separa estas duas fases.

Algoritmo 4.7 *Fatoração ILU(p)*

1. Para $a_{ij} \neq 0$, *defina* $lev(a_{ij}) = 0$
2. Para $i = 2, ..., n$,
3. *Para cada* $k = 1, ..., i-1$ *e para* $lev(a_{ik}) \leq p$
4. *calcule* $a_{ik} = a_{ik}/a_{kk}$
5. *calcule* $a_{i*} = a_{i*} - a_{ik}a_{k*}$
6. *Atualize o nível de preenchimento dos elementos* $a_{ij} \neq 0$ *, usando 4.17*
7. *Fim*
8. *troque cada elemento da linha i que tenha* $lev(a_{ij}) > p$ *por zero*
9. Fim

Saad [40] apresenta algumas desvantagens do algoritmo 4.7: em relação a não previsibilidade da quantidade de preenchimentos e do trabalho computacional para obter a fatoração ILU(p), e ao elevado custo para calcular o nível de preenchimento. Assim, é possível que o algoritmo descarte (ou substitua por zero) elementos de magnitude significativa e resulte numa fatoração incompleta inexata, no sentido que $R = LU - A$ não seja pequeno. Algumas técnicas que serão descritas mais adiante, procuram eliminar essas dificuldades, produzindo uma fatoração incompleta com pequeno erro R e controlando o número de preenchimentos.

4.5.3 Fatoração incompleta modificada LU (MILU)

Em relação ao que foi estudado até aqui, todos os elementos eliminados durante o processo, foram simplesmente descartados. Algumas técnicas tentam reduzir o efeito destes "descartes", *compensando* as entradas descartadas. Saad, [40] cita como exemplo, uma estratégia popular que consiste em somar os elementos abandonados ao final do k-ésimo laço do algoritmo 4.5 e a seguir subtrair esta soma da entrada da diagonal correspondente em U. Esta estratégia de *compensação diagonal* representa a fatoração ILU modificada (MILU).

Assim, na equação (4.11), a linha final u_{i*}, obtida após completar o k-ésimo laço do algoritmo 4.5, passa a ter mais uma modificação, ou seja,

$$u_{ii} := u_{ii} - (r_{i*}e), \tag{4.18}$$

onde r_{i*} é definida pela equação (4.12) e $e = (1, 1, ..., 1)^T$. Observe que r_{i*} é uma linha, $r_{i*}e$ é a soma dos elementos desta linha. A equação acima pode ser reescrita na forma de linha como $u_{i*} = u_{i*} - (r_{i*}e)e_i^T$ e a equação (4.13) fica

$$a_{i*} = \sum_{k=1}^{i-1} l_{ik}u_{k*} + (r_{i*}e)e_i^T - r_{i*}. \tag{4.19}$$

Observe que

$$a_{i*}e = \sum_{k=1}^{i-1} l_{ik}u_{k*}e + (r_{i*}e)e_i^T e - r_{i*}e = \sum_{k=1}^{i-1} l_{ik}u_{k*}e = LUe. \tag{4.20}$$

Como resultado, esta estratégia garante que a soma dos elementos de cada linha de A é igual a soma correspondente de LU e assim $Ae = LUe$. Para alguns problemas esta estratégia apresenta bons resultados, porém em outros, o algoritmo MILU, em geral, não apresenta melhores resultados que ILU.

4.5.4 Método ponderado de modificação de Eijkhout

O método de Eijkhout [18], procura conservar as seguintes propriedades da matriz ao fazer uma fatoração incompleta:

1. Alguma simetria da matriz original é preservada. Note que isto não é trivial no caso de entradas descartadas, como pode ser observado na equação (4.4).

2. A fatoração deveria ser bem-definida no sentido que, determinada matriz com diagonal positiva deveria ter somente pivôs positivos. Note que esta é uma condição mais forte, que pivôs deveriam ser positivos para matrizes que são positivas definidas. Isto, com certeza, faz o método ser potencialmente satisfatório para certos sistemas indefinidos.

3. O espectro da matriz não será muito alterado, em particular, a estratégia de descartes deveria reduzi-lo, ao menos para alguma versão de fatoração modificada (MILU) em M-matrizes.

4. Aplicando a fatoração em uma M-matriz deve resultar uma M-matriz.

Estas modificações são completadas eliminando os elementos de preenchimentos fora da diagonal nas posições (i, j) e (j, i), ao mesmo tempo combinando as somas deles com os elementos (i, i) e (j, j) da diagonal.

$$a_{ii} \leftarrow a_{ii} - a_{ij}\sqrt{a_{ii}/(a_{jj}/2)}, \quad a_{jj} \leftarrow a_{jj} - a_{ji}\sqrt{a_{jj}/(a_{ii}/2)}. \tag{4.21}$$

Esta idéia básica é melhorada por alguns métodos e condições mais dinâmicas para omitir as modificações em certos casos. Para maiores detalhes ver [17]

4.5.5 A aproximação da fatoração incompleta LU (ILUT)

As fatorações incompletas vistas até agora, por basearem-se principalmente nos níveis de preenchimento, não levando em conta muitas vezes o valor numérico dos elementos, podem perder um pouco da precisão nos resultados. Esta característica, porém, pode não ser adequada a muitos problemas práticos e por este motivo alguns métodos alternativos baseados na eliminação Gaussiana, procuram durante o processo de eliminação, considerar a magnitude dos elementos e não somente a sua posição.

Uma destas sugestões talvez um pouco semelhante à ILU(0), porém ignorando os elementos "pequenos" é o que Saad [39] denomina aproximação da fatoração incompleta LU ou ILUT (algoritmo 4.8). Esta, a partir da versão IKJ de eliminação Gaussiana (algoritmo 4.5), aplica algumas regras para decidir quando os elementos são trocados por zeros e não realmente atualizados. No algoritmo citado, w é um vetor usado para acumular combinações lineares de linhas esparsas na eliminação, e w_k é a k-ésima entrada deste vetor. Como antes, a_{i*} denota a i-ésima linha de A.

Algoritmo 4.8 *ILUT*

1. Para $i = 1, ..., n$
2. $w = a_{i*}$
3. *Para* $k = 1, ..., i-1$ *e quando* $w_k \neq 0$
4. $w_k = w_k / a_{kk}$
5. *Aplica a regra de descarte a* w_k
6. *Se* $w_k \neq 0$, *então*
7. $w = w - w_k u_{k*}$
8. *Fim*
9. *Fim*
10. *Aplica a regra de descarte na linha* w
11. $l_{ij} = w_j$ *para* $j = 1, ..., i-1$
12. $u_{ij} = w_j$ *para* $j = 1, .., n$
13. $w = 0$
14.Fim.

No algoritmo 4.8 observamos que a linha 7 é uma operação de atualização esparsa. Comumente usa-se um vetor cheio para w, e associa-se um ponteiro na direção dos elementos não nulos. As linhas 11 e 12 são operações de cópia de vetores esparsos. Ao final de cada laço i, o vetor w é preenchido com alguns elementos não nulos. Por esse motivo, se faz necessário anular tais elementos ao final da eliminação Gaussiana, o que é feito na linha 13.

ILU(0) pode ser visto como um caso particular do algoritmo 4.8, onde o critério de eliminação se refere a descartar elementos que estão em posições não pertencentes à estrutura original da matriz.

Na fatoração ILUT(p, τ), é usada a seguinte regra:

1. Na linha 5, um elemento w_k é descartado, ou seja, substituído por zero, se este for menor que uma tolerância relativa τ_i, obtida pela multiplicação de τ pela norma original da i-ésima linha (e.g., norma-2).

2. Na linha 10, um tipo de regra de eliminação diferente é aplicada. Primeiro, novamente eliminam-se todos os elementos da linha com magnitude menor que a tolerância relativa τ_i. Então, mantém-se só os p

maiores elementos na parte L da linha e os p maiores na parte U da linha, mais o elemento da diagonal, que é sempre mantido.

No segundo passo procura-se controlar o número de elementos por linha. De maneira geral, p pode ser visto como um parâmetro usado para controlar o uso de memória, enquanto τ é usado para reduzir o custo computacional.

4.5.5.1 Análise do método

Os teoremas da existência da fatoração ILUT são semelhantes aos das outras fatorações incompletas. Se a matriz original é diagonalmente dominante, então as matrizes transformadas (geradas durante a eliminação), também serão diagonalmente dominantes.

O vetor linha w resultante da linha 4 do algoritmo 4.8, será representado por u_{i*}^{k+1}. Observe que para $j \leq k$, teremos $u_{ij}^{k+1} = 0$. As linhas 3 a 10 no algoritmo, envolvem uma seqüência de operações da forma

1. $l_{ik} = u_{ik}^{k}/u_{kk}$
2. se $|l_{ik}|$ é suficientemente pequeno, faça $l_{ik} = 0$
3. caso contrário faça $u_{ij}^{k+1} = u_{ij}^{k} - l_{ik}u_{kj} - r_{ij}^{k}, \quad j = k+1, ..., n,$

para $k = 1, ..., i-1$, no qual inicialmente, $u_{i*}^{1} = a_{i*}$, e onde r_{ij}^{k} é um elemento subtraído de um elemento de preenchimento que está sendo descartado. Este r_{ij} é igual a zero quando u_{ij}^{k} não é descartado, ou igual a $u_{ij}^{k} - l_{ik}u_{kj}$ quando o elemento u_{ij}^{k+1} está sendo descartado (transformado em zero). Ao término do i-ésimo passo de eliminação Gaussiana (laço externo do algoritmo 4.8), obtemos a i-ésima linha de U,

$$u_{i*} \equiv u_{i-1,*}^{i} \tag{4.22}$$

e a seguinte relação é satisfeita:

$$a_{i*} = \sum_{k=1}^{i} l_{kj} u_{i*}^{k} + r_{i*}, \tag{4.23}$$

onde r_{i*} é a linha que contém todos os preenchimentos.

Mais detalhes sobre a estratégia básica da implementação da fatoração ILUT(p, τ), bem como algumas modificações, podem ser encontradas em Saad [40][p. 290]

Uma implementação inadequada da fatoração ILUT pode conduzir a uma etapa de fatoração cara e possivelmente a um algoritmo impraticável. Entre as dificuldades que podem causar ineficiências na implementação da fatoração ILUT podemos citar.

1. Geração das combinações lineares das linhas de A (linha 7 do algoritmo 4.8).
2. Seleção dos p maiores elementos em L e U.
3. Necessidade de acesso a elementos de L em ordem crescente de colunas (na linha 3 do algoritmo 4.8).

Para resolver 1, uma alternativa seria gerar uma linha completa, acumulando nesta a combinação linear das linhas anteriores. A linha é zerada de novo, depois que o laço é finalizado. A implementação desta técnica assim como a solução dos problemas 2 e 3, e forma de armazenamento dos dados podem ser encontrados em Saad [40][p 292].

4.6 A abordagem ILUTP

A abordagem ILUT pode falhar para muitas matrizes que aparecem em alguns problemas de aplicações, por uma das seguintes razões.

1. O procedimento ILUT encontra um pivô zero;
2. O procedimento ILUT encontra uma condição de *overflow* ou *underflow*, devido ao crescimento exponencial dos elementos dos fatores;
3. O pré-condicionador ILUT termina normalmente, mas o pré-condicionador da fatoração incompleta calculado é instável.

Uma fatoração ILU instável é aquela onde $M^{-1} = U^{-1}L^{-1}$, tem uma norma muito grande, ocasionando convergência lenta ou divergência da iteração externa. O caso 1 pode ser contornado de certo modo através da atribuição de um valor arbitrário não zero, quando um elemento zero é encontrado na diagonal. Esta solução pode não ser satisfatória pois o pré-condicionador pode não ser tão eficiente quanto deveria. Nestes casos pode-se usar pivotamento, embora esta forma de fatoração possa levar a um algoritmo com custo e complexidade similar a ILUT. O pivotamento por linhas não é muito prático, devido à estrutura de dados usada em ILUT, porém pode-se facilitar a implementação com o pivotamento por colunas.

A fatoração denominada ILUTP, onde "P" significando pivotamento, usa um vetor de permutação *perm* que serve para armazenar a nova ordenação das variáveis, junto com o vetor de permutação reverso. No passo i do processo de eliminação, seleciona-se o maior elemento da linha que é definido como a i-ésima nova variável. Atualizam-se então os dois vetores de permutação. Os elementos da matriz L e U são armazenados na sua forma original. Contudo, quando expandimos a linha de L e U que corresponde ao i-ésimo passo externo da eliminação Gaussiana, os elementos são armazenados segundo seus novos rótulos, usando o vetor *perm* para a translação.

A primeira opção ao fim do processo é deixar todos os elementos rotulados de acordo com a rotulação original, não sendo necessário nenhum trabalho adicional devido as variáveis já estarem nesta forma no algoritmo, mas as variáveis devem ser permutadas a cada passo de pré-condicionamento. A segunda opção é aplicar a permutação em todos os elementos de A assim como a L/U. A permutação

não é feita a cada passo, mas esta deve ser permutada de volta ao término da fase de iteração.

A complexidade do procedimento ILUTP é virtualmente idêntica a ILUT, porém, podem ser usadas algumas variações adicionais. Um parâmetro de tolerância chamado *permtol* pode ser incluído para auxiliar a determinar quando permutar ou não as variáveis. Um elemento a_{ij} fora da diagonal deve ser permutado somente quando $tol \times |a_{ij}| > |a_{ii}|$. Além disso, o pivotamento pode se restringir a um bloco da diagonal *mbloc* de tamanho fixo. Um valor de $mbloc \geq n$ indica que não há restrições ao pivotamento.

Para matrizes difíceis, algumas estratégias são sugeridas:

1. Sempre aplica-se uma modificação em todas as linhas (ou colunas) por exemplo, de forma que suas norma-1 sejam todas iguais a 1; então aplica-se uma modificação nas colunas (ou linhas).

2. Usa-se uma pequena tolerância de descarte (por exemplo, $\epsilon = 10^{-4}$ ou $\epsilon = 10^{-5}$).

3. Emprega-se um parâmetro de preenchimento grande (por exemplo, $lfil = 20$).

4. Não se emprega um valor pequeno para *permtol*. Valores razoáveis estão entre 0.5 e 0.01, sendo 0.5 o melhor em muitos casos.

5. Usa-se $mbloc = n$ a menos que haja razão para um bloco de tamanho justificável.

4.6.1 Conclusões sobre a fatoração incompleta LU

A maioria dos pré-condicionadores ILU já discutidos, são baseados principalmente na variante IKJ da eliminação Gaussiana. De acordo com a forma de eliminação Gaussiana utilizada, podem ser obtidos diferentes tipos de ILUs.

De acordo com Yogin e Timothy [44], o foco da construção efetiva de um pré-condicionador baseado na aproximação da fatoração incompleta, consiste em encontrar um (ou o mais útil) conjunto de elementos de descartes (ou que são mantidos) na seqüência do algoritmo de fatoração. Foram desenvolvidos várias alternativas para os descartes, cada uma objetivando selecionar o conjunto máximo de entradas a descartar procurando com isso produzir o pré-condicionador mais preciso. Estas alternativas incluem descartes de valores, descartes de posições, descartes baseados em economia de armazenamento, ou alguma forma híbrida das três estratégias prévias.

Ainda segundo Yogin e Timothy [44], a idéia básica da estratégia de descartes baseada nos níveis de preenchimento, é associar esse nível a cada entrada de coeficiente da matriz e atualizar esse nível durante toda a fatoração. A maior dificuldade na construção do pré-condicionador está em controlar, ou prever, a quantidade de memória que será utilizada.

Saad em [40] cita ainda uma outra forma de fatoração ILUT que não explora características de simetria, pois em geral se A é simétrica, nem sempre $M = LU$ é simétrica. Em muitas aplicações, incluindo a dinâmica dos fluidos computacional e engenharia estrutural, as matrizes resultantes são armazenadas num formato (SSK) *skyline* esparso ao invés do formato padrão *Linha Esparsa Comprimida* ("compressed sparse row").

(Nesta estrutura, a matriz A é decomposta em

$$A = D + L_1 + L_2^T,$$

na qual D é a diagonal de A e L_1 e L_2 são matrizes estritamente triangulares inferiores. Então usa-se uma representação esparsa de L_1 e L_2 na qual, tipicamente, L_1 e L_2 são armazenados no formato CSR e D é armazenado separadamente. Para matrizes neste formato, pode-se no entanto desenvolver algumas técnicas de fatoração incompleta sem ter que convertê-las para o formato CSR. Duas notáveis vantagens desta abordagem são

1. a economia no armazenamento para matrizes com estrutura simétrica,

2. o fato que o algoritmo gera um pré-condicionador simétrico quando a matriz original é simétrica.

Como exemplo vamos considerar a seqüência de matrizes

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

onde $A_n = A$. Se A_k é não singular e sua fatoração LDU

$$A_k = L_k D_k U_k$$

é calculada, então a fatoração LDU de A_{k+1} é dada por

$$A_{k+1} = \begin{pmatrix} L_k & 0 \\ y_k & 1 \end{pmatrix} \begin{pmatrix} D_k & 0 \\ 0 & d_{k+1} \end{pmatrix} \begin{pmatrix} U_k & z_k \\ 0 & 1 \end{pmatrix},$$

onde

$$z_k = D_k^{-1} L_k^{-1} v_k \tag{4.24}$$

$$y_k = w_k U_k^{-1} D_k^{-1} \tag{4.25}$$

$$d_{k+1} = \alpha_{k+1} - y_k D_k z_k. \tag{4.26}$$

Desta forma, resolvendo dois sistemas triangulares inferiores unitários e calculando um produto interno escalonado, obtemos os últimos pares linha/coluna da fatoração. Para utilizar esta técnica em matrizes esparsas, observando esta esparsidade e os vetores envolvidos, os pares linhas/colunas w_k, v^T são armazenados em seqüência, como uma única linha no modo esparso. Os elementos da diagonal são armazenados separadamente. Este formato é chamado *Unsymmetric Sparse Skyline* (USS). Cada passo da fatoração ILU baseado nesta técnica consiste na solução de dois sistemas lineares esparsos e num produto interno também esparso.

Com o objetivo de resolver o sistema de forma econômica, uma escolha é resolver os sistemas triangulares (4.24) e (4.25) exatamente, utilizando uma estratégia numérica de eliminação para eliminar os termos pequenos no final. No

entanto, o custo total da fatoração ILU com esta estratégia é da ordem de $O(n^2)$ operações, no mínimo, o que não representa uma alternativa aceitável para problemas muito grandes. Como o que se quer é apenas uma solução aproximada, Saad sugere usar a série truncada de Neumann,

$$z_k = D_k^{-1} L_k^{-1} v_k = D_k^{-1}(I + E_k + E_k^2 + ... + E_k^p)v_k, \tag{4.27}$$

onde $E_k \equiv I - L_k$. De fato, segundo Saad [40], por analogia com ILU(p), nota-se que as potências de E_k tendem a ficar menores com o crescimento de p. Detalhando o exame da estrutura de $E_k^p v_k$, percebe-se uma semelhança entre esta técnica e ILU(p) para matrizes simétricas. Outra importante observação, é que o vetor $E_k^j v_k$ pode ser calculado no modo esparso-esparso, i.é., em termos de operações que envolvem produtos de matrizes esparsas por vetores esparsos. Sem isto, o custo total continuaria sendo $O(n^2)$.

O procedimento GMRES(c) pode ser usado na solução dos sistemas triangulares em modo esparso-esparso. Técnicas similares são mostradas em Saad [38].

4.7 Fatoração incompleta de Cholesky

A fatoração incompleta de Cholesky (IC) segundo Wang [43], é um método amplamente conhecido e efetivo para acelerar a convergência dos métodos iterativos, na solução de sistemas lineares simétricos positivos definidos (entre eles o método do Gradiente Conjugado). Uma análise mais detalhada da fatoração incompleta de Cholesky, aplicada ao método do Gradiente Conjugado, pode ser encontrada em Kershaw [27].

O pré-condicionador obtido na fatoração incompleta de Cholesky é um dos pré-condicionadores mais importantes e comumente usados em métodos iterativos para resolver sistemas lineares grandes e esparsos. O problema principal encontrado neste tipo de fatoração, que diminui sua eficiência, vem do aparecimento de pivôs não positivos. Para superar tais falhas, Wang, Bramley e Gallivan [42],

sugerem alguns métodos modificados. Tais métodos usam as informações sobre os elementos eliminados e alteram o processo de fatoração para garantir a existência do fator triangular R.

Em relação aos pivôs não positivos, Wang, Bramley e Gallivan [43], sugerem algumas estratégias que são divididas em duas classes: estratégia numérica e estratégia estrutural. A estratégia numérica, usa valores numéricos gerados durante o processo de fatoração para modificar a fatoração. A estratégia estrutural, cujo trabalho é atribuído a Coleman [8], seleciona um padrão de esparsidade para garantir a finalização do processo de fatoração incompleta de Cholesky (IC).

Em aplicações onde cada matriz de coeficientes tem o mesmo padrão de elementos não nulos e servem para resolver uma seqüência de sistemas lineares, estas estratégias estruturais se fazem especialmente importantes. Isto pode ocorrer na solução de problemas de programação linear, através do uso de métodos de pontos interiores, ou na solução de equações diferenciais parciais não lineares discretizadas em uma malha fixa [43]. Embora os valores possam variar a cada passo, o padrão de esparsidade é fixo e este é usado para obter o pré-condicionador da fatoração incompleta de Cholesky, desde que a fatoração exista e satisfaça certas condições estruturais.

Alguns algoritmos específicos que modificam o padrão de esparsidade foram propostos e testados por Wang, Gallivan e Bramley em 1993 e 1995 em [42][43]. Como as matrizes são reais e simétricas positivas definidas, apenas a parte triangular superior das matrizes precisa ser considerada, já que as partes triangulares inferiores são especificadas por simetria. Consideramos $P_n = \{(i,j)|\, 1 \leq i \leq j \leq n\}$ como o conjunto de todas as possíveis posições não nulas na matriz triangular superior $n \times n$. O padrão de esparsidade $P(A)$ da matriz simétrica A é definido como

$$P(A) = \{(i,j)|\, a_{ij} = a_{ji} \neq 0,\ 1 \leq i \leq j \leq n\}.$$

É importante observar que $P(A)$ apenas considera as posições de elementos não nulos de A na parte triangular superior e $P(A) \subseteq P_n$. Seja U o fator de Cholesky

triangular superior de A, formado não assumindo nenhum cancelamento, de forma que $A = UU^T$. No que segue assumimos que $P(U)$ denota o padrão de esparsidade de U e $P(U) \subseteq P_n$.

Na fatoração incompleta de Cholesky da matriz A, usando o padrão de esparsidade P, descartam-se imediatamente todas as entradas que ocorrem fora do padrão de esparsidade especificado, e não se faz nenhuma outra modificação numérica na matriz. Uma condição imposta a todos os padrões P, para que a fatoração IC seja completada, é que todas as posições (i, i) da diagonal estejam em P.

O algoritmo (4.9), descreve tal fatoração IC, segundo Wang, Bramley e Gallivan em [43].

Embora o algoritmo geralmente seja implementado copiando inicialmente A para R e utilizando a partir daí somente R, a forma mostrada no algoritmo (4.9) é válida para padrões de esparsidade $P \not\subseteq P(A)$. Na prática, as atualizações não seriam executadas em A de qualquer maneira, pois geralmente A precisa ser utilizada no método iterativo que é pré-condicionado por R.

Algoritmo 4.9 *Fatoração Incompleta de Cholesky*
[R]=[A,P]

1. *Para* $k = 1, ..., n$
2. *Se* $a_{kk} > 0$
3. $r_{kk} = \sqrt{a_{kk}}$
4. *Para* $j = k+1, k+2, ..., n$
5. $$r_{kj} = \begin{cases} 0 & (k,j) \notin P \\ a_{kj}/r_{kk} & (k,j) \in P \end{cases}$$
6. *Fim*
7. *Para* $i = k+1, k+2, ..., n$
8. *Para* $j = i, i+1, ..., n$
9. $a_{ij} = a_{ij} - a_{ki}a_{kj}$ *onde* $(i,j) \in P$, $(k,j) \in P$ *e* $(k,i) \in P$
12. *Senão*
13. *Pare (fatoração incompleta falhou)*

4.7.1 Análise do método da fatoração incompleta de Cholesky

Lin e Moré [29] propõem uma fatoração incompleta de Cholesky que combina a fatoração de Jones e Plassman [26] e a fatoração ILUT de Saad [39][40].

Dada uma matriz simétrica A e um padrão de esparsidade S, o fator de Cholesky incompleto de A é uma matriz triangular inferior L, tal que

$$A = LL^T + R, \quad l_{ij} = 0 \ \ se \ \ (i,j) \notin S, \quad r_{ij} = 0 \ \ se \ \ (i,j) \in S. \tag{4.28}$$

Estratégias de preenchimento pré-determinadas fixam o padrão S do fator incompleto antes da fatoração. Meijerink e Van der Vorst [32][33], consideram duas escolhas de S; a primeira como o padrão de esparsidade de A e uma outra que permita mais preenchimentos. Pode-se, também definir S de forma que L seja uma matriz banda, com largura de banda pré-determinada. Estas estratégias requerem uma quantidade de memória previsível, mas são independentes das entradas de A, pois os elementos a serem eliminados dependem unicamente da estrutura determinada.

Um outro tipo de estratégia para descartar elementos é aquele que irá manter elementos não nulos se eles forem maiores que determinado parâmetro fixado. Por exemplo, Munksgaard [34] descarta o elemento $a_{ij}^{(k)}$ durante o k-ésimo passo de eliminação se

$$|a_{ij}^{(k)}| \leq \tau \sqrt{a_{ii}^{(k)} a_{jj}^{(k)}}, \tag{4.29}$$

onde τ é a tolerância estabelecida. Uma desvantagem deste tipo de estratégia, é o fato de não podermos prever a quantidade de memória, pois isto depende de τ. Em geral não é possível determinar o valor de τ de forma que a quantidade de memória a ser usada para armazenar L esteja dentro de limites especificados. Se o valor de τ for grande, então L terá poucos elementos não nulos (e portanto a exigência de memória será pequena), mas também poderá tender a ser um pré-condicionador com pouca eficiência.

Jones e Plassmann [26], em relação à fatoração incompleta de Cholesky, propuseram uma fatoração que procura reter apenas os n_k-ésimos maiores elemen-

tos na parte estritamente triangular inferior da k-ésima coluna de L, onde n_k é o número de elementos na k-ésima coluna da parte estritamente triangular inferior de A.

Em relação ao problema de previsão de memória, uma estratégia similar á citada anteriormente é a fatoração ILUT de Saad [39][40], que tem quantidade de armazenamento previsível e depende dos elementos de A. A fatoração $ILUT$ de uma matriz A qualquer, depende de um parâmetro p relacionado à quantidade de memória e de uma tolerância τ. Esta tolerância é usada para descartar todos os elementos em L e U, menores que τ_k, onde τ_k é definido como o produto de τ pela norma-l_2 da k-ésima linha de A. A fatoração ILUT mantém os p maiores elementos, em magnitude, de cada linha de L e U. No entanto, a fatoração ILUT ignora qualquer simetria na matriz A. Mesmo que A seja simétrica, o padrão de esparsidade de L e U^T são diferentes. Desta forma, o produto LU produzido pela ILUT, não será em geral simétrico para uma matriz simétrica A.

Chih Lin e Moré em [29] apresentam outra implementação da fatoração incompleta de Cholesky, baseada na versão jki da fatoração de Cholesky que é mostrada no algoritmo (4.10). Esta fatoração, na posição da j-ésima coluna de L, reescreve a j-ésima coluna de A. Observa-se que os elementos da diagonal são atualizados à medida que a fatoração é efetuada.

Algoritmo 4.10 *Fatoração de Cholesky*

1 *Para* $j = 1 : n$
2. $a_{jj} = \sqrt{(a_{jj})}$
3. *Para* $k = 1 : j - 1$
4. *Para* $i = j + 1 : n$
5. $a_{ij} = a_{ij} - a_{ik}a_{jk}$
6. *Fim*
7. *Fim*
8. *Para* $i = j + 1 : n$
9. $a_{ij} = a_{ij}/a_{jj}$
10. $a_{ii} = a_{ii} - a_{ij}^2$

Como não objetivamos discutir técnicas de fatoração de Cholesky para matrizes densas, um estudo mais abrangente pode ser encontrado em [30].

O algoritmo 4.11 apresenta a fatoração incompleta de Cholesky com limitação de memória. Neste, ao iniciar a j-ésima iteração, foram calculadas as colunas $l_1, ..., l_{j-1}$ da matriz triangular inferior L e armazenado o i-ésimo componente de l_k em $a(i,k)$ para $i \geq k$. Os elementos a_{ii}^j da diagonal, para $i \geq j$, de

$$A_j = A - \sum_{k=1}^{j-1} l_k l_k^T \tag{4.30}$$

também foram calculados e armazenados em $a(i,i)$, para $i \geq j$. Calculamos l_j combinando elementos em

$$A_j = l_j l_j^T + \sum_{k=j+1}^{n} l_k l_k^T. \tag{4.31}$$

Como $l_{ik} = 0, \quad \text{para} \quad i < k$, esta definição de l_j implica em

$$l_{jj}^2 = a_{jj}^{(j)}, \quad l_{jj} l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}, \quad i > j.$$

O algoritmo 4.11 calcula l_{jj} e l_{ij} a partir destas expressões e retém os $n_j + p$ maiores elementos em l_j, onde n_j é o número de elementos não nulos na parte estritamente triangular inferior da matriz original A. Os elementos da diagonal de A_{j+1} podem ser determinados por

$$a_{ii}^{(j+1)} = a_{ii}^{(j)} - l_{ij}^2, \quad i > j.$$

O algoritmo (4.11), só retém elementos fora da diagonal se estes estiverem entre os maiores elementos da coluna atual. Os elementos diagonais sempre são retidos. Particularmente, isto implica em que o padrão de esparsidade do fator incompleto talvez não inclua o padrão de esparsidade da matriz original A.

A implementação sugerida por Chih Lin e Moré[29], e que segue a implementação de Jones e Plassman [26], tem como principais detalhes, as estruturas de dados necessárias para atualizar a_{ij} através de $a_{ij} - a_{ik} a_{jk}$ por operações esparsas, que só operam sobre elementos não nulos, e o algoritmo usado para selecionar os maiores elementos da coluna

Algoritmo 4.11 *Fatoração Incompleta de Cholesky*

1. $Para \quad j = 1, ..., n$
2. $\quad a_{jj} = sqrt(a_{jj})$
3. $\quad col - len = tamanho(i > j : \ a_{ij} \neq 0)$
4. $\quad Para \quad k = 1, ..., j-1 \quad e \quad a_{j,k} \neq 0$
3. $\quad\quad Para \quad i = j+1, ..., n \quad e \quad a_{i,k} \neq 0$
4. $\quad\quad\quad a_{ij} = a_{ij} - a_{ik}a_{jk}$
5. $\quad\quad Fim$
6. $\quad Fim$
7. $\quad Para \quad i = j+1, ..., n \quad e \quad a_{ij} \neq 0$
8. $\quad\quad a_{ij} = a_{ij}/a_{jj}$
9. $\quad\quad a_{ii} = a_{ii} - a_{ij}^2$
10. $\quad Fim$
11. $\quad$ *Retenha os maiores col-len + p elementos em* $a(j+1 : n, j)$.
12. Fim

Na apresentação dada, ignoramos a função representada pelo ordenamento da matriz. É importante observar que o ordenamento da matriz afeta o preenchimento da mesma, e conseqüentemente a fatoração incompleta de Cholesky. Sobre esta questão, Duff e Meurant [15] e Eijkhout [16] observaram que o número de iterações no método do Gradiente Conjugado pode dobrar se o "grau de ordenamento mínimo" é usado para reordenar a matriz. Deve-se considerar que estas condições não foram estabelecidas para pré-condicionadores com limitação de memória e por isto não se pode dizer que valem para os mesmos.

Em algumas considerações sobre o algoritmo 4.11, observa-se que a performance da fatoração incompleta de Cholesky, depende da estratégia usada para escalonar a matriz, pois isto afeta a escolha dos maiores elementos retidos durante a fatoração. Ainda, se houver necessidade de controlar a ocorrência de matrizes positivas definidas em geral, ou matrizes indefinidas que invariavelmente surgem em aplicações de otimização, o algoritmo (4.11) pode ser modificado.

Uma alternativa de contornar o problema é aumentar o tamanho dos elementos da diagonal, até se obter uma fatoração satisfatória. Uma estratégia co-

mum é aumentar qualquer pivô não positivo para um limite positivo, e prosseguir com a fatoração. Porém, esta estratégia deve ser utilizada com cuidado. Por exemplo, considere a matriz

$$A = \begin{bmatrix} 1 & \gamma_1 & 0 \\ \gamma_1 & 1 & \gamma_2 \\ 0 & \gamma_2 & 1 \end{bmatrix}, \quad \gamma_1, \ \gamma_2 \in (0,1).$$

Os cálculos mostram que depois dos primeiros dois passos do algoritmo de Cholesky, obtém-se a matriz triangular inferior

$$A = \begin{bmatrix} 1 & & \\ \gamma_1 & \delta_1 & \\ 0 & \delta_2 & 1-\delta_2^2 \end{bmatrix}, \quad \delta_1 = \sqrt{1-\gamma_1^2}, \ \ \delta_2 = \frac{\gamma_2}{\delta_1}.$$

Assim, para calcular a fatoração de Cholesky, seria preciso somar pelo menos $\delta_2^2 - 1$ ao elemento diagonal (3,3), uma perturbação que não é limitada quando γ_1 aproxima-se de 1. Por outro lado, a fatoração de Cholesky de $A + \alpha I$ tem bons resultados para uma pertubação pequena $\alpha > 0$. Assim, quando γ_1 aproxima-se de 1, a matriz A aproxima-se de

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & \gamma_2 \\ 0 & \gamma_2 & 1 \end{bmatrix},$$

que tem um autovalor unitário, e para $\gamma_2 \in [0;0,1]$, dois autovalores próximos a 2 e $-\frac{1}{2}\gamma_2^2$. Assim a fatoração de Cholesky de $A + \alpha I$, onde $\alpha > \gamma_2^2$, pode ser efetuada.

O exemplo anterior, mostra claramente que precisamos modificar os elementos da diagonal, para evitar encontrar pivôs negativos. Esta observação conduziu a várias propostas de modificação na fatoração de Cholesky da forma $A + E$, onde E é uma matriz diagonal. Estas aproximações são aplicáveis a matrizes indefinidas em geral, mas somente quando são retidos todos os elementos na fatoração. Assim, novamente, perde-se a vantagem de se ter armazenamento previsível.

Várias modificações foram propostas para a fatoração incompleta de Cholesky aplicável a matrizes positivas definidas em geral. A fatoração de Cholesky

incompleta citada por Manteuffel [22], para a matriz modificada

$$\hat{A} = D^{-1/2} A D^{-1/2}, \quad D = diag(a_{ii}),$$

requer o cálculo de um $\alpha \geq 0$ adequado, para o qual a fatoração de Cholesky incompleta de $\hat{A}+\alpha I$ é bem sucedida. Manteuffell usou uma fatoração com preenchimento fixo e mostrou que se $\hat{A} + \alpha I$ for uma H-matriz, então a fatoração incompleta de Cholesky de $\hat{A} + \alpha I$ será bem sucedida. Porém, não recomendou um procedimento para determinar um α satisfatório. Jennings e Malik [29, p.31] propuseram usar

$$a_{ii}^{(k)} = a_{ii}^{(k)} + \sigma |a_{ij}^{(k)}|, \quad a_{jj}^{(k)} = a_{jj}^{(k)} + \frac{1}{\sigma} |a_{ij}^{(k)}|,$$

se $a_{ij}^{(k)}$ é eliminado durante o k-ésimo passo e provaram que se a matriz A é simétrica e positiva definida, esta modificação garante que a fatoração incompleta é bem sucedida para algum $\sigma > 0$. Alguns autores sugerem $\sigma = 1$. Outra sugestão seria usar o parâmetro $\omega \in [0, 1]$ com

$$a_{ii}^{(k)} = a_{ii}^{(k)} + \omega |a_{ij}^{(k)}|, \quad a_{jj}^{(k)} = a_{jj}^{(k)} + \omega |a_{ij}^{(k)}|,$$

e usar um processo de procura para determinar um ω apropriado.

Uma estratégia alternativa é proposta por Chih Lin e Moré em [29], no algoritmo 4.12, onde inicialmente a matriz é escalonada em norma-2 pelas colunas de A. Se A possui colunas de zeros, então o algoritmo é aplicado às submatrizes com colunas não zeros.

Jones e Plassman [26], usam uma aproximaçção similar em matrizes positivas definidas, mas com $p = 0$ no algoritmo 4.12. Nesta aproximação a matriz A é escalonada como no algoritmo citado e gera-se um α_F que inicia com $\alpha_0 = 0$, incrementando α_k pela constante (10^{-2}). Se a matriz A possuir elementos diagonais negativos, a matriz resultante pode vir a ser mal-condicionada, e esse motivo leva a sugerir que o processo seja modificado, principalmente quando tratar-se de matrizes indefinidas em geral. Também observa-se que a estratégia de incrementar α_k por uma constante, nem sempre garante a eficiência do processo.

Uma versão mais recente do algoritmo 4.12, que pode ser encontrado em [29, p.31], utiliza $\alpha_0 = \alpha_s$ se $min(a_{ii}) \leq 0$ e $\alpha_s = \frac{1}{2}\|\hat{A}\|_\infty$. Esta estratégia conduz à conclusão em no máximo duas iterações (desde que $\hat{A}_2$ seja diagonal dominante), mas também tende a gerar um α_F grande, e assim uma fatoração incompleta de Cholesky, que pode vir a ser um pré-condicionador não muito eficiente.

Algoritmo 4.12 *Fatoração Incompleta de Cholesky (icfm)*

1. *escolha* $\alpha_s > 0$ *e* $p \geq 0$
2. *Calcule* $\hat{A} = D^{-1/2}AD^{-1/2}$, $D = diag(\|Ae_i\|_2)$
3. *Fixe* $\alpha_0 = 0$, *se* $min(\hat{a}_{ii}) > 0$;
   senão $\alpha_0 = -min(\hat{a}_{ii}) + \alpha_s$.
4. *Para* $k = 0, 1, ...,$
5. *use algoritmo 4.11 em* $\hat{A}_k = \hat{A} + \alpha_k I$;
   se conseguir $\alpha_F = \alpha_k$ *e saia.*
6. *Use* $\alpha_{k+1} = max(2\alpha_k, \alpha_s)$

A escolha de $\alpha_0 = 0$ é certamente razoável se A é positiva definida ou de forma mais geral, se A tem elementos diagonais positivos. Como fazer uma escolha razoável do valor inicial de α_0 não é uma tarefa simples se a matriz A for indefinida, mas esta escolha de α_0 é mais garantida se $\hat{A}_0$ tem elementos diagonais positivos.

A escolha de α_s precisaria estar relacionada ao menor autovalor da submatriz de $\hat{A}$ definida por S, mas esta informação não é fácil de ser avaliada. Lin e Moré usaram $\alpha_s = 10^{-3}$ nos resultados numéricos. Também testaram com $\alpha_s = 10^{-6}$ e obtiveram resultados semelhantes. A principal desvantagem da escolha de um α_s pequeno é que o algoritmo 4.12, possa vir a necessitar um número muito grande de iterações.

Nos testes aplicados por Lin e Moré [29], verificou-se que o tempo computacional para a fatoração incompleta de Cholesky depende de p e do número de iterações requerida pelo algoritmo 4.12. Se as matrizes possuem elementos diagonais positivos, então o número de iterações l, é relacionado diretamente ao valor final α_F, pela relação $\alpha_F = 2^{l-2}\alpha_s$ para $l > 1$. Assim, os resultados mostram que em

muitos casos o número de iterações é pequeno, com um número maior de iterações ocorrendo para $p = 0$.

Lin e Moré [29], afirmam que a fatoração incompleta de Cholesky definida pelo algoritmo 4.11 existe quando A é uma H-matriz. Lembrando que $A \in \mathbb{R}^{n\times n}$ é uma H- matriz se a matriz associada

$$\mathcal{M}(A) = \begin{cases} |a_{ij}| & se \;\; i = j \\ -|a_{ij}| & se \;\; i \neq j \end{cases},$$

é uma M-matriz; ou seja, a inversa de $\mathcal{M}(A)$ é uma matriz não negativa. Nos resultados seguintes, será importante saber que uma matriz $A \in \mathbb{R}^{n\times n}$, com $a_{ij} \leq 0$, para $i \neq j$, é uma M- matriz se e só se, há um $x > 0$ em $\mathbb{R}^n$ tal que $Ax > 0$. Estes resultados mostram, em particular, que uma matriz estritamente diagonal dominante é uma H-matriz.

Meijerink e Van der Vorst [32] provaram que se A é uma M-matriz, então a fatoração incompleta LU (Cholesky) existe para algum padrão de esparsidade pré-determinado S, e Manteuffel [31] estendeu esse resultado para H-matrizes com elementos diagonais positivos. Porém estes resultados não se aplicam ao algoritmo 4.11, pois o padrão de esparsidade S é determinado durante a fatoração.

Para provar a existência da fatoração proposta por Meijerink e Van der Vorst [32], citada acima, observa-se que a cada fase da fatoração incompleta de Cholesky, no algoritmo (4.11), procura-se calcular um vetor v que aparece na forma matricial

$$A = \begin{bmatrix} \alpha & v^T \\ v & B \end{bmatrix}. \tag{4.32}$$

Deletando algumas entradas em v, e executando a decomposição de Cholesky de A, temos

$$\begin{bmatrix} \alpha & v^T \\ v & B \end{bmatrix} = \begin{bmatrix} \alpha^{1/2} \\ \alpha^{-1/2}w \end{bmatrix} \begin{bmatrix} \alpha^{1/2} \\ \alpha^{-1/2}w \end{bmatrix}^T + \begin{bmatrix} 0 & 0 \\ 0 & B - \frac{1}{\alpha}ww^T \end{bmatrix} + E,$$

onde w é o vetor obtido deletando entradas em v, e

$$E = \begin{bmatrix} 0 & (v-w)^T \\ (v-w) & 0 \end{bmatrix}$$

é a matriz erro. Esta descrição da fatoração incompleta de Cholesky concorda com o algoritmo 4.11, se associarmos α com o elemento diagonal a_{jj}, e o vetor v com os elementos calculados no primeiro laço do algoritmo 4.11 e armazenados em $a_{j+1:n,j}$. Esta descrição condiz completamente com a fatoração produzida pelo algoritmo citado, se no laço final de algoritmo fossem calculados os elementos diagonais do complemento de Schur

$$B - \frac{1}{\alpha} ww^T \tag{4.33}$$

a partir dos elementos diagonais de B e os componentes de w. No entanto, o algoritmo usa todos os componentes de v. Conseqüentemente, esta descrição concorda com o algoritmo 4.11 se o complemento de Shur (4.33) é substituído por

$$B - \frac{1}{\alpha}(ww^T + diag\{(v-w)^2\}), \tag{4.34}$$

onde $diag\{(v-w)^2\}$ é a matriz diagonal com entradas $(v_i - w_i)^2$, e a matriz erro E que é substituída por

$$E = \begin{bmatrix} 0 & (v-w)^T \\ (v-w) & 0 \end{bmatrix} + \frac{1}{\alpha} diag\{(v-w)^2\}.$$

Desde que $w_i \in \{0, v_i\}$, os elementos diagonais da equação (4.34) concordam com os elementos diagonais do complemento de Schur da matriz original (4.32)

Ambas as versões do algoritmo 4.11 são interessantes. A versão baseada em (4.33) tem elementos diagonais maiores, diminuindo assim, as chances de obter um pivô negativo quando as outras colunas são processadas. Também se nota que a versão baseada em (4.33) tem uma matriz de erro local menor e só depende da magnitude dos elementos que foram eliminados. A versão baseada em (4.34) é a fatoração incompleta de Cholesky, proposta por Jones e Plassmann.

Os resultados numéricos apresentados por Lin e Moré [30][seção 6], mostram que a versão baseada em (4.33) tem um melhor desempenho.

5 RESULTADOS

5.1 Problemas teste

Para os problemas teste foram utilizadas matrizes extraídas do Matrix Market, da coleção Harwell-Boeing[1]. Cada problema foi denominado segundo sua matriz de coeficientes e foram utilizadas as matrizes: NOS4, NOS5 e NOS7, do conjunto LANPRO[2]; BCSSTK01, BCSSTM01, BCSSTK02, BCSSTM02, do conjunto BCSSTRUC1[3]; BCSSTK22, BCSSTM22, do conjunto BCSSTRUC3[4].

Nos conjuntos BCSSTRUC1 e BCSSTRUC3, as matrizes sempre vêm em pares, sendo que a matriz K, representa a matriz de rigidez e a matriz M representa a matriz de massa para modelamento de estruturas.

A seguir serão apresentadas, brevemente, cada uma das matrizes utilizadas e a tabela 5.1, que descreve suas características.

1. LANPRO: Equações lineares em engenharia estrutural. Lanczos com reortogonalização parcial.
   (a) NOS4: Aproximação de elementos finitos para uma estrutura de vigas.
   (b) NOS5: Relatórios de projetos de construções com torres interligadas. Cada viga é modelada como uma aproximação de diferenças finitas para o operador bi-harmônico com um extremo livre e outro fixo.
   (c) NOS6: Aproximação de diferenças finitas para a equação de Poisson numa região modelada, com condições de contorno.

[1] http://www.math.nist.gov/MatrixMarket/colletions/hb.html
[2] http://www.math.nist.gov/MatrixMarket/data/Harwell-Boeing/lanpro/lanpro.html
[3] http://www.math.nist.gov/MatrixMarket/data/Harwell-Boeing/bcsstruc1/bcsstruc1.html
[4] http://www.math.nist.gov/MatrixMarket/data/Harwell-Boeing/bcsstruc3/bcsstruc3.html

2. BCSSTRUC1: Matrizes de engenharia estrutural BCS (matrizes de autovalores). Estas matrizes representam análise dinâmica em engenharia estrutural. Foram extraídas de vários pacotes de engenharia estrutural, como GT-STRUDL, MSC/NASTRAN e BCS ATLAS.

   Alguns dos problemas coletados demonstram o efeito de técnicas de engenharia estrutural padrão, como condensação estática e formulações " massa agregada".

   (a) BCSSTK01 e BCSSTM01: Problema teste pequeno.

   (b) BCSSTK02 e BCSSTM02: Equipamento de perfuração de petróleo - condensado de forma estática. Do problema de autovalores generalizado $Kx = \lambda Mx$.

3. BCSSTRUC3: Matrizes de engenharia estrutural BCS (problemas de autovalores). Estes autoproblemas simétricos generalizados foram extraídos de vários pacotes de engenharia estrutural, como GT-STRUDL, MSC/NASTRAN e BCS ATLAS. Representam problemas interessantes encontrados depois que o primeiro conjunto de matrizes de engenharia estrutural foi coletado.

   (a) BCSSTK22 e BCSSTM22: Sistema destinado à tecelagem; do problema de autovalor generalizado $Kx = \lambda Mx$.

Tabela 5.1: *Características das Matrizes Teste*

Matriz	Tamanho	Entradas	κ	Tipo
NOS4	100×100	347	2,7e+03	SPD
NOS5	468×468	2820	2,9e+04	SPD
NOS6	675×675	1965	8,0e+06	SPD
BCSSTK01	48×48	224	1,6e+06	SPD
BCSSTM01	48×48	48	inf	SPS-D
BCSSTK02	66×66	2211	1,3e+04	SPD
BCSSTM02	66×66	66	8,8	SPD
BCSSTK22	138×138	417	1,7e+05	SPI
BCSSTM22	138×138	138	9,4e+02	SPD

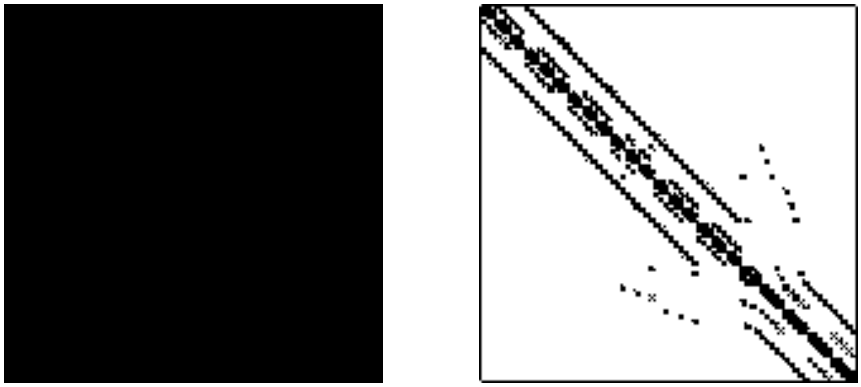

Figura 5.1: *Matrizes NOS4 e NOS5*

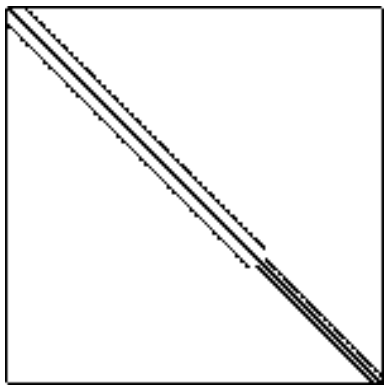

Figura 5.2: *Matriz NOS6*

O vetor solução utilizado em todos os problemas foi o vetor $x = (1, 1, ..., 1)^T$, para que pudéssemos conferir o resultado. No método RGMRES utilizamos $x^{(0)} = (0, 0, ..., 0)^T$ e os valores $c = 5$[5] e $c = 10$ para estabelecer uma comparação entre as bases ortonormais. Como critérios de parada utilizamos um resíduo mínimo de $1,0 \times 10^{-10}$ ou o máximo de 5000 iterações.

5.2 Resultados numéricos

Os códigos foram implementados no MatLab 5.3 e executados num computador Pentium IV 1.5Gz com 128 MB de memória, sob sistema operacional

[5] c representa a dimensão da base ortonormal para gerar uma aproximação da solução.

Figura 5.3: *Matrizes BCSSTK01 e BCSSTM01*

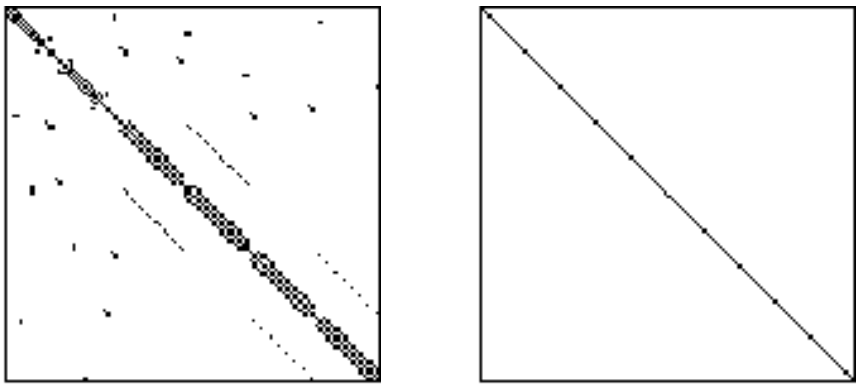

Figura 5.4: *Matrizes BCSSTK22 e BCSSTM22*

Windows 98. Foram avaliados os pré-condicionadores de Jacobi, da fatoração incompleta LU (LUINC) e da fatoração incompleta de Cholesky (CHOLINC), estes dois últimos do próprio MatLab, aplicados aos métodos RGMRES e Gradiente Conjugado. O algoritmo usado por cholinc para computar o fator de Cholesky incompleto é incomum. Dada uma matriz simétrica A, o procedimento cholinc chama no Matlab o procedimento luinc que usa uma fatoração LU incompleta com pivotamento. A referência é feita em Saad [40]. As colunas da matriz triangular superior de U que se obteve de luinc são modificadas pela raiz quadrada do valor absoluto do elemento diagonal naquela coluna, e a matriz modificada é então o incompleto (superior triangular) fator de Cholesky.

Os parâmetros utilizados para análise nos testes foram: número de iterações, norma de r (resíduo) e o tempo de processamento das iterações. Não foi analisado neste trabalho, o tempo de construção e de implementação dos pré-condicionadores.

Nas tabelas apresentadas a seguir, observamos alguns resultados que passaremos a analisar.

Tabelas referentes aos resultados do pacote LANPRO.

Tabela 5.2: *Conjunto LANPRO - Método RGMRES*

Matriz	Pré-cond.	c	Iterações	Resíduo	Tempo
NOS4	-	5	464	1,5002e-010	1,3700
	-	10	134	9,9719e-011	1,1600
	Jacobi	5	150	2,8543e-011	1,5400
		10	58	2,3851e-011	1,2000
	LUINC	5	030	5,1549e-012	0,3300
		10	8	3,6089e-012	0,2200
	CHOLINC	5	4	1,3207e-012	0,0500
		10	2	1,8832e-012	0,0600
NOS5	-	5	5001	1,8362e-004	32,1400
		10	5001	4,0960e-004	74,8700
	Jacobi	5	348	8,2264e-006	18,1800
		10	77	7,4870e-006	7,8500
	LUINC	5	5001	1,6231e+0,06	973,9300
		10	5001	1,5919e+0,06	1855,6600
	CHOLINC	5	18	2,2313e-007	1,7100
		10	6	7,6098e-008	1,0400
NOS6	-	5	5001	3,0732e+001	1720,3200
		10	5001	2,3606e+001	78,6500
	Jacobi	5	5001	2,2943e+001	1729,0600
		10	5001	2,3055e+001	967,9000
	LUINC	5	5001	2,6316e+001	757,2800
		10	5001	2,7551e+001	1456,2900
	CHOLINC	5	5001	1,9191e-003	1711,8100
		10	2	2,9754e-006	0,1700

Conforme as tabelas 5.2 e 5.3, referentes ao pacote de matrizes LANPRO, no RGMRES, para a matriz NOS4 ocorreu convergência em todos os casos, sendo que o mais eficiente foi o obtido com o uso do pré-condicionador CHOLINC.

Tabela 5.3: *Conjunto LANPRO - Método do Gradiente Conjugado*

Matriz	Pré-cond.	Iterações	Resíduo	Tempo
NOS4	-	90	9,9998e-011	0,0600
	Jacobi	82	4,8622e-011	0,1600
	LUINC	25	3,2116e-011	0,1100
	CHOLINC	13	9,0098e-012	0,0600
NOS5	-	530	7,2749e-011	0,9900
	Jacobi	312	7,8459e-011	2,6900
	LUINC	5001	5,2177e+006	166,8100
	CHOLINC	37	5,9086e-011	0,5500
NOS6	-	2065	9,2206e-011	3,2400
	Jacobi	159	6,7246e-011	2,0300
	LUINC	51	4,5072e-011	1,3700
	CHOLINC	21	9,2066e-011	0,1700

Para NOS5 o RGMRES apenas foi eficiente com os pré-condicionadores de Jacobi e CHOLINC, sendo que não houve convergência com LUINC e este resultado foi ainda pior que para o RGMRES não pré-condicionado. Quanto à matriz NOS6 no RGMRES quando usamos $c = 5$, o que está abaixo do padrão considerado como ideal $10 \leq c \leq 50$, realmente não observamos um resultado satisfatório. O número limite de iterações não foi suficiente para que houvesse convergência e o tempo foi excessivamente grande. Já com $c = 10$ tivemos um resultado muito bom, tanto no que se refere ao número de iterações como ao tempo de processamento das mesmas. Este fato comprova o que foi discutido no método RGMRES (cap 3), de que um valor inadequado na escolha de c leva a produzir uma seqüência estacionária de resíduos e a não ocorrer convergência.

Nas outras duas matrizes, o valor de $c = 10$ levou a um número menor de iterações, embora o fator tempo não tenha sofrido significativa alteração, talvez pelo fato de que as matrizes eram um pouco menores e então este aumento em c não altere muito os cálculos.

Quanto ao método do GC, este mostrou ser bem eficiente para todas as matrizes citadas, embora o RGMRES com $c = 10$, tenha apresentado melhores resultados em termos de número de iterações. Pode-se observar que CHOLINC

levou a um número reduzido de iterações e tempo, mostrando-se o melhor para as três matrizes e que a associação GC-LUINC não foi nada eficaz na matriz NOS5, embora esta apresente características muito próximas das outras.

Nas matrizes do pacote BCSSTRUC01, tabelas 5.4 e 5.5, observamos uma sensível diferença entre as matrizes K e M. As primeiras, BCSSTK01 e 02 apresentam um comportamento oposto em relação à utilização dos pré-condicionadores LUINC e CHOLINC. A primeira não converge com LUINC, mas tem um bom resultado ao se aplicar CHOLINC, quer seja no RGMRES ou GC. Já a matriz BCSSTK02 no RGMRES converge com LUINC mas não converge com CHOLINC ao utilizar $c = 5$, porém, ao utilizarmos $c = 10$, o ganho de tempo e a redução no número de iterações é bastante significativo. Esta mesma matriz no GC tem o resultado oposto ao anterior, convergindo com a utilização de CHOLINC, embora com um tempo e iterações não tão boas quanto no RGMRES. Novamente observa-se que o número de iterações, a norma do erro e o tempo de execução foram sensivelmente melhorados ao aplicarmos o método RGMRES usando $c = 10$.

Nas matrizes M, observamos que qualquer dos dois métodos, RGMRES ou GC, resolveriam facilmente mesmo sem uso de pré-condicionadores, diretamente por se tratarem em geral de matrizes diagonais, cuja solução pelo método direto é muito eficiente. O único inconveniente ocorre com a matriz BCSSTM01, onde Jacobi não pode ser aplicado, provavelmente devido à existência de zeros na diagonal.

As tabelas 5.6 e 5.7 apresentam os resultados obtidos com as matrizes do pacote BCSSTRUC3. A matriz BCSSTK22 no RGMRES não obteve bons resultados sem o uso de pré-condicionador ou ao associar o pré-condicionador LUINC, porém ao utilizarmos Jacobi ou CHOLINC obtivemos uma pequena melhora nos resultados. Para estas matrizes, no método RGMRES e comparando a utilização de $c = 5$ e $c = 10$, não se observa um ganho tão significativo, pelo fato de trabalharmos com matrizes de tamanho relativamente pequeno. Já com o método do Gradiente Conjugado, GC, os resultados foram sensivelmente melhores, principalmente com o uso do pré-condicionador CHOLINC, onde o número de iterações e o tempo das

mesmas foi bastante reduzido. Porém, não se obteve bom resultado com a utilização do pré-condicionador LUINC, onde não ocorreu convergência em nenhum dos casos. Nesta última tabela, um fato que deve ser observado é em relação ao tempo. O sistema adotado no Matlab desconsidera resultados tão pequenos, abaixo de $10^{-5}s$ o que leva ao valor 0,00000 na tabela.

Para a matriz BCSSTM22 o resultado pode ser considerado o mesmo que para as matrizes M do pacote anterior onde se poderia deixar de lado o uso de qualquer pré-condicionador, apesar de se observar que o uso destes poderia diminuir ainda mais o número de iterações e o tempo para tal. Também observamos que o resíduo melhorou sensivelmente com o uso de pré-condicionadores

Um fato que vale ser ressaltado é que o método do Gradiente Conjugado para esse tipo específico de matrizes é bastante eficaz, e mesmo que em alguns casos o número de iterações tenha sido maior, o tempo de processamento não foi proporcionalmente maior. Concluímos que para matrizes SPD, tanto o GC quanto o RGMRES são eficazes, pois resolvem o sistema. Porém o GC é mais eficiente que o RGMRES por apresentar menor custo computacional. No entanto se matrizes de coeficientes dos sistemas fossem indefinidas, então o método RGMRES com certeza apresentaria melhor desempenho, ou seja, mais eficiência, pois resolveria o sistema.

Tabela 5.4: *Conjunto BCSSTRUC1 - Método RGMRES*

Matriz	Pré-cond.	c	Iterações	Resíduo	Tempo
BCSSTK01	-	5	5001	1,2240e+004	14,2300
		10	5001	3,1982e+001	35,4300
	Jacobi	5	209	2,7250e-002	1,2600
		10	5001	4,0503e+000	64,3700
	LUINC	5	5001	2,9061e+011	38,0100
		10	5001	3,6950e+011	77,1700
	CHOLINC	5	19	1,5965e-003	0,1700
		10	642	6,6708e-011	10,4400
BCSSTM01	-	5	1	2,2015e-013	0,0000
		10	1	2,2015e-013	0,0500
	Jacobi	5	0	7,7460e+002	1,8100
		10	0	7,7460e+002	0,1100
	LUINC	5	1	6,6046e-013	0,0000
		10	1	6,6046e-013	0,0600
	CHOLINC	5	1	6,8390e-013	0,0500
		10	1	7,9378e-013	0,1100
BCSSTK02	-	5	5001	1,5334e-007	22,5200
		10	5001	2,4052e-006	49,5400
	Jacobi	5	195	3,2166e-007	1,8100
		10	93	3,1542e-007	1,7000
	LUINC	5	1	1,8066e-011	0,0600
		10	1	1,7151e-011	0.0600
	CHOLINC	5	5001	2.1105e+002	129,1300
		10	11	5,9269e-007	0,5000
BCSSTM02	-	5	7	2,1748e-011	0,0600
		10	3	3,6087e-012	0,0000
	Jacobi	5	1	5,4336e-016	0,0000
		10	1	5,4336e-016	0,0500
	LUINC	5	1	5,4826e-016	0,0000
		10	1	5,4826e-016	0,0000
	CHOLINC	5	1	4,3802e-016	0,0000
		10	1	3,9245e-016	0,0000

Tabela 5.5: *Conjunto BCSSTRUC1 - Método do Gradiente Conjugado*

Matriz	Pré-cond.	Iterações	Resíduo	Tempo
BCSSTK01	-	192	9,641e-011	0,0500
	Jacobi	81	5,3750e-011	0,0500
	LUINC	5001	1,0198e+010	6,0400
	CHOLINC	36	1,2953e-011	0,0600
BCSSTM01	-	2	2,7519e-014	0,0600
	Jacobi	1	-	0,3300
	LUINC	1	0,0000e+000	0,0000
	CHOLINC	1	1,9691e-013	0,0000
BCSSTK02	-	79	3,7576e-011	0,1100
	Jacobi	64	5,5072e-011	0,1600
	LUINC	5001	5,5634-002	33,1800
	CHOLINC	28	3,3789e-011	0,1100
BCSSTM02	-	12	2,6067e-015	0,0000
	Jacobi	1	1,6997e-017	0,0000
	LUINC	1	0,0000e+000	0,0000
	CHOLINC	1	3,1277e-016	0,0000

Tabela 5.6: *Conjunto BCSSTRUC3 - Método RGMRES*

Matriz	Pré-cond.	c	Iterações	Resíduo	Tempo
BCSSTK22	-	5	5001	2,1794e+001	25,2700
		10	5001	1,5709e-002	39,7100
	Jacobi	5	391	2,5324e-005	7,4700
		10	104	5,8312e-005	2,6400
	LUINC	5	5001	2,2445e+007	108,4200
		10	5001	3,2489e+006	155,5500
	CHOLINC	5	9	1,8012e-006	0,1600
		10	3	1,5020e-006	0.0600
BCSSTM22	-	5	192	9,5874e-011	0,6100
		10	39	9,9597e-011	0,3300
	Jacobi	5	1	1,4553e-017	0,0000
		10	1	1,4553e-017	0,0000
	LUINC	5	1	2,4558e-017	0,0500
		10	1	2,4558e-017	0,0000
	CHOLINC	5	1	7,7084e-018	0,0000
		10	1	7,7084e-018	0.0000

Tabela 5.7: *Conjunto BCSSTRUC3 - Método do Gradiente Conjugado*

Matriz	Pré-cond.	Iterações	Resíduo	Tempo
BCSSTK22	-	514	4,9452e-011	0,3300
	Jacobi	145	9,2376e-011	0,3300
	LUINC	5001	2,3674e+007	14,5000
	CHOLINC	22	1,8852e-011	0,0000
BCSSTM22	-	59	8,9455e-011	0,0000
	Jacobi	1	2,5378e-018	0,0000
	LUINC	1	0,0000e+000	0,0000
	CHOLINC	1	4,1691e-018	0,0000

6 CONCLUSÕES E TRABALHOS FUTUROS

Com relação ao método iterativo usado em cada tipo de problema, não verificamos, na bibliografia pesquisada, nenhuma regra que estabeleça qual deva ser usado em cada um dos casos. A escolha do melhor método, ou seja o de menor custo computacional, que necessite de um número menor de iterações para aproximar da solução exata é, muitas vezes, mais uma questão de experiência ou até mesmo de sorte. Mesmo assim, em alguns casos especiais é possível estabelecerem-se alguns critérios de comparação entre os métodos e condições suficientes para a convergência, de aplicação relativamente simples e que facilitam a decisão prévia pelo melhor método.

O uso de pré-condicionadores, associados aos métodos iterativos, que podem vir a melhorar a eficiência dos métodos iterativos, também depende de certa experimentação anterior para que se possa definir qual deva ser usado em cada um dos casos.

O que observamos neste trabalho, em relação principalmente aos dois métodos explorados, mais especificamente, o método do Gradiente Conjugado (GC) e o método do Resíduo Mínimo Generalizado em sua versão reinicializada (RGMRES), utilizados nos testes, é que para sistemas cujas matrizes de coeficientes são simétricas e positivas definidas, o GC normalmente é mais eficaz. O RGMRES é considerado muito eficiente para resolver sistemas não simétricos e não singulares, por sua vez o GC não os resolveria.

Sobre os pré-condicionadores das fatorações incompletas, aplicados em matrizes principalmente esparsas, o principal problema observado pelos autores pesquisados, foi o fato de no decorrer da eliminação de certos elementos alguns pivôs encontrados serem nulos, o que podia interromper os cálculos, devido a divisão por zero, ou negativos que não resultavam em uma matriz também simétrica

e positiva definida. Este fato em alguns casos, diminuiu a precisão dos cálculos, fazendo com que a convergência esperada não ocorresse.

Observamos que para matrizes SPD, tanto o GC quanto o RGMRES são eficazes, pois resolvem o sistema, mas o GC é mais eficiente, pois tem menor custo computacional (o número de operações aritméticas de ponto flutuante por iteração é menor que o do RGMRES). Para o caso de matrizes não-simétricas, o RGMRES é mais eficaz, pois resolve o sistema, enquanto o GC, na maioria dos casos não é, pois não consegue resolver o sistema.

Para os próximos trabalhos, julgamos que a aplicação dos pré-condicionadores obtidos com fatorações incompletas em sistemas cuja matriz de coeficientes não satisfaçam às condições aqui analisadas, ou a aplicação dos mesmos em computadores com memória compartilhada ou computação paralela, poderiam resultar em uma análise mais pormenorizada em relação aos métodos que aqui procuramos expor. Deste modo deixamos como sugestão para trabalhos futuros:

1. Fatorações incompletas em matrizes não-simétricas.
2. A paralelização das fatorações incompletas LU e de Cholesky
3. A associação dos pré-condicionadores das fatorações incompletas a outros métodos iterativos.

BIBLIOGRAFIA

[1] On the eigenvalue distribuition of a class of preconditioning methods. *Numer.Math.*, 48 (1986), 479–498.

[2] On the rate of convergence of the preconditioned conjugate gradient method. *Numer.Math.*, 48 (1986), 499–523.

[3] *Analysis of incomplete factorizations with fixed storage allocation, in Preconditioning Methods - Theory and Aplications.* 1983.

[4] Templates for the solution of linear systems: building blocks for iterative methods.

[5] *O Método Iterativo GMRES em Blocos Dinâmicos para a solução de sistemas lineares com múltiplos termos independentes em computadores paralelos.* Dissertação (mestrado), UFRGS, Porto Alegre, 2001.

[6] Incomplete LU factorization: A multifrontal approach. Technical Report, Computer and Information Sciences Departament, University of Florida, Gainesville, FL, 32611 USA., 1995.

[7] *Algoritmos Numéricos.* LTC - Livros Técnicos e Científicos, Rio de Janeiro, RJ, 2001.

[8] A chordal preconditioner for large scale optimization. *Mathematical Programming.*, 40 (1988), 265–287.

[9] Block preconditioning for the conjugate gradient method. *Technical Report LBL-14856 6*, 6 (1985), 220–252.

[10] *Métodos numéricos para as engenharias e ciências aplicadas*, 2 ed. Editora da UNICAMP, Campinas, SP., 1993.

[11] *A study on interative methods for the solution of systems of linear equations on transputer networks.* Ph.d. diss., University of Kent at Canterbury Computing Laboratory, Kent, 1992.

[12] PIM 2.2 The Parallel Iterative Methods Package for Systems of Linear Equations. User's Guide (Fortran 77 version).

[13] The Parallel Iterative Methods (PIM) package for the solution of systems of linear equations on parallel computers. *Applied Numerical Mathematics 19* (1995), 33–50.

[14] . Approximating the inverse of a matrix for use on iterative algorithms on vectors processors. *Computing 22* (1979), 257–268.

[15] The effect of ordering on preconditioned conjugate gradients. *BIT 29* (1989), 635–657.

[16] Analysis of parallel incomplete point factorizations. *Lin. Alg. Appl. 154*, 156 (1991), 723–740.

[17] On the existence problem of incomplete factorization methods, Dec. 1999. LAPACK Working Note 144, UT-CS-99-435.

[18] The 'weighted modification' incomplete factorization method. Technical Report LAPACK Working Note 145, UT-CS-99-436, Computer Science Departament, May 1999.

[19] Eficient implementation of a class of preconditioned conjugate gradiente methods. *SIAM J. Sci.Stat. Comput. 2*, 2 (1981), 1–4.

[20] A stability analysis of incomplete LU factorisation. *Math. Comp.*, 47 (1986), 191–217.

[21] An Ad-Hoc SOR method. *J. Comput. Phys 43* (1981), 31–45.

[22] Necessary and sufficient conditions for the existence of a conjugate gradient method. *SIAM J. Numer. Anal. 21*, 2 (1984), 352–362.

[23] *Matrix computations*. Academic Press, New York, 1981.

[24] *Simplified Linear Equation Solvers Users Manual*. Argonne National Laboratory, June 1993.

[25] Polynomial preconditions for conjugate gradient calculations. *SIAM J. Numer Anal. 20* (1983), 362–376.

[26] An improved incomplete cholesky factorization. *ACM Trans. Mat. Software.*, 21 (1995), 5–17.

[27] The incomplete Cholesky-Conjugate Gradient method for the iterative solution of systems of linear equations*. *Journal of Computational Physics*, 26 (1978), 43–65.

[28] ITPACK 2C: A Fortran package for solving large sparse linear systems by adaptive accelerated iterative methods. *ACM Trans. Math. Soft.*, 8 (1982), 302–322. Algorithm 586.

[29] Incomplete Cholesky factorizations with limited memory. *SIAM, J. Comput.*, 1 (1999), 24–45.

[30] An incomplete Cholesky factorization for dense matrices. *Department of Computer Science and information Engineering.*, 2 (1999), 1–25.

[31] An incomplete factorization technique for positive definite linear systems. *Mathematics of Computation 34*, 150 (Apr. 1980), 473–497.

[32] An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comp. 31* (1977), 148–162.

[33] Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems. *J. comput. Phys. 44* (1981), 134–155.

[34] Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients. *ACM Trans. Math. Software 6* (1980), 206–209.

[35] How fast are nonsymmetric matrix iterations? To appear: Siam. J. Matrix Anal. Applics, Mar.1990.

[36] *Introdução à Álgebra Linear.* LTC - Livros Técnicos e Científicos, Rio de Janeiro, RJ, 1998.

[37] Krylov subspaces methods for solving large unsymmetric systems. *Math. Comp.*, 37 (1981), 105–126.

[38] Practical use of polynomial preconditionings for the conjugate gradient method. *SIAM J. Sci. Stat. Comput 6*, 4 (Oct. 1985), 865–881.

[39] A dual threshold incomplete ILU factorization. *Numer. Linear Algebra Appl.*, 4 (1994), 387–402.

[40] *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, 20 Park Plaza, Boston, 1995.

[41] GMRES: a Generalized Minimal Residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.* 7 (1986), 856–869.

[42] Incomplete Cholesky factorization with sparsity pattern modification. Technical Report TR394, Indiana University Bloomington, Bloomington., 1993.

[43] A necessary and sufficient symbolic condition for the existence of incomplete Cholesky factorization. Technical Report TR440, Indiana University Bloomington, Bloomington, IN 47405, 1995.

[44] Incomplete LU factorization: A multifrontal approach. Technical Report TR95-024, Computer and Information Sciences Departament, University of Florida, Gainesville, 32611 USA., 1995.

.