Notas em Matemática Aplicada ISSN 2175-3385
E-ISSN 2236-5915

Volume 94, 2023

Sociedade Brasileira de Matemática Aplicada e Computacional

2023

A Sociedade Brasileira de Matemática Aplicada e Computacional - SBMAC publica, desde as primeiras edições do evento, monografias dos cursos que são ministrados nos CNMAC.

Para a comemoração dos 25 anos da SBMAC, que ocorreu durante o XXVI CNMAC em 2003, foi criada a série **Notas em Matemática Aplicada** para publicar as monografias dos minicursos ministrados nos CNMAC, o que permaneceu até o XXXIII CNMAC em 2010.

A partir de 2011, a série passa a publicar, também, livros nas áreas de interesse da SBMAC. Os autores que submeterem textos à série Notas em Matemática Aplicada devem estar cientes de que poderão ser convidados a ministrarem minicursos nos eventos patrocinados pela SBMAC, em especial nos CNMAC, sobre assunto a que se refere o texto.

O livro deve ser preparado em **Latex, com as figuras em .eps, .pdf e etc.** e ter entre **80 e 150 páginas**. O texto deve ser redigido de forma clara, acompanhado de uma excelente revisão bibliográfica e de **exercícios de verificação de aprendizagem** ao final de cada capítulo. O idioma pode ser Português ou Espanhol.

Veja todos os títulos publicados nesta série na página
http://https://proceedings.science/notas-sbmac

Sociedade Brasileira de Matemática Aplicada e Computacional

2023

PRECONDICIONADORES BASEADOS NA APROXIMAÇÃO DA INVERSA PARA SOLUÇÃO DE SISTEMAS LINEARES DE GRANDE PORTE

Julia Sekiguchi da Cruz
julia-seki@hotmail.com

Programa de Pós-Graduação em Engenharia Mecânica
Faculdade de Engenharia
Universidade do Estado do Rio de Janeiro

Moisés Ceni de Almeida
moisesceni@gmail.com

Departamento de Matemática
Instituto de Ciências Exatas
Universidade Federal Rural do Rio de Janeiro

Luiz Mariano Carvalho
luizmc@ime.uerj.br

Departamento de Matemática Aplicada
Instituto de Matemática e Estatística
Universidade do Estado do Rio de Janeiro

Michael Ferreira de Souza
michael@ufc.br

Departamento de Estatística e Matemática Aplicada
Centro de Ciências
Universidade Federal do Ceará

Sociedade Brasileira de Matemática Aplicada e Computacional

São Carlos - SP, Brasil
2023

Coordenação Editorial: Mateus Bernardes

Coordenação Editorial da Série: Sandra M. C. Malta

Editora: SBMAC

Capa: Matheus Botossi Trindade

Patrocínio: SBMAC

Catalogação elaborada pela Biblioteca do IBILCE/UNESP
Bibliotecária: Maria Luiza Fernandes Jardim Froner

Cruz, Julia S. da
Precondicionadores Baseados na Aproximação da Inversa para Solução de Sistemas Lineares de Grande Porte - São Carlos, SP : SBMAC, 2023, 114 p., 21,5 cm - (Notas em Matematica Aplicada; v. 94)

ISBN 978-65-86388-12-1 e-ISBN 978-65-86388-11-4

1. Sistema Linear de Grande Porte 2. Métodos de Krylov 3. Precondicionador 4. Inversa Aproximada 5. Matriz em Blocos I. Cruz, Julia S. da II. Almeida, Moisés C. de III. Carvalho, Luiz M IV. Souza, Michael F. IV. Título. V. Série

CDD - 51

Conteúdo

Prefácio

Sistemas lineares da forma $Ax = b$, em que $A \in \mathbb{R}^{n \times n}$ é não singular e esparsa, $b \in \mathbb{R}^n$ é o vetor dos termos independentes e $x \in \mathbb{R}^n$ é o vetor solução são de grande relevância em problemas da ciência e indústria. Para matrizes de grande porte, o uso de métodos diretos para a sua resolução, como a eliminação gaussiana, é impraticável, pois a complexidade é de $\mathcal{O}(n^3)$, fazendo-se necessário o uso de métodos iterativos. Observe-se que para algumas matrizes esparsas especiais a complexidade é de menor ordem. Dentre eles, temos os métodos de projeção em subespaços de Krylov como por exemplo o Método de Gradientes Conjugados (CG), para matrizes simétricas positivas definidas (SPD).[1] Porém, métodos de Krylov podem convergir lentamente sem um precondicionador adequado.

A ideia do precondicionador é construir um operador em que a matriz precondicionada possua um menor número de condicionamento e uma melhor distribuição de autovalores. Essas alterações no problema original comumente fazem com que o sistema resultante seja solucionado em um menor número de iterações. Existem diversos tipos de precondicionadores, dentre eles, os que aproximam a fatoração da inversa de A. Neste livro, abordamos um precondicionador deste tipo, conhecido como Inversa Aproximada (*Approximate Inverse* ou AINV).

O livro apresenta vários resultados teóricos sobre o precondicionador de Inversa Aproximada proposto por Benzi, Meyer e Tůma, em 1996. Alguns destes resultados se encontravam dispersos em artigos científicos e outros são inéditos. Nossa principal ferramenta é a Álgebra Linear Computacional, tanto para tratar a versão escalar, quanto para a versão em blocos do método.

Também reunimos e classificamos as principais variações do AINV encontradas na literatura da área, pontuando suas semelhanças e diferenças, além de estudarmos suas complexidades. Todas as variantes são apresentadas com seus respectivos algoritmos e detalhes são discutidos. Além disso, adaptamos algumas das variações escalares para o tratamento em matrizes em blocos.

[1] Daqui para frente, chamaremos os métodos de projeção em subespaços de Krylov apenas por métodos de Krylov.

O público-alvo de nosso estudo são pesquisadores (na academia e na indústria) e estudantes de graduação e pós-graduação que necessitam resolver sistemas lineares de grande porte, pois neste contexto, o uso de métodos iterativos, em particular dos métodos de Krylov, é mandatório, sendo, portanto, imprescindível a utilização de precondicionadores.

No capítulo 1, é feita uma revisão sobre dois métodos de Krylov. No Capítulo 2, descrevemos e analisamos o método de biconjugação que é a base dos precondicionadores de Inversa Aproximada. Dentre elas, abordamos as relações com os fatores L e U da matriz do problema, estratégias de descarte, procedimentos para evitar a quebra, entre outros. No Capítulo 3, revisamos as principais variações do AINV, baseadas nos seus trabalhos originais, além de classificá-las em quatro grupos. No Capítulo 4, analisamos as complexidades das variações do AINV. No Capítulo 5, exibimos as adaptações do AINV para matrizes simétricas em blocos, apresentando alguns de seus principais resultados. No Capítulo 6, discutimos as adaptações do AINV para matrizes não simétricas em blocos, além de analisar os principais trabalhos já publicados e abordar algumas variações desse tipo de precondicionador. Ao final de cada capítulo, propomos alguns exercícios.

Por fim, gostaríamos de agradecer ao trabalho dos editores e revisores que contribuíram para a melhoria do texto final, sendo que todos os possíveis erros restantes são de nossa total responsabilidade.

Fortaleza, Rio de Janeiro e Seropédica, 14 de fevereiro de 2023.

Julia Sekiguchi da Cruz, Moisés Ceni de Almeida,
Luiz Mariano Carvalho e Michael Ferreira de Souza.

Capítulo 1

Métodos de Krylov

1.1 Introdução

O objetivo deste capítulo é apresentar algumas informações breves e relevantes sobre os métodos de Krylov para solução de sistemas lineares $Ax = b$, em que $A \in \mathbb{R}^{n\times n}$ é não-singular, $b \in \mathbb{R}^n$ é o lado direito conhecido e $x \in \mathbb{R}^n$ é o vetor de incógnitas do problema. Uma revisão mais completa deste métodos pode ser encontrada em [27]. Discutiremos adiante algumas propriedades dos espaços de Krylov[1]. Esses espaços apareceram originalmente em uma técnica proposta por A.N. Krylov para construção de polinômios característicos de matrizes. Sem entrar em detalhes (que podem ser vistos em [57, seção 7.11]), o resultado do método é a construção de matrizes K, não singular, e H, Hessenberg[2] tal que o produto $K^{-1}AK = H$ seja válido. Trata-se de uma relação de similaridade e, portanto, A e H têm o mesmo polinômio característico. As colunas da matriz K, em uma primeira fase do método, podem ser construídas pela multiplicação de um vetor b por A, a saber, a $j-$ésima coluna de $K(:, j)$[3] é dada por $A^{j-1}b$, ou seja

$$K = \begin{bmatrix} b & Ab & A^2b & \dots & A^{k-1}b \end{bmatrix}.$$

[1] Aleksei Nikolaevich Krylov (1863-1945) mostrou em 1931 [49] como usar sequências da forma $\{b, Ab, A^2b, \dots\}$ para construir o polinômio característico de uma matriz. Krylov foi um matemático aplicado russo (engenheiro marítimo de formação) cujos interesses científicos ultrapassavam em muito as áreas de seu treinamento inicial em ciência naval, que envolviam flutuação, estabilidade, etc. Krylov foi diretor do Instituto de Física-Matemática da Academia de Ciências da União Soviética de 1927 a 1932, e em 1943 ganhou um "prêmio estatal" por suas teorias sobre bússolas. Foi condecorado como "herói do trabalho socialista" e é um dos poucos matemáticos que tem um acidente geográfico lunar associado a seu nome, trata-se da cratera Krylov. (traduzido pelos autores de [57])

[2] Uma matriz A é denominada Hessenberg superior (inferior) caso $a_{ij} = 0$, para todos $i > j + 1$ ($i < j + 1$).

[3] Estamos usando a notação do Matlab para nos referirmos a partes de uma matriz: $A(i, :)$ será a $i-$ésima linha da matriz A e $A(:, j)$ será a $j-$ésima coluna da matriz A.

A técnica desenvolvida nesse método está na gênese de todos os, assim chamados, métodos de Krylov.

O subespaço de Krylov de dimensão k associado à solução iterativa do sistema linear $Ax = b$ é formado por todas as combinações lineares dos vetores: r_0, Ar_0, A^2r_0,..., $A^{k-1}r_0$, em que x_0 é uma aproximação inicial e $r_0 = b - Ax_0$ é o resíduo inicial e sera denotado por $\mathcal{K}_k(A, r_0)$. No Exercício 1.1, o leitor deverá provar que, dadas algumas condições, a solução exata do sistema linear, $Ax = b$, está no espaço afim $x_0 + \mathcal{K}_k(A, r_0)$, tal que k seja o menor inteiro para o qual $\mathcal{K}_{k+1}(A, r_0) \subset \mathcal{K}_k(A, r_0)$.

Em [27], são apresentados métodos de projeção em subespaços quaisquer, aqui trataremos de métodos de projeção em subespaços de Krylov. Sejam $\mathcal{K}_k$ e $\mathcal{L}_k$ subespaços de Krylov, ambos com dimensão k. Um método de projeção consiste em, dado um valor inicial x_0, construir uma sequência $\left(x_k\right)_{k=1:m}$ de vetores que atendam as seguintes propriedades:

$$x_k - x_0 \in \mathcal{K}_k, \tag{1.1.1}$$

$$r_k = b - Ax_k \perp \mathcal{L}_k. \tag{1.1.2}$$

Como não se sabe *a priori* o valor exato da solução e a cada novo passo do método tem-se uma nova aproximação x_k, à primeira vista, o melhor que se pode fazer é calcular a diferença, $r_k = b - Ax_k$, chamada de **resíduo**. Desta forma, podemos chamá-los de **métodos de projeção em subespaços de Krylov (MPSKs)**.

1.2 Método de gradientes conjugados (CG)

O **método de gradientes conjugados (CG)** é o método padrão para solução iterativa de sistemas lineares quando a matriz dos coeficientes é simétrica e positiva definida[4]. As propriedades deste método podem ser encontradas em diversos trabalhos, com as mais variadas abordagens, dentre estes destacamos o artigo original de Hestenes e Stiefel [42], o livro de Golub e Van Loan [38], o livro de Saad [67] e o livro de Málek e Strakoš [58].

Enquanto método de projeção, o **CG** pode ser descrito utilizando as equações 1.1.1 e 1.1.2, com $\mathcal{K}_k = \mathcal{L}_k = \{r_0, Ar_0, A^2r_0, \ldots, A^{k-1}r_0\}$.

Nos algoritmos deste livro, usaremos tanto a notação (r_k, p_k) quanto $r_k^{\mathsf{T}} p_k$ para representar o produto interno usual entre dois vetores r_k e p_k. Ou seja,

$$(r_k, p_k) = r_k^{\mathsf{T}} p_k = \sum_{i=1}^{n} r_k(i) * p_k(i),$$

[4] A matriz A é positiva definida, ou somente positiva, caso $x^T Ax > 0$, para todo $x \neq 0$.

em que $r_k(i)$ é i-ésima coordenada do vetor r_k.

Apresentamos o método **CG** no Algoritmo 1.

Algoritmo 1: Método de gradientes conjugados (CG)

Dados: matriz $A \in \mathbb{R}^{n\times n}$ simétrica e positiva definida, lado direito $b \in \mathbb{R}^n$, aproximação inicial $x_0 \in \mathbb{R}^n$ e tolerância tol.
Resultado: x^*, tal que $Ax^* \approx b$.

$k \leftarrow 0$;
$r_0 \leftarrow b - Ax_0$;
$p_0 \leftarrow r_0$;
$c \leftarrow \text{tol} * ||b||_2$;
enquanto $||r_k||_2 > c$ **faça**
  $a_k \leftarrow \dfrac{(r_k, r_k)}{(Ap_k, p_k)}$;
  $x_{k+1} \leftarrow x_k + a_k p_k$, minimiza aproximação na direção da vez;
  $r_{k+1} \leftarrow r_k - a_k Ap_k$, resíduos ortogonais;
  $b_k \leftarrow \dfrac{(r_{k+1}, r_{k+1})}{(r_k, r_k)}$;
  $p_{k+1} \leftarrow r_{k+1} + b_k p_k$, direções A-ortogonais;
  $k \leftarrow k + 1$;
  $x^* \leftarrow x_k$;

1.3 Método de resíduos mínimos generalizado (GMRES)

O GMRES é um dos métodos mais utilizados para resolver iterativamente sistemas lineares não singulares quaisquer. As propriedades deste método e de algumas de suas variações podem ser encontradas no livro de Saad [67], um dos proponentes deste método, e no livro de Meurant e Tebbens [56]. Uma revisão histórica do GMRES é apresentada em [82].

Enquanto método de projeção, o **GMRES** pode ser descrito utilizando as equações 1.1.1 e 1.1.2, com $\mathcal{K}_k = \{r_0, Ar_0, A^2r_0, \ldots, A^{k-1}r_0\}$ e $\mathcal{L}_k = A\mathcal{K}_k$.

Há dezenas de variações do GMRES, aqui apresentaremos, no Algoritmo 2, apenas a mais simples, a versão do método com recomeço que costroi uma base ortonormal para um subespaço de Krylov.[5]

[5]Uma base ortonormal de um espaço vetorial é formada por vetores ortogonais, com norma igual a 1, que sejam linearmente independentes e gerem este espaço vetorial.

Algoritmo 2: Método de resíduos mínimos generalizado (GMRES) com recomeço

Dados: matriz $A \in \mathbb{R}^{n\times n}$ não singular, lado direito $b \in \mathbb{R}^n$, aproximação inicial $x_0 \in \mathbb{R}^n$, m o tamanho máximo do subespaço de Krylov a ser construído e tolerância tol.

Resultado: x^*, tal que $Ax^* \approx b$.

1 $r_0 \leftarrow b - Ax_0$;
2 $v_1 \leftarrow \dfrac{r_0}{||r_0||_2}$;
3 **para** $j = 1 : m$ **faça**
4 método de Arnoldi;
5 $w_j \leftarrow Av_j$;
6 **para** $i = 1 : j$ **faça**
7 Gram-Schmidt modificado;
8 $h_{ij} \leftarrow (w_j, v_i)$ monta a matriz $\overline{H}_m$;
9 $w_j \leftarrow w_j - h_{ij}v_j$;
10 $h_{j+1,j} \leftarrow ||w_j||_2$;
11 **se** $h_{j+1,j} = 0$ **então**
12 happy breakdown;
13 $m \leftarrow j$ e vá para 15;
14 $v_{j+1} \leftarrow \dfrac{w_j}{h_{j+1,j}}$ novo vetor da base ortonormal do subespaço de Krylov;
15 $y_m \leftarrow \min_{y\in\mathbb{R}^m} || \, ||r_0||_2 e_1 - \overline{H}_m y||_2$;
16 $x^* = x_0 + V_m y_m$;
17 Caso x^* não seja uma boa aproximação para a solução exata então $x_0 \leftarrow x^*$ e vá para 1;

1.4 Precondicionadores

Precondicionadores são essenciais para uma boa desempenho dos MPSKs. Apesar do nome precondicionamento estar ligado ao condicionamento da matriz dos coeficientes, há outros fatores que influenciam na convergência, como por exemplo a distribuição dos autovalores.

Um precondicionador pode ser tanto uma matriz quanto um método não linear. Sendo comum o uso de outros métodos de Krylov como precondicionadores, dando origem a métodos flexíveis ou ainda de iteração interna e externa.

Um bom precondicionador tem que acelerar a convergência do método, no mínimo, para compensar o custo de sua construção, mas o objetivo é sempre mais ambicioso. O difícil problema de se encontrar um precondicionador eficiente é que se deve identificar um operador M, linear ou não, que atenda a pelos menos, necessária mas não exclusiva-

mente, às seguintes propriedades:

1. *De alguma forma M^{-1} é uma boa aproximação para A^{-1}.* Embora não haja uma teoria geral, pode-se dizer que M deve ser de tal forma que $M^{-1}A$, ou AM^{-1}, deve ser próxima da matriz identidade e que seus autovalores sejam aglomerados em uma pequena região do plano complexo, longe de 0;

2. *M seja eficiente.* De forma que o método precondicionado convirja muito mais rapidamente do que a não precondicionado, compensando largamente o custo de construção e armazenamento de M;

3. *M ou M^{-1} sejam construídas em paralelo.* Para explorar as várias arquiteturas atuais, que cada vez mais utilizam o processamento paralelo.

Algebricamente, precondicionar um sistema linear $Ax = b$ seria aplicar uma das três transformações:

Pela esquerda:

$$M^{-1}Ax = M^{-1}b. \tag{1.4.3}$$

Como o lado direito é alterado, é necessário se ter atenção no critério de parada utilizado, pois ele deve refletir o novo problema tratado.

Pela direita:

$$AM^{-1}y = b, \quad \text{com} \quad M^{-1}y = x. \tag{1.4.4}$$

Nesse caso o lado direito não é alterado.

Ambos os lados: Caso o problema tratado seja simétrico e positivo-definido, torna-se importante preservar essas propriedades, nesse caso ao se aplicar um precondicionador simétrico em ambos os lados garantem-se ambas as características:

$$M^{-1/2}AM^{-1/2}y = M^{-1/2}b, \quad \text{com} \quad M^{-1/2}y = x. \tag{1.4.5}$$

1.4.1 Método de gradientes conjugados precondicionado (PCG)

Devemos observar que no Algoritmo 3 o precondicionador não é aplicado diretamente na matriz A, mas sim no resíduo. Esta técnica é uma forma de evitar a aplicação do precondicionador em ambos os lados de A e, ao mesmo tempo, garantir a simetria da matriz precondicionada. Para os detalhes desta abordagem ver [73, pág 40].

Algoritmo 3: Método de gradientes conjugados precondicionado (PCG)

Dados: matriz $A \in \mathbb{R}^{n\times n}$ simétrica e positiva definida, lado direito $b \in \mathbb{R}^n$, aproximação inicial $x_0 \in \mathbb{R}^n$, um precondicionador M e tolerância tol.
Resultado: x^*, tal que $Ax^* \approx b$.

1 $k \leftarrow 0$;
2 $r_0 \leftarrow b - Ax_0$;
3 $p_0 \leftarrow M^{-1}r_0$, primeira aplicação do precondicionador;
4 $c \leftarrow \text{tol} * ||b||_2$;
5 **enquanto** $||r_k||_2 > c$ **faça**
6 $\quad a_k \leftarrow \dfrac{(r_k, M^{-1}r_k)}{(Ap_k, p_k)}$, uso de aplicação anterior do precondicionador;
7 $\quad x_{k+1} \leftarrow x_k + a_k p_k$, ;
8 $\quad r_{k+1} \leftarrow r_k - a_k Ap_k$;
9 $\quad b_k \leftarrow \dfrac{(r_{k+1}, M^{-1}r_{k+1})}{(r_k, M^{-1}r_k)}$, nova aplicação do precondicionador e reuso de anterior;
10 $\quad p_{k+1} \leftarrow M^{-1}r_{k+1} + b_k p_k$, uso de aplicação anterior do precondicionador;
11 $\quad k \leftarrow k + 1$;
12 $x^* \leftarrow x_k$;

1.4.2 Método de resíduos mínimos generalizado precondicionado

Para o GMRES, com precondicionamento pela esquerda, ver Algoritmo 4, o precondicionador é usado no cálculo do primeiro resíduo, ver Passo 1, e durante o procedimento de Arnoldi, quando da preparação do novo vetor, no Passo 4.[6] Ou seja, $w = Av_i$ passa a ser $M^{-1}Av_i = M^{-1}w$. Vale sempre lembrar que, apesar de usarmos a notação matricial, o operador M, pode ser não-linear. Para a opção de aplicação do precondicionador pela direita ver [67].

1.4.3 Alguns precondicionadores

Um dos problemas mais críticos no desenvolvimento de solvers eficientes é a construção de precondicionadores e essa importância deverá se tornar cada vez mais evidente [77, pág. 319]. Vamos descrever brevemente algumas famílias de precondicionadores:

[6] O procedimento de Arnoldi é composto pelas linhas de 5 a 9 do Algoritmo 2.

Algoritmo 4: Método de resíduos mínimos generalizado (GMRES) com recomeço precondicionado pela esquerda

Dados: matriz $A \in \mathbb{R}^{n\times n}$ não singular, lado direito $b \in \mathbb{R}^n$, aproximação inicial $x_0 \in \mathbb{R}^n$, m o tamanho máximo do subespaço de Krylov a ser construído, um precondicionador M e tolerância tol.

Resultado: x^*, tal que $Ax^* \approx b$.

1 $r_0 \leftarrow M^{-1}(b - Ax_0)$, primeira aplicação do precondicionador;
2 $v_1 \leftarrow \dfrac{r_0}{||r_0||_2}$;
3 **para** $j = 1 : m$ **faça**
4 $w_j \leftarrow M^{-1}Av_j$, aplicação do precondicionador;
5 **para** $i = 1 : j$ **faça**
6 $h_{ij} \leftarrow (w_j, v_i)$;
7 $w_j \leftarrow w_j - h_{ij}v_j$;
8 $h_{j+1,j} \leftarrow ||w_j||_2$;
9 **se** $h_{j+1,j} = 0$ **então**
10 $m \leftarrow j$ e vá para 12;
11 $v_{j+1} \leftarrow \dfrac{w_j}{h_{j+1,j}}$;
12 $y_m \leftarrow \min_{y\in\mathbb{R}^m} || \, ||r_0||_2 e_1 - \overline{H}_m y||_2$;
13 $x^* = x_0 + V_m y_m$;
14 Caso x^* não seja uma boa aproximação para a solução exata então $x_0 \leftarrow x^*$ e vá para 1;

- fatorações incompletas;
- multigrid;
- decomposição de domínios;
- inversa aproximada.

As principais referências dessa seção são [5, cap. 10], [9] e [55, cap. 8]. Outras fontes são: [29], [53], [57], [74], [77] e [78].

Fatorações incompletas

Baseadas na eliminação gaussiana, as fatorações incompletas são precondicionadores muito utilizados. A ideia é que a fatoração aproximada $\widetilde{L}\widetilde{U}$, seja o mais próxima de A. Há algumas classes em que é possível provar a existência de uma fatoração incompleta, mas não há resultado geral de existência, mesmo quando da existência da fatoração LU

completa. Quando a matriz do sistema for simétrica e positiva definida podemos, também, fazer uma fatoração incompleta dos fatores de Cholesky.

Apresentaremos três alternativas, mas há outras, e para maiores detalhes as referências [28], [38] e [67] devem ser consultadas.

A ideia básica é realizar a fatoração incompleta baseada em algum critério que pare a fatoração antes que os fatores completos tenham sido calculados. Alguns dos critérios usados são:

1. **Sem preenchimento:** nesse caso os fatores aproximados $\tilde{L}$ e $\tilde{U}$ terão estrutura tal que $\tilde{L}\tilde{U}$ tenha a mesma esparsidade do que A. Também denominada preenchimento 0.

2. **Preenchimento controlado por posição:** permite-se que algumas posições anteriormente nulas em A venham a ser não nulas no produto $\tilde{L}\tilde{U}$. Esse controle se dá através da análise de um grafo de dependências da matriz A, ver [37].

3. **Preenchimento controlado por valor:** permite-se algum preenchimento, baseado em que o novo elemento do fator esteja acima de um teto pré-estabelecido.

Multigrid

Em vários problemas de computação científica, quando do uso de métodos iterativos, se identifica o seguinte fenômeno: após algumas iterações, o erro se torna suave, mas não necessariamente menor. Um dos princípios básicos do método de multigrid (multimalhas) [78] é exatamente o de buscar a suavização do erro. Essa parte do método faz uso de suavizadores. O outro princípio básico do método é o seguinte: a quantidade que é suave em uma determinada malha pode ser, sem grande perda, ou mesmo perda alguma, aproximada em uma malha mais grossa, com, por exemplo, o dobro de tamanho em cada célula. E assim, caso se tenha certeza que o erro tornou-se suave, após algumas iterações, pode-se aproximar o erro por um procedimento adequado em uma malha mais grossa, e assim, nesse segundo momento, a iteração torna-se bem mais barata.

Essa abordagem cria uma sequência de problemas auxiliares e pode ser construída através de procedimentos geométricos ou algébricos. Se o problema original é definido em uma malha que foi obtida através de vários passos de refinamento, pode-se usar uma hierarquia entre as malhas para se definir operadores de transferência entre malhas mais finas e mais grossas, nesse caso estaremos tratando do denominado Método Multigrid Geométrico [80]. Se, no entanto, uma hierarquia não

é definida, os operadores de transferência podem ser construídos algebricamente, a partir da matriz do sistema. Essa abordagem chama-se Método Multigrid Algébrico [66]. Sobre esses esquemas, suas implantações e demais aspectos do método, consultar [20], [41] , [78] e [79].

Decomposição de domínios

Esse classe de precondicionadores é bem adaptada à computação paralela. É baseada na ideia simples de dividir o domínio de definição do problema em vários subdomínios, e resolver em cada subdomínio um subproblema e reunir a informação completa após. Essa ideia simples tem uma grande variedade de decorrências, e está fortemente ligada à solução de equações diferencias parciais em domínios com geometria complexas ou quando equações diferentes são utilizadas em regiões diferentes do domínio de um mesmo problema. A grosso modo, podem-se dividir as abordagens de decomposição de domínio em duas famílias:

- O primeiro grupo recebe o nome de métodos de Schwarz. O domínio é dividido em subdomínios com recobrimento e subproblemas locais são resolvidos em cada subdomínio. A solução de um subdomínio se transforma em uma condição de fronteira para os subdomínios vizinhos, pois o recobrimento permite essa possibilidade. Esse método foi proposto por H.-A. Schwarz em 1870 [72] para provar a existência de soluções de problemas de equações diferenciais definidas em domínios com geometrias complexas, que separados em regiões mais simples, em que se conheçam as soluções, permite a prova de existência de solução para a região completa.

- O segundo grupo usa subdomínios sem recobrimento. É possível, nesse caso, dividir as incógnitas do problema em dois grupos: as que estão na interface dos subdomínios e as que se encontram nos diversos interiores de cada subdomínio. De forma completamente algébrica, pode-se calcular a matriz do complemento de Schur das incógnitas da interface em relação às demais incógnitas. O problema é resolvido para a interface e a solução serve de condição de fronteira para os problemas internos que podem ser resolvidos de forma independente, ver [25], [26] e [26]. Esses métodos recebem várias denominações, entre elas métodos de subestruturação ou métodos do complemento de Schur.

A quantidade e diversidade de métodos que surgiram nos últimos 20 anos é notável. Para um visão atual dos métodos de decomposição de domínio, recomenda-se a leitura dos livros [53], [74] e [76].

Inversas aproximadas

Nesse caso, ao invés de se construir uma aproximação de A, se constrói uma aproximação da inversa de A, daí o nome do precondicionador. Uma forma de se construir essa aproximação é minimizar, aproximadamente, a norma de Frobenius[7] da matriz $\|I - AM\|_F$, em que M é a inversa aproximada e precisa ter uma certa estrutura, através da fórmula

$$\|AM - I\|_F^2 = \sum_{j=1}^{n} \|(AM - I)e_j\|_2^2 \tag{1.4.6}$$

cuja solução do problema de minimização pode ser feita separando em n problemas independentes de quadrados mínimos, a serem resolvidos de maneira aproximada,

$$\min_{m_k} \|Am_k - e_k\|, \ k = 1 : n. \tag{1.4.7}$$

Esta alternativa é proposta em [40] para ser implantada em paralelo, e permite a construção de uma inversa aproximada.

Neste livro, trataremos de uma outra possibilidade, a partir do próximo capítulo.

Para outras abordagens, ver [30, 31, 39, 81].

Exercício 1.1. *Demonstrar que a solução do sistema $Ax = b$ está no espaço afim $x_0 + \mathcal{K}_k(A, r_0)$, tal que $\mathcal{K}_{k+1}(A, r_0) \subset \mathcal{K}_k(A, r_0)$, x_0 é o chute inicial e $r_0 = b - Ax_0$.*

Exercício 1.2. *Provar que os resíduos produzidos nos diversos passos do algoritmo de gradientes conjugados são ortogonais.*

Exercício 1.3. *Provar que as direções produzidas nos diversos passos do algoritmo de gradientes conjugados são A-ortogonais*[8]*.*

Exercício 1.4. *Provar que a minimização que ocorre no Passo12 do GMRES (Algoritmo 2) é equivalente à minimização do resíduo.*

Exercício 1.5. *Provar que a aplicação do precondicionador aparentemente pela esquerda no algoritmo de gradientes conjugados precondicionado (Algoritmo 3) é equivalente à aplicação do precondicionador na matriz em ambos os lados (ver 1.4.5).*

[7] A norma de Frobenius de uma matriz é dada por $||A||_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^2}$, em que a_{ij} são as entradas de A.

[8] Dois vetores são A-ortogonais caso $(x, Ay) = 0$, com A simétrica e positiva definida.

Capítulo 2

Inversa Aproximada e suas características

2.1 Introdução

Seja $A \in \mathbb{R}^{n \times n}$ uma matriz esparsa e não-singular. O precondicionador Inversa Aproximada (*Approximate Inverse* ou simplesmente AINV), proposto por Benzi, Meyer e Tůma para matrizes simétricas positivas definidas (SPD) em [12] e para qualquer matriz não-singular, Benzi e Tůma, em [13], tem o objetivo de calcular uma fatoração da inversa aproximada de A. O AINV tem como base o método de biconjugação completa que calcula as matrizes Z, W e D, tais que $W^{\mathsf{T}}AZ = D$, em que D é diagonal, Z e W são triangulares superior e inferior unitárias, respectivamente. Desta forma, $A^{-1} = ZD^{-1}W^{\mathsf{T}}$. A biconjugação completa fornece dois conjuntos de vetores, $\{z_i\}_{i=1}^{n}$ e $\{w_i\}_{i=1}^{n}$ que são biconjugados em relação à A, ou seja, $w_i^{\mathsf{T}}Az_j = 0$ se $i \neq j$. Esses vetores são as colunas das matrizes Z e W:

$$Z = [z_1, z_2, ..., z_n], W = [w_1, w_2, ..., w_n],$$

tais que

$$W^{\mathsf{T}}AZ = D = \begin{bmatrix} d_{11} & 0 & \cdots & 0 \\ 0 & d_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_{nn} \end{bmatrix} = diag(d_{11}, d_{22}, ..., d_{nn}).$$

O método de biconjugação é apresentado no Algoritmo 5. Ele se inicia com Z_0, $W_0 \in \mathbb{R}^{n \times n}$ como matrizes identidade $I_{n \times n}$ e ao final do processo, são produzidas W, D e Z, tais que $A^{-1} = ZD^{-1}W^{\mathsf{T}}$ ou $W^{\mathsf{T}}AZ = D$. No Algoritmo 5, a_i^{T} e c_i^{T} são a i-ésima linha de A e A^{T}, respectivamente. As entradas d_{ii} de D são chamadas de pivôs e os

escalares $r_j^{(i-1)}$ e $s_j^{(i-1)}$ são os multiplicadores. Observemos que, caso A seja simétrica, então $Z = W$ e a fatoração fica como $A^{-1} = ZD^{-1}Z^{\mathsf{T}}$ ou $Z^{\mathsf{T}}AZ = D$.

Algoritmo 5: Biconjugação pela direita

Dados: matriz A $n \times n$ não-singular.
Resultado: D, Z e W tais que $A^{-1} = ZD^{-1}W^{\mathsf{T}}$.

$z_i^{(0)}, w_i^{(0)} \leftarrow e_i$, $i = 1, \ldots, n$;
para $i \leftarrow 1$ **até** n **faça**
  $z_i \leftarrow z_i^{(i-1)}$; $w_i \leftarrow w_i^{(i-1)}$;
  $d_{ii} \leftarrow a_i^{\mathsf{T}} z_i$ ou $c_i^{\mathsf{T}} w_i$;
  para $j \leftarrow i+1$ **até** n **faça**
    $r_j^{(i-1)} \leftarrow {a_i}^{\mathsf{T}} z_j^{(i-1)}$;
    $s_j^{(i-1)} \leftarrow {c_i}^{\mathsf{T}} w_j^{(i-1)}$;
    $z_j^{(i)} \leftarrow z_j^{(i-1)} - z_i \dfrac{r_j^{(i-1)}}{d_{ii}}$;
    $w_j^{(i)} \leftarrow w_j^{(i-1)} - w_i \dfrac{s_j^{(i-1)}}{d_{ii}}$;
$D \leftarrow diag(d_{11}, d_{22}, ..., d_{nn})$, $Z \leftarrow [z_1, z_2, ..., z_n]$ e $W \leftarrow [w_1, w_2, ..., w_n]$

Vejamos que é possível que d_{ii} seja zero, o que pararia o processo de execução do algoritmo. Neste caso, dizemos que o algoritmo falhou. A Proposição 1 garante uma condição necessária e suficiente para que não ocorra falha.

Proposição 1. *Seja A uma matriz não-singular de ordem n e d_{ii}'s $(1 \leq i \leq n)$ os pivôs gerados aplicando-se o Algoritmo 5 em A. Seja Δ_i o i-ésimo menor principal líder de A[1], com $\Delta_0 = 1$, então*

$$d_{ii} = \frac{\Delta_i}{\Delta_{i-1}}. \tag{2.1.1}$$

Demonstração. A demonstração é direta por indução em (2.1.1). Ver [3].

□

Portanto, de acordo com a Proposição 1, o Algoritmo 5 não falha se e somente se todos os menores principais líderes de A são não nulos.

[1] O i-ésimo menor principal líder da matriz quadrada A é o determinante da submatriz formada pelas i primeiras linhas e colunas de A.

Algoritmo 6: Biconjugação pela esquerda

Dados: matriz A $n \times n$ não-singular.
Resultado: D, Z e W tais que $A^{-1} = ZD^{-1}W^{\mathsf{T}}$.

$z_1, w_1 \leftarrow e_1$;
$d_{ii} \leftarrow a_{11}$;
para $j \leftarrow 2$ **até** n **faça**
  $z_j^{(0)}, w_j^{(0)} \leftarrow e_j$;
  para $i \leftarrow 1$ **até** $j-1$ **faça**
    $r_j^{(i-1)} \leftarrow a_i{}^{\mathsf{T}} z_j^{(i-1)}$;
    $s_j^{(i-1)} \leftarrow c_i{}^{\mathsf{T}} w_j^{(i-1)}$;
    $z_j^{(i)} \leftarrow z_j^{(i-1)} - z_i \dfrac{r_j^{(i-1)}}{d_{ii}}$;
    $w_j^{(i)} \leftarrow w_j^{(i-1)} - w_i \dfrac{s_j^{(i-1)}}{d_{ii}}$;
  $z_j \leftarrow z_j^{(j-1)}$; $w_j \leftarrow w_j^{(j-1)}$;
  $d_{jj} \leftarrow a_j{}^{\mathsf{T}} z_j$ ou $c_j{}^{\mathsf{T}} w_j$;
$D \leftarrow diag(d_{11}, d_{22}, ..., d_{nn})$, $Z \leftarrow [z_1, z_2, ..., z_n]$ e $W \leftarrow [w_1, w_2, ..., w_n]$

Comentário 1. *Se o Algoritmo 5 não falhar, temos que* $a_j^{\mathsf{T}} z_i = c_j^{\mathsf{T}} w_i = 0$, *para* $1 \leq j < i \leq n$, *e* $z_{ii} = w_{ii} = 1$, *para* $1 \leq i \leq n$, *então*

$$d_{ii} = a_i^{\mathsf{T}} z_i = w_i^{\mathsf{T}} A z_i = c_i^{\mathsf{T}} w_i. \tag{2.1.2}$$

Também temos que

$$d_{ii} = a_i^{\mathsf{T}} z_i = z_i^{\mathsf{T}} A z_i \tag{2.1.3}$$

e

$$d_{ii} = c_i^{\mathsf{T}} w_i = w_i^{\mathsf{T}} A^{\mathsf{T}} w_i. \tag{2.1.4}$$

As igualdades (2.1.3) *e* (2.1.4) *são verdadeiras pois* Z *e* W *são triangulares superiores unitárias e* AZ *e* AW *são triangulares inferiores. Mas as expressões* $a_i^{\mathsf{T}} z_i$ *e* $c_i^{\mathsf{T}} w_i$ *têm um custo menor em relação às expressões da direita. A utilização destas últimas são interessantes para evitar a falha do AINV em matrizes positivas definidas, como será explicado mais detalhadamente na Seção 2.7.*

As matrizes Z e W podem ser densas mesmo que A seja esparsa. Portanto, algumas entradas de Z e W devem ser descartadas durante a execução do algoritmo, a fim de garantir a esparsidade e tornar o método viável para matrizes de grande porte. Efetuar descartes durante o método de biconjugação é a principal ideia do AINV. Mas, primeiramente, veremos algumas características do método de biconjugação, ou seja, sem considerar os descartes.

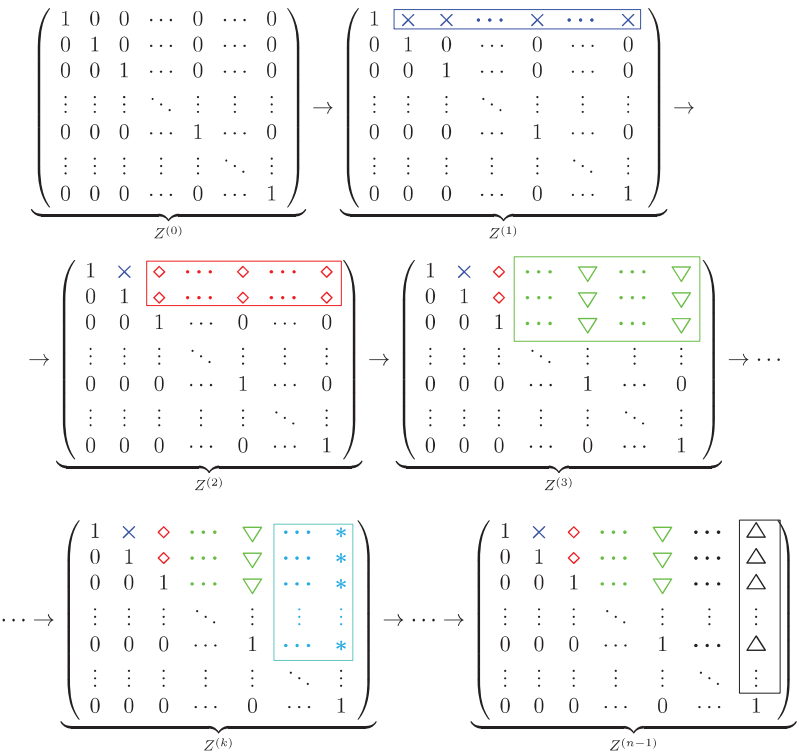

Figura 2.1: Esquema da biconjugação pela direita

2.2 Versões *pela direita* e *pela esquerda*

Existem, pelo menos, duas maneiras de se executar o processo de biconjugação, conhecidas como pela direita, Algoritmo 5, e pela esquerda, Algoritmo 6 (conhecidas na literatura como *pela direita* e *pela esquerda*, respectivamente). A diferença entre as versões pela direita e pela esquerda é como as matrizes Z e W são preenchidas. Na versão pela direita, as matrizes Z e W são preenchidas por linhas, isto é, na iteração i, os vetores z_j e w_j, com $j > i$, preenchem cada posição i. Note, porém, que a i-ésima posição de cada z_j e w_j, com $j > i$, será atualizada nas iterações seguintes. A Figura 2.1 mostra como ocorre o preenchimento da matriz Z por iteração. As regiões atualizadas da matriz estão destacadas por retângulos.

Na versão pela esquerda, as matrizes Z e W são atualizadas por colunas, isto é, na i-ésima iteração as colunas z_{i+1} e w_{i+1} são calculadas, usando os vetores $z_1, \ldots, z_i$ e $w_1, \ldots, w_i$, que são as colunas à esquerda

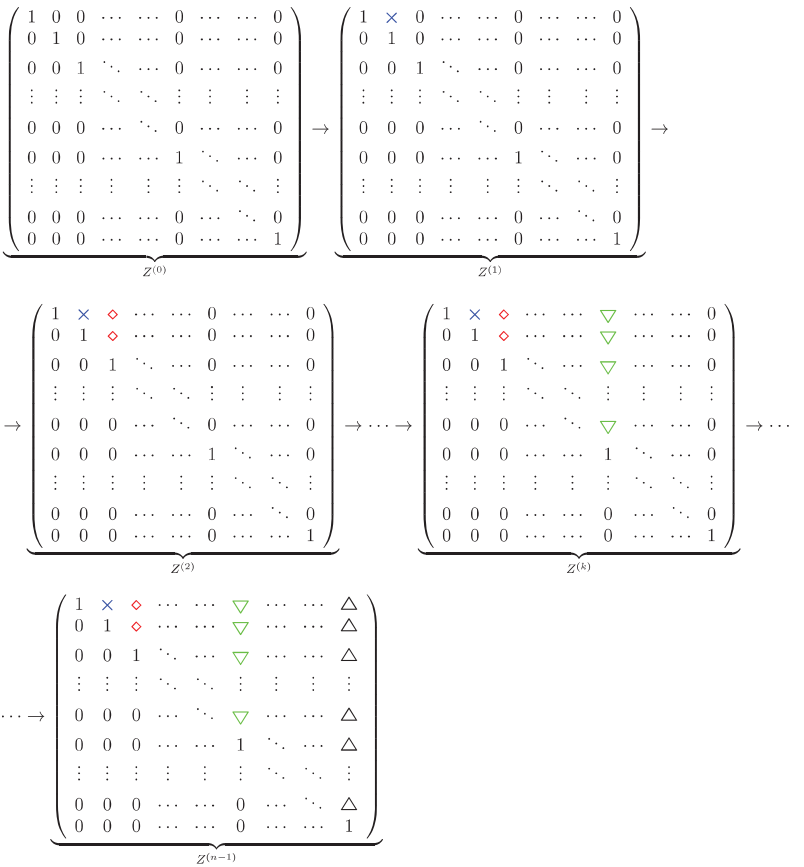

Figura 2.2: Esquema da biconjugação pela esquerda

de z_{i+1} e de w_{i+1}. Nesta alternativa, ao final da i-ésima iteração, a coluna fica em sua versão final, não sendo mais atualizada nas iterações seguintes, conforme mostra a Figura 2.2.

Proposição 2. *Os Algoritmos 5 e 6, em aritmética exata, produzem os mesmos resultados finais e intermediários.*

Demonstração. Usemos $\left(\square\right)_{\mathrm{RL}}$ para representar o vetor ou escalar gerado pela versão pela direita (Algoritmo 5) e $\left(\square\right)_{\mathrm{LL}}$ para a versão pela esquerda (Algoritmo 6).

Como d_{ii} depende de z_i (w_i) e $r_j^{(i-1)}$ ($s_j^{(i-1)}$) depende de $z_j^{(i-1)}$ ($w_j^{(i-1)}$), é suficiente demonstrar, por indução, que a hipótese H_j é verdadeira

para cada $j \in \{1, \ldots, n\}$.

$$H_j : \begin{cases} \left(z_j^{(i-1)}\right)_{\mathrm{RL}} = \left(z_j^{(i-1)}\right)_{\mathrm{LL}}, & \text{para } 1 \leqslant i \leqslant j \leqslant n; \quad (2.2.5a) \\ \left(w_j^{(i-1)}\right)_{\mathrm{RL}} = \left(w_j^{(i-1)}\right)_{\mathrm{LL}}, & \text{para } 1 \leqslant i \leqslant j \leqslant n. \quad (2.2.5b) \end{cases}$$

Primeiramente, provamos (2.2.5a): para $j = 1$, temos

$$\left(z_1^{(i-1)}\right)_{\mathrm{RL}} = \left(z_1^{(0)}\right)_{\mathrm{RL}} = e_1 = \left(z_1^{(0)}\right)_{\mathrm{LL}} = \left(z_1^{(i-1)}\right)_{\mathrm{LL}}, \ 1 \leqslant i \leqslant 1 \leqslant n.$$

Agora, pela hipótese de indução, assumimos que $H_1, H_2, \ldots, H_{j-1}$ são verdadeiras e, com isso, provamos H_j. Faremos uma segunda indução em i com j fixado. O primeiro passo, para $i = 1$, é

$$\left(z_j^{(1)}\right)_{\mathrm{RL}} = \left(z_j^{(0)}\right)_{\mathrm{RL}} - (z_1)_{\mathrm{RL}} \frac{\left(r_j^{(0)}\right)_{\mathrm{RL}}}{(d_{11})_{\mathrm{RL}}} = e_j - e_1 \frac{a_1^{\mathsf{T}} e_j}{a_1^{\mathsf{T}} e_1} = e_j - e_1 \frac{a_{1j}}{a_{11}} =$$

$$= \left(z_j^{(1)}\right)_{\mathrm{LL}}.$$

Agora, para este j fixado, supomos que $\left(z_j^{(k)}\right)_{\mathrm{RL}} = \left(z_j^{(k)}\right)_{\mathrm{LL}}$, para $1 \leqslant k < i$, obtendo assim,

$$\left(z_j^{(i)}\right)_{\mathrm{RL}} = \left(z_j^{(i-1)}\right)_{\mathrm{RL}} - (z_i)_{\mathrm{RL}} \frac{\left(r_j^{(i-1)}\right)_{\mathrm{RL}}}{(d_{ii})_{\mathrm{RL}}} = \left(z_j^{(i)}\right)_{\mathrm{LL}}.$$

Com isso, provamos (2.2.5a). A prova de (2.2.5b) é feita de forma análoga. □

Como as versões pela direita e pela esquerda se equivalem, produzindo os mesmos resultados finais e intermediários, trataremos os próximos resultados apenas utilizando a versão pela direita, Algoritmo 5, ao menos que se diga algo contrário. Os mesmos resultados podem ser obtidos utilizando a versão pela esquerda, Algoritmo 6.

2.3 Relações com os fatores L e U

Consideremos a fatoração

$$A = W^T D Z^{-1}, \tag{2.3.6}$$

obtida pelo Algoritmo 5, em que W e Z são matrizes triangulares superiores unitárias e D é matriz diagonal, todas não-singulares. Então,

diante da unicidade da fatoração LDU de A, temos que (2.3.6) corresponde a esta fatoração. Ou seja,

$$W = L^T, \quad Z = U^{-1} \tag{2.3.7}$$

e D é a mesma matriz em ambas as fatorações. Baseando-se nessas relações, obteremos os próximos resultados.

Lema 2.1. *Considerando o Algoritmo 5, para qualquer j fixo, com $1 \leqslant j \leqslant n$,*

$$z_j = z_j^{(i)} - \sum_{k=i+1}^{j-1} z_k \frac{r_j^{(k-1)}}{d_{kk}} \tag{2.3.8}$$

e

$$w_j = w_j^{(i)} - \sum_{k=i+1}^{j-1} w_k \frac{s_j^{(k-1)}}{d_{kk}}, \tag{2.3.9}$$

para $0 \leqslant i < j$.

Demonstração. As igualdades (2.3.8) ((2.3.9)) seguem diretamente das linhas 3, 4, 6 e 8 (3, 4, 7 e 9) do Algoritmo 5. □

Proposição 3. *Seja $A \in \mathbb{R}^{n\times n}$ uma matriz não-singular com todos os menores principais diferentes de zero. Seja a fatoração $A = LDU$, em que L é triangular inferior unitária, U é triangular superior unitária e D é diagonal não-singular. Com $1 \leqslant i \leqslant j \leqslant n$, temos que $L = [l_{ji}]$, $U = [u_{ij}]$ e d_{ii} são as entradas da diagonal de D. Além disso, sejam $r_j^{(i-1)}$ e $s_j^{(i-1)}$, em que $1 \leqslant i < j \leqslant n$, os escalares produzidos pelo Algoritmo 5. Então,*

$$u_{ij} = \frac{r_j^{(i-1)}}{d_{ii}} \tag{2.3.10}$$

e

$$l_{ji} = \frac{s_j^{(i-1)}}{d_{ii}}. \tag{2.3.11}$$

Demonstração. Seja e_j o j-ésimo vetor canônico, pelo Lema 2.1, temos

$$z_j = e_j - \sum_{k=1}^{j-1} z_k \frac{r_j^{(k-1)}}{d_{kk}}.$$

Fixando j, com $1 \leq i < j \leq n$, então,

$$w_i^{\mathsf{T}} A z_j = w_i^{\mathsf{T}} A e_j - \sum_{k=1}^{j-1} w_i^{\mathsf{T}} A z_k \frac{r_j^{(k-1)}}{d_{kk}}.$$

Para $j \neq i$, $w_i A z_j = 0$ e, de (2.1.2), $w_i^{\mathsf{T}} A z_i = d_{ii}$; assim,

$$0 = w_i^{\mathsf{T}} A e_j - \frac{r_j^{(i-1)}}{d_{ii}} w_i^{\mathsf{T}} A z_i \Rightarrow r_j^{(i-1)} = w_i^{\mathsf{T}} A e_j. \tag{2.3.12}$$

De acordo com (2.3.6) e (2.3.7),

$$Z^{-1} = U = D^{-1} W^{\mathsf{T}} A \Rightarrow u_{ij} = \frac{w_i^{\mathsf{T}} A e_j}{d_{ii}}. \tag{2.3.13}$$

Através de (2.3.12) e (2.3.13),

$$u_{ij} = \frac{r_j^{(i-1)}}{d_{ii}}.$$

A demonstração de que $l_{ji} = \frac{s_j^{(i-1)}}{d_{ii}}$ pode ser feita de forma análoga e é deixada a cargo do leitor, no Exercício 2.3.

□

Logo, pela Proposição 3, além das entradas de L^{-1} e U^{-1} (W e Z, respectivamente), o método de biconjugação também calcula implicitamente as entradas dos fatores L e U de A. É possível que o cálculo dessas entradas seja feita de forma explícita, adaptando o laço interno do Algoritmo 5, como pode ser visto no Algoritmo 7. Analogamente, para a versão pela esquerda, pode se adaptar o laço interno do Algoritmo 6, como é visto no Algoritmo 8.

Algoritmo 7: Cálculo das entradas dos fatores L e U no método de biconjugação pela direita

1 **para** $j \leftarrow i+1$ **até** n **faça**
2 $\quad r_j^{(i-1)} \leftarrow {a_i}^{\mathsf{T}} z_j^{(i-1)}$;
3 $\quad s_j^{(i-1)} \leftarrow {c_i}^{\mathsf{T}} w_j^{(i-1)}$;
4 $\quad l_{ji} = \frac{s_j^{(i-1)}}{d_{ii}}$;
5 $\quad u_{ij} = \frac{r_j^{(i-1)}}{d_{ii}}$;
6 $\quad z_j^{(i)} \leftarrow z_j^{(i-1)} - z_i u_{ij}$;
7 $\quad w_j^{(i)} \leftarrow w_j^{(i-1)} - w_i l_{ji}$;

Proposição 4. *Sejam d_{ii}, $r_j^{(i-1)}$'s e $s_j^{(i-1)}$'s os escalares produzidos pelo Algoritmo 5; então,*

$$d_{ii} = a_{ii} - \sum_{k=1}^{i-1} \frac{r_i^{(k-1)} s_i^{(k-1)}}{d_{kk}}, \tag{2.3.14}$$

Algoritmo 8: Cálculo das entradas dos fatores L e U no método de biconjugação pela esquerda.

para $i \leftarrow 1$ **até** $j-1$ **faça**
- $r_j^{(i-1)} \leftarrow a_i{}^{\mathsf{T}} z_j^{(i-1)}$;
- $s_j^{(i-1)} \leftarrow c_i{}^{\mathsf{T}} w_j^{(i-1)}$;
- $l_{ji} = \frac{s_j^{(i-1)}}{d_{ii}}$;
- $u_{ij} = \frac{r_j^{(i-1)}}{d_{ii}}$;
- $z_j^{(i)} \leftarrow z_j^{(i-1)} - z_i u_{ij}$;
- $w_j^{(i)} \leftarrow w_j^{(i-1)} - w_i l_{ji}$;

$$r_j^{(i-1)} = a_{ij} - \sum_{k=1}^{i-1} s_i^{(k-1)} u_{kj} \ e \tag{2.3.15}$$

$$s_j^{(i-1)} = a_{ji} - \sum_{k=1}^{i-1} r_i^{(k-1)} l_{jk}, \tag{2.3.16}$$

para $1 \leqslant i \leqslant j \leqslant n$.

Demonstração. Usando as mesmas hipóteses da Proposição 3, seja $A = LDU$, $L = [l_{ji}]$, $U = [u_{ij}]$ e d_{ii} as entradas da diagonal de D, portanto, as entradas da diagonal A, a_{ii} $(1 \leq i \leq n)$, podem ser expressadas como

$$a_{ii} = \sum_{k=1}^{i} l_{ik} d_{kk} u_{ki} = l_{ii} d_{ii} u_{ii} + \sum_{k=1}^{i-1} l_{ik} d_{kk} u_{ki}, \text{ considerando (2.3.10) e (2.3.11)},$$

$$a_{ii} = d_{ii} + \sum_{k=1}^{i-1} \frac{s_i^{(k-1)}}{d_{kk}} d_{kk} \frac{r_i^{(k-1)}}{d_{kk}} \Rightarrow d_{ii} = a_{ii} - \sum_{k=1}^{i-1} \frac{r_i^{(k-1)} s_i^{(k-1)}}{d_{kk}},$$

provando (2.3.14). Para $1 \leqslant i \leq j \leqslant n$,

$$a_{ij} = \sum_{k=1}^{i} l_{ik} d_{kk} u_{kj} = l_{ii} d_{ii} u_{ij} + \sum_{k=1}^{i-1} l_{ik} d_{kk} u_{kj}, \text{ considerando (2.3.10) e (2.3.11)},$$

$$a_{ij} = d_{ii} \frac{r_j^{(i-1)}}{d_{ii}} + \sum_{k=1}^{i-1} \frac{s_i^{(k-1)}}{d_{kk}} d_{kk} \frac{r_j^{(k-1)}}{d_{kk}} \Rightarrow r_j^{(i-1)} = a_{ij} - \sum_{k=1}^{i-1} s_i^{(k-1)} u_{kj},$$

provando (2.3.15). Para $1 \leqslant i \leq j \leqslant n$,

$$a_{ji} = \sum_{k=1}^{j} l_{jk} d_{kk} u_{ki} = l_{ji} d_{ii} u_{ii} + \sum_{k=1}^{j-1} l_{jk} d_{kk} u_{ki}, \text{ considerando (2.3.10) e (2.3.11)},$$

$$a_{ji} = d_{ii}\frac{s_j^{(i-1)}}{d_{ii}} + \sum_{k=1}^{j-1} \frac{s_j^{(k-1)}}{d_{kk}} d_{kk} \frac{r_i^{(k-1)}}{d_{kk}} \Rightarrow s_j^{(i-1)} = a_{ji} - \sum_{k=1}^{j-1} r_i^{(k-1)} l_{jk},$$

provando (2.3.16). □

Utilizando a Proposição 4, podemos calcular os $r_j^{(i-1)}$'s de forma recursiva usando os $s_i^{(k-1)}$ gerados previamente, ou seja, esses escalares podem ser calculados independente da matriz Z. Portanto, é possível calcular as entradas de U sem precisar utilizar a matriz Z. Exibimos, então, nos Algoritmos 9 e 10 as modificações nos laços internos dos Algoritmos 5 e 6, respectivamente, a fim de calcular as entradas de L e U, sem utilizar Z, de forma explícita.

Algoritmo 9: Cálculo das entradas dos fatores L e U no método de biconjugação pela direita, sem utilizar Z

1 **para** $j \leftarrow i+1$ **até** n **faça**
2 $\quad s_j^{(i-1)} \leftarrow c_i{}^{\mathsf{T}} w_j^{(i-1)}$;
3 $\quad r_j^{(i-1)} = a_{ij} - \sum_{k=1}^{i-1} s_i^{(k-1)} u_{kj}$;
4 $\quad l_{ji} = \frac{s_j^{(i-1)}}{d_{ii}}$;
5 $\quad u_{ij} = \frac{r_j^{(i-1)}}{d_{ii}}$;
6 $\quad w_j^{(i)} \leftarrow w_j^{(i-1)} - w_i^{(i-1)} l_{ji}$;

Algoritmo 10: Cálculo das entradas dos fatores L e U no método de biconjugação pela esquerda, sem utilizar Z

1 **para** $i \leftarrow 1$ **até** $j-1$ **faça**
2 $\quad s_j^{(i-1)} \leftarrow c_i{}^{\mathsf{T}} w_j^{(i-1)}$;
3 $\quad r_j^{(i-1)} = a_{ij} - \sum_{k=1}^{i-1} s_i^{(k-1)} u_{kj}$;
4 $\quad l_{ji} = \frac{s_j^{(i-1)}}{d_{ii}}$;
5 $\quad u_{ij} = \frac{r_j^{(i-1)}}{d_{ii}}$;
6 $\quad w_j^{(i)} \leftarrow w_j^{(i-1)} - w_j l_{ji}$;

O mesmo raciocínio é válido para $s_j^{(i-1)}$, W e L. Podemos modificar os laços internos dos Algoritmos 5 e 6 a fim de calcular as entradas de L e U, sem utilizar W de forma explícita.

Outra relação que destacamos aqui é entre os multiplicadores gerados no algoritmo de biconjugação e as entradas do complemento de Schur do algoritmo da eliminação gaussiana com ordem IJK, sem descartes. Essas relações foram demonstradas em [18] e são dadas por:

$$(S^{(i-1)})_{ji} = e_j A z_i^{(i-1)},\ (S^{(i-1)})_{ij} = (w_i^{(i-1)})^T A e_j, \qquad j \geq i, \quad (2.3.17)$$

onde $S^{(i-1)}$ é o complemento de Schur da matriz A gerado na i-ésima iteração da eliminação gaussiana com ordem IJK e $z_i^{(i-1)}$ e $w_i^{(i-1)}$ os vetores de Z e W gerados na i-ésima iteração do algoritmo de biconjugação. Ou seja, os vetores $(r_i^{(i-1)}, r_{i+1}^{(i-1)}, ..., r_n^{i-1})$ e $(s_i^{(i-1)}, s_{i+1}^{(i-1)}, ..., s_n^{i-1})$ formados pelos escalares da i-ésima iteração da biconjugação são, respectivamente, a primeira linha e primeira coluna transposta do complemento de Schur $S^{(i-1)}$ gerado na i-ésima iteração da eliminação gaussiana com ordem IJK.

2.4 Estratégias de descarte

Como mencionado, a proposta do AINV é de se fazer descartes durante o algoritmo de biconjugação a fim de que ele seja viável ou também para acelerar seu tempo de processamento, promovendo a esparsidade das matrizes produzidas. Existem diversas estratégias de descartes que podem ser empregadas. Consideremos $z_{kj}^{(i)}$ a entrada k do vetor $z_j^{(i)}$ e $w_{kj}^{(i)}$ a entrada k do vetor $w_j^{(i)}$ calculados nos passos 8 e 9 do Algoritmo 5 (versão pela direita), respectivamente. Abaixo listamos as principais estratégias de descartes utilizadas nos trabalhos sobre o AINV.

2.4.1 Tolerância fixa em Z e W

Este tipo de estratégia baseia-se na magnitude das entradas dos vetores de Z e W gerados durante o algoritmo. Caso, $\mid z_{kj}^{(i)} \mid < \tau$ ($\mid w_{kj}^{(i)} \mid < \tau$), em que τ é um escalar positivo (geralmente entre 0 e 1), então $z_{kj}^{(i)}$ ($w_{kj}^{(i)}$) é substituído por zero. Esse raciocínio é análogo para versão pela esquerda no Algoritmo 6.

A estratégia de tolerância fixa para o descarte predominou durante os primeiros trabalhos de inversa aproximada, sendo usada no AINV, SAINV e suas versões não simétricas, [12, 13, 14, 10, 9, 22]. A experiência da bibliografia indica o parâmetro $\tau = 0{,}1$ como suficiente para garantir a esparsidade desejada com uma qualidade aceitável (é necessário fazer um reescalamento, normalmente aquele que divide cada entrada da matriz pelo maior módulo dentre as entradas, ver detalhes na Seção 2.6). Com o progresso dos trabalhos, outras estratégias competitivas foram aparecendo, como veremos adiante.

2.4.2 Tolerância variável em Z e W

Também baseia-se na magnitude das entradas de Z e W, porém a tolerância τ pode variar durante o processo, geralmente dependendo de algum aspecto do problema. Por exemplo, a cada iteração i, pode ser verificado se $\mid w_{kj}^{(i)} \mid < \tau \parallel a_i \parallel_2$, em que $\parallel a_i \parallel_2$ é a norma dois da linha i de A e τ é um escalar positivo. Caso seja menor, a entrada $w_{kj}^{(i)}$ é descartada.

A estratégia de tolerância variável foi inicialmente proposta no contexto dos algoritmos de inversa aproximada, em 2001, em um texto que tratava da Inversa Aproximada por Blocos, o BAINV, [11]. Normalmente utilizando $\tau = 0{,}1$, o descarte por tolerância variável não recebeu muita atenção nos testes na literatura de inversa aproximada. Apenas 10 anos mais tarde recebeu uma versão diferente, utilizando linhas de W para descartar as entradas de L no contexto do RIF, [64].

2.4.3 Padrão de zeros predefinido em Z e W

Neste caso, se determinada entrada dos vetores de Z e W estiver em alguma posição considerada "desfavorável", então ela é descartada. Um exemplo seria copiar a esparsidade da matriz A, ou seja descartando $z_{kj}^{(i)}$ se sua posição em Z for a mesma posição de uma entrada nula em A.

A estratégia apareceu uma única vez na literatura no contexto de AINV e apresentou resultados pobres na comparação com BAINV, [11]. Apesar dos autores sugerirem outros padrões fixos de esparsidade, não houve indicativo de testes realizados com padrões diferentes do descrito acima.

2.4.4 Descarte em relação a $r'_j s$ e $s'_j s$

Nessa estratégia, são avaliados os valores dos multiplicadores e pivôs para a atualização de Z e W. Uma estratégia seria descartar os escalares $\frac{r_j^{(i-1)}}{d_{ii}}$ e $\frac{s_j^{(i-1)}}{d_{ii}}$ evitando, assim, a atualização dos vetores $z_j^{(i)}$ e $w_j^{(i)}$, caso esses escalares fossem menores que uma determinada tolerância. Outro critério utilizado é o descarte das entradas dos vetores $z_i \frac{r_j^{(i-1)}}{d_{ii}}$ e $w_i \frac{s_j^{(i-1)}}{d_{ii}}$ nas linhas 8 e 9 do Algoritmo 5.

Essa estratégia surgiu no contexto do AINV pelo ISAINV [36], cuja diferença era o descarte duplo. Nesses testes, o parâmetro utilizado foi de 0,1 também. Por apresentar melhores resultados que o SAINV, o algoritmo recebeu o I, de *improved*, na sigla em inglês. Já no texto de [64], os autores utilizaram uma engenhosa estratégia de descarte que explica os motivos do sucesso do descarte dos multiplicadores, já que os

relaciona com as entradas de L e U, fatores de $A = LDU$, controlando sua esparsidade e adicionou algumas ferramentas a mais para melhorar os resultados numéricos, como veremos no critério a seguir.

2.4.5 Descarte em relação a L e U

Quando são calculadas as entradas de L e U durante o algoritmo, são feitos descartes nas suas entradas. Por exemplo, l_{ji} e u_{ij} $(j > i)$ podem ser desconsideradas se sua magnitude for menor que uma tolerância fixa ou variável, que pode ser ou não igual à tolerância utilizada nas entradas de Z e W. Outro critério seria escolher parâmetros τ_1 e τ_2 tais que se $\mid lji \mid\mid\mid e_i^{\mathsf{T}} L^{-1} \mid\mid_\infty < \tau_1$ e $\mid u_{ij} \mid\mid\mid e_i^{\mathsf{T}} U^{-1} \mid\mid_\infty < \tau_2$, então l_{ji} e u_{ij} são anulados. A vantagem deste último é a preservação de informações das inversas de L e U, que podem impactar na qualidade do pré-condicionador.

2.4.6 Pós-filtragem

Trata-se de descartar algumas entradas de Z e W (ou também de L e U), após o término do algoritmo, a fim de aumentar a esparsidade dessas matrizes, caso seja conveniente. Nesse caso é utilizado um parâmetro maior que aquele utilizado durante o algoritmo.

A ideia se originou do uso dessa estratégia no contexto de outro precondicionador, o FSAI, [47]. Nesse contexto, em 1990, os autores mostraram que o uso de um filtro adicional, posterior ao cálculo dos fatores, poderia melhorar a eficiência do precondicionador.

No contexto do AINV, essa possibilidade foi testada para o SAINV e para o SBAINV em [11]. O que se observou nos testes foi que o SAINV é bastante influenciado pela pós-filtragem e pode reduzir o número de iterações do método iterativo, como se pode observar no gráfico:

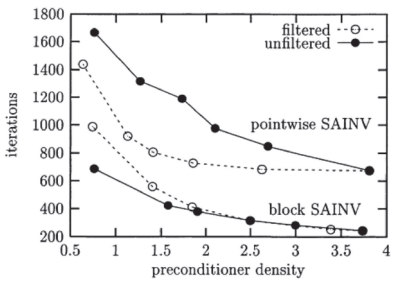

Figura 2.3: Efeito da pós-filtragem. Fonte: [11]

No caso da figura 2.3, podemos observar os resultados nos testes aplicados na matriz S3RMT3M3, do Matrix Market[54]. Os valores dos parâmetros utilizados para o SBAINV são:

- Tolerâncias de descarte sem a pós-filtragem: 1e-1; 4e-2; 3e-2; 2,5e-2 e 2e-2;
- Tolerâncias utilizadas na pós-filtragem: 3e-1; 2e-1; 1,5e-1; 1e-1 e 5e-2.

Para o SAINV, os valores são:

- Tolerâncias de descarte sem a pós-filtragem: 6e-2; 3,5e-2; 2,5e-2; 2e-2 e 1,5e-2;
- Tolerâncias utilizadas na pós-filtragem: 4e-1; 2e-1; 1,5e-1; 1e-1 e 6e-2.

Observe que para o caso bloco, filtrar produziu mais iterações.

2.5 Apresentação do Algoritmo Inversa Aproximada

Nos Algoritmos 11 e 12 apresentamos as versões pela esquerda e pela direita do AINV, respectivamente, com os possíveis descartes.

2.6 Reordenamentos e escalamentos

Na literatura de Inversa Aproximada é comum encontrar diferentes possibilidades para reordenamentos e escalamentos. Nesta seção, nosso objetivo é apresentar possibilidades e explicar como os escalamentos são feitos. Os reordenamentos serão apenas apresentados sem as explicações matemáticas, já que são encontradas em literatura específica.

Abaixo listaremos as possibilidades e, em seguida, vamos apresentá-los com mais detalhes:

Escalamentos

- Máximo: basta identificar qual é o maior elemento da matriz, isto é, $max \mid a_{ij} \mid = \alpha$ para $1 \leq i \leq n$ e $1 \leq j \leq n$ e fazer utilizar $\frac{1}{\alpha}A$, A é a matriz de coeficientes de ordem $n \times n$.
- Jacobi: para cada elemento $a_{ii} \neq 0$ da diagonal de A, com $1 \leq i \leq n$, multiplica-se todos os elementos da linha i por $1/a_{ii}$, no caso de A ser não simétrica. Se A for simétrica, a fim de se manter a simetria, multiplica-se a linha i e a coluna i de A por $\frac{1}{\sqrt{a_{ii}}}$.

Algoritmo 11: AINV pela direita

Dados: matriz A $n \times n$ não-singular.
Resultado: D, Z e W tais que $A^{-1} \approx ZD^{-1}W^{\mathsf{T}}$.

$z_i^{(0)}, w_i^{(0)} \leftarrow e_i$, $i = 1, \ldots, n$;
para $i \leftarrow 1$ **até** n **faça**
  $z_i \leftarrow z_i^{(i-1)}$; $w_i \leftarrow w_i^{(i-1)}$;
  $d_{ii} \leftarrow a_i^{\mathsf{T}} z_i^{(i-1)}$ ou $c_i^{\mathsf{T}} w_i^{(i-1)}$;
  para $j \leftarrow i + 1$ **até** n **faça**
    $r_j^{(i-1)} \leftarrow a_{i-1}^{\mathsf{T}} z_j^{(i-1)}$;
    $s_j^{(i-1)} \leftarrow c_{i-1}^{\mathsf{T}} w_j^{(i-1)}$;
    Se necessário, descartar $r_j^{(i-1)}$ ou $s_j^{(i-1)}$ (opcional);
    $z_j^{(i)} \leftarrow z_j^{(i-1)} - z_i \frac{r_j^{(i-1)}}{d_{ii}}$;
    $w_j^{(i)} \leftarrow w_j^{(i-1)} - w_i \frac{s_j^{(i-1)}}{d_{ii}}$;
    Aplicar descarte nas entradas de $z_j^{(i)}$ e $w_j^{(i)}$;
Aplicar descarte nas entradas de Z e W (pós-filtragem opcional);
$D \leftarrow diag(d_{11}, d_{22}, ..., d_{nn})$, $Z \leftarrow [z_1, z_2, ..., z_n]$, e $W \leftarrow [w_1, w_2, ..., w_n]$

- Bloco Jacobi: se A for não simétrica, é feito $\hat{A} = G^{-1}A$, tal que

$$G = diag(A_{11}, A_{22}, ..., \ldots, A_{NN}),$$

em que A_{II} são os blocos-diagonais de A. Para o caso simétrico, primeiro calcula os fatores de Cholesky dos blocos diagonais de A, isto é, $A_{II} = L_I L_I^T$ e depois faz $\hat{A} = G^{-1}AG^{-T}$, aonde

$$G = diag(L_1.L_2, \ldots, L_N).$$

Este escalamento foi fortemente recomendado no caso bloco do AINV no texto de [11]. Nele se concluiu que o uso do escalamento Bloco Jacobi é superior aos demais, como mostra o resultado na Figura 2.4.

Nesse exemplo, a matriz usada foi a S3DKT3M2 do Matrix Market, ψ é tolerância para o descarte e ρ é a densidade do precondicionador. O número de iterações se refere ao uso do método iterativo CG precondicionado.

Reordenamentos:

- Approximate Minimum Degree Algorithm (AMD) - tirado de [4].
- MultiLevel Nested Dissection reordering (MNLD) - tirado de [45] e [46].

Algoritmo 12: AINV pela esquerda

Dados: matriz A $n \times n$ não-singular.
Resultado: D, Z e W tais que $A^{-1} \approx ZD^{-1}W^{\mathsf{T}}$.

$z_1, w_1 \leftarrow e_1$;
$d_{11} \leftarrow a_{11}$;
para $j \leftarrow 2$ **até** n **faça**
  $z_j^{(0)}, w_j^{(0)} \leftarrow e_j$;
  para $i \leftarrow 1$ **até** $i-1$ **faça**
    $r_j^{(i-1)} \leftarrow a_i^{\mathsf{T}} z_j^{(i-1)}$;
    $s_j^{(i-1)} \leftarrow c_i^{\mathsf{T}} w_j^{(i-1)}$;
    Se necessário, descartar $r_j^{(i-1)}$ ou $s_j^{(i-1)}$ (opcional);
    $z_j^{(i)} \leftarrow z_j^{(i-1)} - z_i \frac{r_j^{(i-1)}}{d_{ii}}$;
    $w_j^{(i)} \leftarrow w_j^{(i-1)} - w_i \frac{s_i^{(j-1)}}{d_{ii}}$;
    Aplicar descarte nas entradas de $z_j^{(i)}$ e $w_j^{(i)}$;
  $z_j \leftarrow z_j^{(j-1)}$; $w_j \leftarrow w_j^{(j-1)}$;
  $d_{jj} \leftarrow a_j^{\mathsf{T}} z_j$ ou $c_j^{\mathsf{T}} w_j$;
Aplicar descarte nas entradas de Z e W (pós-filtragem opcional);
$D \leftarrow diag(d_{11}, d_{22}, ..., d_{nn})$, $Z \leftarrow [z_1, z_2, ..., z_n]$ e $W \leftarrow [w_1, w_2, ..., w_n]$

- Minimum Degree (MIP) - tirado de [21].
- Directed Component Relaxation (DCR) - tirado de [51].

2.7 Falha na Inversa Aproximada

Vimos que o método de biconjugação pode falhar caso ocorram pivôs nulos ou muito pequenos, mas que seria garantida a não falha caso os menores principais da matriz A fossem não nulos. Porém, ao se executar o AINV, os descartes podem impactar nos valores dos pivôs, sendo possível que ocorram pivôs nulos ou muito pequenos, mesmo se os menores principais de A forem não nulos. Por exemplo, sem os descartes, o método não falha para matrizes SPD, pelo critério de Sylvester, mas com os descartes, ele pode falhar. Entretanto, para algumas famílias de matrizes, o AINV não falha, como veremos a seguir. Primeiramente, vejamos algumas definições.

Definição 2.1. *Uma matriz $A \in \mathbb{R}^{n \times n}$, $A = [a_{ij}]$, é estritamente dia-*

Block SAINV, unscaled			Block SAINV, J. scaled			Block SAINV, block J. scaled		
ψ	ρ	Iterations	ψ	ρ	Iterations	ψ	ρ	Iterations
0.5	1.98	>10 000	0.5	1.95	5905	0.25	1.52	3969
0.5	1.75	1892	0.5	1.47	1499	0.25	1.02	1535
0.5	3.12	379	0.5	1.34	672	0.25	0.76	689
0.5	2.76	500	0.5	1.33	688	0.25	0.76	673

Figura 2.4: Comparação entre os diferentes escalamentos. Fonte: [11].

gonal dominante se

$$| a_{ii} | > \sum_{j \neq i} | a_{ij} | .$$

Definição 2.2. *Uma matriz quadrada A é uma Z-matriz se $a_{ij} \leq 0, \ \forall\, i \neq j$.*

Definição 2.3. *Uma Z-matriz A é uma M-matriz se ela pode ser escrita como $A = sI - B$ para alguma matriz $B \geq 0$ e algum escalar $s \geqslant \rho(B)$, em que $\rho(B)$ denota o raio espectral de B. Nós denotamos por $\mathcal{M}$ o conjunto de todas as M-matrizes.*

Definição 2.4 (Matriz comparação, ver [17])**.** *Seja a matriz A, definimos as entradas c_{ij} da sua matriz* comparação $\mathcal{C}(A)$ *como:*

$$c_{ij} = \begin{cases} -|a_{ij}|, & se\ i \neq j \\ |a_{ii}|, & se\ i = j \end{cases} .$$

Por definição, $\mathcal{C}(A)$ é Z-matriz, para qualquer matriz A.

Definição 2.5 (H-matriz)**.** *Uma matriz é uma H-matriz se sua matriz comparação for uma M-matriz. Ver [17]. Neste caso, a sua matriz comparação é a sua M-matriz associada.*

A seguir temos dois resultados que asseguram que o AINV não falha para duas das classes de matrizes definidas acima.

Proposição 5. *Seja A uma M-matriz e d_{ii}'s os pivôs produzidos pelo AINV aplicado em A, sem executar descartes. E sejam $\bar{d_{ii}}$'s os pivôs produzidos pelo AINV aplicado em A, considerando descartes. Então, para todo $1 \leqslant i \leqslant n$,*

$$\bar{d_{ii}} \geq d_{ii} > 0.$$

Demonstração. A demonstração se encontra em [12, 13]. □

Proposição 6. *Seja A uma H-matriz e $\hat{A}$ a sua M-matriz associada. Sejam d_{ii}'s e $\hat{d}_{ii}$'s os pivôs gerados pelo AINV, sem executar descartes, aplicado em A e $\hat{A}$, respectivamente. Então $d_{ii} \geq \hat{d}_{ii}$, para todo $1 \leqslant i \leqslant n$. Além disso, sejam $\bar{d}_{ii}$'s os pivôs calculados pelo AINV aplicado a A, levando em conta os descartes. Então $\bar{d}_{ii} \geq \hat{d}_{ii}$.*

Demonstração. A demonstração se encontra em [12, 13]. □

As proposições acima nos dizem que os pivôs gerados pelo AINV em matrizes do tipo H e M são sempre positivos, garantindo que ele não falhe (mas podem ocorrer pivôs de magnitude muito baixa). De acordo com [52], qualquer matriz diagonal dominante é uma matriz H. Assim sendo, se o processo de biconjugação falhar, uma alternativa, sugerida em [10], seria selecionar um escalar $\alpha > 0$ e aplicar novamente o algoritmo na matriz $A^{'} = A + \alpha I$. É desejável que α seja grande o bastante para evitar a falha, mas pequeno o suficiente para que $A^{'}$ seja próximo de A, evitando precondicionadores de baixa qualidade. Para matrizes mal condicionadas, esse procedimento geralmente fornece precondicionadores ruins. Além disso, a falha pode ocorrer perto do final do processo de biconjugação, e ele pode ter que ser recalculado várias vezes antes que o valor satisfatório de α seja encontrado. Uma outra estratégia seria realizar modificações na diagonal principal apenas quando a necessidade surgir, modificando os pivôs se sua magnitude for menor que um limite especificado.

Uma forma de evitar a falha para matrizes SPD é utilizar o método de "redução diagonalmente compensada", formulado por Axelsson [6] e Axelsson e Kolotilina [7]. O objetivo desta técnica é associar qualquer matriz SPD A a uma determinada M-matriz SPD $\hat{A}$, através de um processo de redução das entradas positivas que estão fora da diagonal e compensação dessas entradas nas entradas da diagonal de A. Benzi, Cullum e Tůma descrevem, em [10], a variação mais simples deste método, que baseia-se em substituir por zero as entradas positivas a_{ij} fora da diagonal de A (redução) e adicionar às correspondentes entradas da diagonal a_{ii} (compensação diagonal). Para isso, A é escrita como

$$A = B + R,$$

onde R contém apenas as respectivas entradas positivas de A fora da diagonal e fazer

$$\hat{A} = B + \Delta,$$

tal que Δ é matriz diagonal satisfazendo

$$\Delta e = Re,$$

onde e é o vetor formado por 1's. Logo,

$$\hat{A} = A + (\Delta - R).$$

Temos que $\Delta - R$ é uma M-matriz simétrica singular, já que suas entradas de fora da diagonal não são positivas e a soma de todos os elementos de cada linha sempre dá zero (ver [16]). Em particular, $\Delta - R$ é positiva semidefinida, o que faz com que $\hat{A}$ seja positiva definida. Além disso, como $\hat{A}$ não possui entradas positivas fora da diagonal, logo ela é uma M-matriz (ver [16]). A ideia, então, é aplicar o AINV na matriz $\hat{A}$ (sendo livre de falha, pois $\hat{A}$ é uma M-matriz) e utilizar o precondicionador gerado $M \approx \hat{A}^{-1}$ no sistema original $Ax = b$. É esperado que M seja um bom precondicionador para A, pois A está próxima de ser uma M-matriz. No trabalho, eles estimam essa proximidade através de um parâmetro η que é dado por $\eta = \frac{\|R\|_F}{\|A\|_F}$, sendo $\|\cdot\|_F$ a norma de Frobenius e R a matriz descrita acima. Observemos que $0 \leq \eta < 1$ e que $\eta = 0$ se e, somente se, A for uma M-matriz.

Para matrizes positivas definidas, Benzi e Tůma sugeriram, em [10], uma modificação no cálculo dos pivôs a fim de assegurar que eles sejam positivos, garantindo que o processo não falhe. Originalmente, os pivôs são calculados como $d_{ii} = a_i^{\mathsf{T}} z_i$ ou $d_{ii} = c_i^{\mathsf{T}} w_i$. Em aritmética exata, temos que $a_i^{\mathsf{T}} z_i = z_i^{\mathsf{T}} A z_i$ e $c_i^{\mathsf{T}} w_i = w_i^{\mathsf{T}} A^{\mathsf{T}} w_i$, como visto no Comentário 1, nas expressões (2.1.3) e (2.1.4). Porém, ao utilizarmos os descartes nas entradas dos vetores z_i e w_i, as igualdades (2.1.3) e (2.1.4) não são necessariamente verdadeiras e, desta forma, pode-se gerar pivôs nulos ao utilizarmos as expressões do lado esquerdo de cada igualdade. Portanto, para garantir que o método não falhe, podemos escolher d_{ii} como sendo $z_i^{\mathsf{T}} A z_i$ ou $w_i^{\mathsf{T}} A^{\mathsf{T}} w_i$, já que A é positiva definida e, por isso, tais expressões sempre resultarão em valores positivos. A sugestão, então, é substituir as linhas 4 e 11 dos Algoritmos 11 (pela direita) e 12 (pela esquerda) pelas expressões

$$d_{ii} \leftarrow (z_i^{(i-1)})^{\mathsf{T}} A z_i^{(i-1)} \text{ ou } (w_i^{(i-1)})^{\mathsf{T}} A w_i^{(i-1)}, \tag{2.7.18}$$

$$d_{jj} \leftarrow (z_j^{(j-1)})^{\mathsf{T}} A z_j^{(j-1)} \text{ ou } (w_j^{(j-1)})^{\mathsf{T}} A w_j^{(j-1)}, \tag{2.7.19}$$

respectivamente.

Estas fórmulas foram bastante utilizadas em trabalhos posteriores sobre variações do AINV, para evitar a falha do algoritmo com matrizes positivas definidas.

Exercício 2.1. *Reescreva de forma simplificada os Algoritmos de Biconjugação pela direita e pela esquerda (Algoritmos 5 e 6), caso A seja uma matriz simétrica $n \times n$ não-singular.*

Exercício 2.2. *Seja A seja uma matriz não-singular, com menores principais diferentes de zero, tal que*

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}.$$

Utilizando o Algoritmo de Biconjugação, encontre a fatoração $A^{-1} = ZD^{-1}W^T$ *da inversa de* A*, em que* D *é diagonal e* Z *e* W *são triangulares superior e inferior unitárias. A partir do resultado, também encontre a fatoração de* A*, tal que* $A = LDU$*, em que* D *é diagonal,* L *é triangular inferior unitária e* U *é triangular superior unitária.*

Exercício 2.3. *Considere as condições da Proposição 3. Mostre que*

$$s_j^{(i-1)} = z_i^{\mathsf{T}} A^{\mathsf{T}} e_j \tag{2.7.20}$$

e, a partir disso, mostre que

$$l_{ji} = \frac{s_j^{(i-1)}}{d_{ii}}.$$

Exercício 2.4. *Assim como nos Algoritmos 9 e 10, reescreva os laços internos dos Algoritmos 5 e 6 a fim de calcular as entradas de* L *e* U*, sem utilizar* W *de forma explícita.*

Capítulo 3

Principais variações da Inversa Aproximada

Neste capítulo, faremos uma breve descrição de cada uma das principais variações do AINV encontradas na literatura. Analisaremos cada variação com o mesmo enfoque aplicado pelos autores de cada trabalho.

3.1 Inversa Aproximada (AINV)

O AINV foi originalmente proposto por Benzi, Meyer e Tůma em [12], em 1996, para matrizes simétricas e definidas positivas (SPD). O método calcula a fatoração da inversa aproximada de A, fornecendo as matrizes Z e D, tais que $A^{-1} \approx ZD^{-1}Z^T$. O algoritmo é apresentado na sua versão pela direita, como no Algoritmo 5 (sem fazer o cálculo de W). No trabalho, eles demonstram as Proposições 5 e 6, garantindo a que ele não falha para matrizes M e H.

Nos testes numéricos apresentados, eles utilizam o AINV para gerar precondicionadores de diferentes matrizes para o método dos Gradientes Conjugados (CG) [42]. Eles utilizaram o escalamento máximo e reordenamento MMD. Como critérios de descarte, eles utilizaram o de tolerância para as entradas de Z, com valor de tolerância indo de $0,1$ até $0,6$ e também padrão de zeros pré-definido em Z. Eles também cogitaram não efetuar a atualização dos vetores de Z durante o algoritmo, caso $\frac{r_j^{(i-1)}}{p_i}$ tivesse magnitude muito baixa. Porém, tal procedimento produziu resultados numéricos insatisfatórios, sendo, assim, desconsiderado.

Como as matrizes utilizadas não eram necessariamente do tipo M e H, foi necessária a utilização de estratégias para evitar a falha, além de restringir a ocorrência de elementos muito grandes em Z. Se algum pivô p_i fosse menor que $\sqrt{\varepsilon_M}$, em que ε_M é a precisão de máquina, então ele poderia ser substituído por

Quadro Resumitivo	
Autores e Ano:	Benzi, Meyer e Tüma - 1996
Entrada:	Matriz esparsa A SPD
Saídas:	Z e D tais que $ZD^{-1}Z^{\mathsf{T}} \approx A^{-1}$.
Robustez:	Matrizes M e H
Na Literatura	
Método iterativo:	Gradientes Conjugados
Descartes:	Tolerância (normalmente 0,1) ou Estrutura de Zeros Pré-definida
Escalamento:	Máximo
Reordenamento:	MMD

Tabela 3.1: AINV

$$p_i \longleftarrow \max\{\sqrt{\varepsilon_M}, \mu\sigma\theta\},$$

onde

$$\sigma = \max_{i\leq k\leq n-1}\{r_k^{(i-1)}\}, \theta = \|z_i^{(i-1)}\|_\infty \neq 0$$

e $\mu = 0,1$ é um parâmetro de relaxação. Entretanto, eles ressaltaram que tais modificações poderiam afetar a qualidade do precondicionador. Os testes foram feitos comparando o desempenho do AINV com o precondicionador de Cholesky incompleto (IC) [71] para o CG, desde que as densidades dos precondicionadores estivessem próximas. O lado direito dos testes foi calculada a partir do vetor solução tendo todas as entradas iguais a 1. O critério de parada utilizado foi quando a norma euclidiana do resíduo não precondicionado ficasse menor que 10^{-9}. Os autores concluíram que o precondicionador AINV tem a mesma robustez que o IC, sendo competitivo com ele. Eles reforçaram que o AINV pode ser uma forte ferramenta na resolução de sistemas lineares SPD esparsos e de grande porte em arquiteturas modernas de alto desempenho. Na Tabela 3.1 exibimos um resumo das principais características deste trabalho.

Comentário 2. *Aqui nos referimos como AINV ao algoritmo proposto por Benzi, Meyer e Tůma em [12], que é o algoritmo para matrizes simétricas. Porém, no restante do trabalho o AINV faz referência ao caso geral, para qualquer matriz, como é exposto no Algoritmo 5, ao menos que especifiquemos que seja para o caso simétrico.*

3.2 Inversa Aproximada Não Simétrica (AINV-NS)

A Inversa Aproximada Não Simétrica (*Nonsymmetric Approximate Inverse* ou simplesmente AINV-NS) foi proposto por Benzi e Tůma em [13], em 1998. Trata-se de uma versão mais generalizada da sugerida em [12], pois pode ser aplicado a qualquer matriz quadrada inversível, não necessariamente simétrica. O método fornece as matrizes Z, W e D, tais que $A^{-1} \approx ZD^{-1}W^T$. O AINV-NS já foi descrito no Capítulo 2 deste trabalho, correspondendo ao Algoritmo 5. A versão utilizada foi a pela direita, mas eles também citaram a versão pela esquerda como opção (Algoritmo 6). No artigo, eles reafirmaram que o algoritmo não falha para matrizes M e H, indicando que a demonstração é análoga à desenvolvida em [12].

Nos testes, foi utilizado o valor de tolerância fixo como critério de descarte das entradas de Z e W. Eles ressaltaram que quanto menor o valor de tolerância, maior a densidade do precondicionador e menor o número de iterações na fase de resolução do sistema. Porém, nesse caso, a execução do AINV poderia ser lenta. Mas, se fossem efetuados muitos descartes nas matrizes Z e W, a qualidade do precondicionador poderia não ser satisfatória. Eles escolheram, então, ajustar a tolerância de modo que o número de não zeros do precondicionador fosse próximo ao número de não zeros de A. Na maioria das vezes, o valor escolhido foi de $0,1$. Em alguns casos não foi possível encontrar uma tolerância que resultasse em precondicionadores com densidades próximas de A, gerando precondicionadores muito densos ou muito esparsos.

Foram comparados resultados do AINV-NS com os do precondicionador baseado na fatoração ILU(0) [68] e os métodos de Krylov foram BIGSTAB, QMR e GMRES (ver métodos em [8]). O lado direito foi calculado usando o vetor solução com todas as entradas iguais a 1. O critério de parada utilizado foi quando a norma euclidiana do resíduo não precondicionado ficasse menor que 10^{-8}. Eles também utilizaram o escalamento máximo e reordenamento MMD. A partir dos testes, os autores concluíram que o AINV-NS acelerou a convergência de diversos métodos iterativos, podendo ser comparada, em média, com a taxa de convergência obtidas pelo ILU(0). Eles também mencionaram que o tempo de construção do precondicionador do AINV-NS foi maior que do ILU(0), porém seu custo não seria algo proibitivo. Na Tabela 3.2 exibimos um resumo das principais características deste trabalho.

Quadro Resumitivo	
Autores e Ano:	Benzi e Tüma - 1998
Entrada:	Matriz esparsa A não simétrica e não singular
Saídas:	Z, W e D tais que $ZD^{-1}W^{\mathsf{T}} \approx A^{-1}$.
Robustez:	Matrizes M e H
Na Literatura	
Método iterativo:	BICGSTAB, QMR e GMRES
Descartes:	Tolerância (normalmente 0,1) ou Estrutura de Zeros Pré-definida
Escalamento:	Máximo
Reordenamento:	MMD

Tabela 3.2: AINV-NS

3.3 Inversa Aproximada Estabilizada (SAINV)

A Inversa Aproximada Estabilizada (*Stabilized Approximate Inverse* ou simplesmente SAINV) foi desenvolvido por Benzi e Tůma em [10] para matrizes simétricas, em 2000, e seu principal diferencial é o fato dela não falhar para matrizes SPD quaisquer. Isso é possível pois o SAINV é semelhante ao AINV (para matrizes simétricas de [12]), com a única diferença no cálculo dos pivôs que é dado por $d_{ii} = z_i^T A z_i$, como na equação (2.7.18), garantindo ser livre de falha para matrizes SPD.

Como o SAINV calcula o pivô utilizando produtos matriz-vetor, é natural que haja questionamento sobre seu custo, já que ele é necessariamente maior que o do AINV (para matrizes simétricas de [12]) original, pois este utiliza produtos vetor-vetor. Porém, os autores afirmaram que o cálculo do pivô no SAINV é viável e com custo não muito maior que do AINV. Eles argumentaram que os vetores z_i são esparsos devido aos descartes, além da matriz A também ser esparsa e, portanto, os n produtos matriz-vetor de cada iteração são feitos no modo "esparso×esparso". Além disso, como Z é triangular superior, apenas as i primeiras colunas de A entram nesse produto, a cada iteração i. Eles afirmaram que ao descartar as entradas de z_i de modo que a matriz final Z possua $O(n)$ entradas não nulas e assumindo uma distribuição uniforme dessas entradas nas colunas de Z, o custo para calcular os d_{ii}'s na forma de expressões bilineares esparsas envolvendo A é linear.

Nos testes feitos, os autores compararam o SAINV com os precondicionadores de Jacobi, FSAI [47] e AINV para o método CG. No AINV, foi utilizada a estratégia de redução diagonalmente compensada (descrita na Seção 2.7) afim de se evitar a falha. A estratégia de descarte

utilizado no SAINV foi o de tolerância nas entradas de Z, com valor de $0,1$. O critério de parada utilizado foi quando a norma euclidiana do resíduo inicial fosse menor que 10^{-8} ou quando o método realizasse mais de 10.000 iterações. O lado direito foi construído como $b = Ax$, aonde x é um vetor com entradas randômicas e uniformemente distribuídas no intervalo (0,1). Eles também utilizaram escalamento Jacobi e reordenamento MMD. A partir dos testes, os autores concluíram que o SAINV é uma opção de precondicionador robusto e eficaz para ser utilizado no CG, especialmente se combinado com o escalamento Jacobi e o reordenamento MMD. O AINV combinado com a redução diagonalmente compensada também se mostrou ser um precondicionador confiável, mas não tão eficaz quanto o SAINV, com a possível exceção em problemas de difusão. Na Tabela 3.3 exibimos um resumo das principais características deste trabalho.

Quadro Resumitivo	
Autores e Ano:	Benzi, Cullum e Tüma - 2000
Entrada:	Matriz esparsa A SPD
Saídas:	Z e D tais que $ZD^{-1}Z^{\mathsf{T}} \approx A^{-1}$.
Robustez:	Matrizes SPD
Na Literatura	
Método iterativo:	CG
Descartes:	Tolerância (normalmente 0,1) ou Estrutura de Zeros Pré-definida
Escalamento:	Jacobi
Reordenamento:	MMD

Tabela 3.3: SAINV

3.4 Inversa Aproximada Estabilizada Não Simétrica (SAINV-NS)

Em 2000, Bridson e Tang apresentaram a Inversa Aproximada Estabilizada Não Simétrica (*Stabilized Nonsymetric Approximate Inverse* ou simplesmente SAINV-NS), em [22]. Eles afirmam que o algoritmo proposto é baseado no SAINV e pode ser aplicado a qualquer matriz simétrica ou não simétrica. As diferenças em relação ao AINV são os cálculos $r_j^{(i-1)}$, $s_j^{(i-1)}$ e d_{ii}. Os escalares $r_j^{(i-1)}$, $s_j^{(i-1)}$ são calculados utilizando as equações $r_j^{(i-1)} = w_i^T A e_j$ e $s_j^{(i-1)} = z_i^T A^T e_j$ (equações (2.3.12) e (2.7.20)). Ou seja, dado o Algoritmo 6, as linhas 6 e 7 são substituídas pelas equações (2.3.12) e (2.7.20), respectivamente. Já o pivô é

dado por $d_{jj} = w_j^T A z_j$, substituindo a linha 11 do Algoritmo 6 por esta expressão.

Nos testes, a estratégia de descarte utilizada foi de tolerância fixa no valor de $0,1$ nas entradas de z_j e w_j e também nas entradas dos vetores $z_i \frac{r_j^{(i-1)}}{p_i}$ e $w_i \frac{s_j^{(i-1)}}{q_i}$ das linhas 8 e 9 do Algoritmo 6, respectivamente. Eles utilizaram escalamento o Jacobi e o reordenamento MIP. Os precondicionadores gerados foram utilizados para os métodos CG, no caso SPD, e BIGSTAB, para os demais tipos de matrizes. O lado direito foi calculado usando o vetor solução com todas as entradas iguais a 1 e o critério de parada utilizado foi quando a norma euclidiana do resíduo ficasse menor que 10^{-6}. Eles compararam as versões pela esquerda e pela direita do SAINV-NS, concluindo que os precondicionadores melhoraram os desempenhos dos métodos iterativos, porém a versão pela direita foi mais rápida na fase de construção. Além disso eles também ressaltaram que o reordenamento da inversa aproximada melhorou a eficiência da memória cache durante a execução do método iterativo. Na Tabela 3.4 exibimos um resumo das principais características deste trabalho.

Quadro Resumitivo	
Autores e Ano:	Bridson e Tang - 2000
Entrada:	Matriz esparsa A não simétrica e não singular.
Saídas:	Z, W e D tais que $ZD^{-1}W^{\mathsf{T}} \approx A^{-1}$.
Robustez:	Matrizes M e H
Na Literatura	
Método iterativo:	BICGSTAB
Descartes:	Tolerância (normalmente 0,1)
Escalamento:	Jacobi
Reordenamento:	MIP

Tabela 3.4: NS-SAINV

3.5 Inversa Aproximada com Pivoteamento (AINVP)

Em 2002, Bollhofer e Saad propuseram a Inversa Aproximada com Pivoteamento (AINVP) (*Approximate Inverse with Pivoting* ou simplesmente AINVP) para quaisquer matrizes, em [19]. A ideia sugerida é aplicar o processo de pivoteamento completo na versão pela direita do AINV, baseado no processo de pivoteamento completo da eliminação gaussiana com ordem IJK [67]. A principal motivação do uso de pivo-

teamento é evitar o aparecimento de pivôs nulos ou de baixa magnitude durante o algoritmo, ou seja, evitar a falha ou a instabilidade. Aqui comentamos, de forma generalizada, o processo de pivoteamento completo utilizado. A ideia é, a cada iteração, permutar o pivô d_{ii} com outro multiplicador calculado na iteração, dada uma condição predeterminada, com intuito de evitar a falha. Para preservar a consistência do método, deve-se também reordenar as linhas e colunas de A e as colunas de $Z - I$ e $W - I$. Para realizar as permutações em A, são utilizadas as matrizes permutação Π e Σ, que no início do algoritmo são dadas pela identidade I. Então, a cada iteração, é executada a etapa de pivoteamento e depois as colunas de Z e W são atualizadas na forma usual do AINV (linhas 8 e 9 do Algoritmo 5). O algoritmo de biconjugação é executado na matriz $\Pi A\Sigma$ e, ao final do método, a inversa aproximada fica como $(\Pi A\Sigma)^{-1} \approx Z^T D^{-1} W^T$. O Algoritmo 13 apresenta uma implementação do AINVP.

A seguir, detalhamos as etapas do Algoritmo 13, a cada iteração i:

- São obtidos $s_i^{(i-1)}, ..., s_n^{(i-1)}$ a partir de $s_j^{(i-1)} = w_j^T(\Pi A\Sigma)z_i$, $j \geq i$ (linha 7).

- Caso $|s_i^{(i-1)}| < \alpha \max_{m \geq i+1} |s_m^{(i-1)}|$ (condição de pivoteamento), para um parâmetro $\alpha \in (0, 1]$ escolhido previamente, então é selecionado k, tal que $|s_k^{(i-1)}| = \max_{m \geq i+1} |s_m^{(i-1)}|$. Assim, são permutados os elementos $s_i^{(i-1)}$ e $s_k^{(i-1)}$. Além disso, são permutadas as colunas i e k de $W - I$ e as linhas i e k de Π (linha 8 até 11).

- São obtidos $r_i^{(i-1)}, ..., r_n^{(i-1)}$ a partir de $r_j^{(i-1)} = w_i^T(\Pi A\Sigma)z_j$, $j \geq i$ (linha 13).

- Caso $|r_i^{(i-1)}| < \alpha \max_{m \geq i+1} |r_m^{(i-1)}|$ (condição de pivoteamento), para um parâmetro $\alpha \in (0, 1]$ escolhido previamente, então é escolhido k, tal que $|r_k^{(i-1)}| = \max_{m \geq i+1} r_m^{(i-1)}$. Assim, são permutados os elementos $r_i^{(i-1)}$ e $r_k^{(i-1)}$. Além disso, são permutadas as colunas i e k de $Z - I$ e Σ (linha 14 ate 17).

- Se o pivô $r_i^{(i-1)}$ for permutado (e, consequentemente, as linhas e colunas das matrizes), então os valores $s_i^{(i-1)}, ..., s_n^{(i-1)}$ devem ser recalculados e novamente aplicada a condição de pivoteamento a $s_i^{(i-1)}$. De forma análoga, se o pivô $s_i^{(i-1)}$ for permutado, então os valores $r_i^{(i-1)}, ..., r_n^{(i-1)}$ devem ser recalculados e novamente aplicada a condição de pivoteamento a $r_i^{(i-1)}$. Esse processo é executado até que a condição de pivoteamento seja totalmente atendida (isso é feito utilizando as variáveis booleanas satisfied$_r$ e satisfied$_s$).

- Por fim, são atualizados os valores dos vetores $z_j^{(i)}$ e $w_j^{(i)}$ para $j \geq i$ e aplicadas estratégias de descarte em suas entradas (linha 22 até 24).

Podemos observar que o AINVP é bem mais complexo e custoso que o $AINV$ original, sendo interessante no uso de problemas difíceis de convergirem ou para se evitar a falha, já que seu objetivo é evitar pivôs nulos ou de baixa magnitude.

Nos testes numéricos, os valores escolhidos para α foram de $0,1$ ou 1. Os descartes foram feitos por magnitude, usando tolerâncias de $\tau = 0,1$ ou $\tau = 0,01$. Eles compararam o AINVP com o ILU e ILUP para os métodos GMRES(30) e QMR. O critério de parada utilizado foi quando o método atingisse um máximo de 500 iterações, ou se a norma relativa do resíduo ficasse menor do que $\sqrt{eps}$ ou o próprio eps, aonde eps indica a precisão da máquina. O lado direito foi calculado usando o vetor solução com todas as entradas iguais a 1. Os autores concluíram que o AINVP produz precondicionadores bastante robustos, principalmente em problemas difíceis. Na Tabela 3.5 exibimos um resumo das principais características deste trabalho.

Quadro Resumitivo	
Autores e Ano:	Bollhofer e Saad - 2002
Entrada:	Matriz esparsa A não singular e não simétrica.
Saídas:	Z, W, Π, Σ e D tais que $ZD^{-1}W^{\mathsf{T}} \approx (\Pi A \Sigma)^{-1}$..
Robustez:	–
Na Literatura	
Método iterativo:	GMRES(30) e QMR
Descartes:	Tolerância (entre 0,01 e 0,1)
Escalamento:	–
Reordenamento:	–

Tabela 3.5: AINVP

3.6 Fatoração Incompleta Robusta (RIF)

Em 2003, Benzi e Tůma propuseram a Fatoração Incompleta Robusta (RIF) (*Robust Incomplete Factorization* ou simplesmente RIF), em [15]. Primeiramente, os autores apresentaram um problema de mínimos quadrados que busca resolver

$$\min_{x \in \mathbb{R}^n} \|b - Ax\|_2, \tag{3.6.1}$$

onde A é esparsa, $m \times n$ e possui posto completo. Naturalmente, o problema vale para o caso $m = n$, mas o maior interesse é quando $m > n$. Para resolver (3.6.1) podemos usar métodos diretos ou métodos iterativos aplicados implicitamente às equações normais do problema, dadas por

$$Cx = f, \tag{3.6.2}$$

onde

$$C = A^T A, f = A^T b.$$

Neste caso, eles escolheram o CGLS [35] como método iterativo, que é uma variação do CG. Os autores consideraram, então, produzir precondicionadores para o CGLS, ressaltando o interesse de que esse precondicionador tivesse as seguintes propriedades:

1 nenhuma entrada de $C = A^t A$ precisaria ser explicitamente calculada;

2 a fatoração incompleta não falharia;

3 O armazenamento intermediário seria insignificante.

Então eles apresentaram RIF, cuja ideia é utilizar o SAINV para obter a fatoração LDL de C. Seja $C = LDL^T$ a fatoração LDL de C (fatoração de Cholesky sem do cálculo de raízes quadradas), em que L é diagonal inferior unitária. Então $C^{-1} = L^{-T} D^{-1} L^{-1}$ e, portanto, $Z = L^{-T}$ (como visto na Seção 2.3). Sabemos que, para $L = [l_{ji}]$, então $l_{ji} = \frac{r_j^{(i-1)}}{d_{ii}}$ (ver equação (2.3.11), considerando o caso simétrico em que $Z = W$ e, então, $r_j^{(i-1)} = s_j^{(i-1)}$). Logo, para calcular as entradas de L, bastaria armazenar os multiplicadores $r_j^{(i-1)}$ durante o SAINV aplicado à C. Este procedimento pode ser visto no Algoritmo 7, considerando apenas o cálculo das entradas de L e Z. A fatoração LDL aproximada seria, então, utilizada como precondicionador para (3.6.2).

Como o processo é baseado no SAINV, então o algoritmo é livre de falha pois $C = A^T A$ é SPD. Eles ressaltaram esse aspecto como uma vantagem em relação a outros processos que usam a fatoração LDL da matriz como precondicionador. Em relação às características requeridas, temos que a primeira propriedade é garantida, já que durante o processo, o cálculo dos multiplicadores e dos pivôs se baseiam no produto $z_i^T C z_j = z_i^T A^T A z_j = (Az_i)^T (Az_j)$, $1 \leq i \leq j \leq n$. Portanto, $A^T A$ não precisa ser explicitamente calculada. O segundo item é garantido pois o RIF não falha quando aplicado a matrizes SPD. O terceiro item também é alcançado, já que não é necessário armazenar os vetores intermediários e nem as colunas de Z gerados durante o algoritmo.

Para os testes numéricos, o critério de descarte utilizado foi de tolerância fixa τ nas entradas de Z e de L. Eles também sugeriram o uso da tolerância variável $\tau\|a_i\|_2$ como opção, em que a_i é a i-ésima coluna de A. Neste caso, eles recomendavam que se executasse um reordenamento em A de modo que $\|a_i\|_2 = 1$ (e, portanto, C possuiria diagonal unitária). Eles afirmaram que tal estratégia poderia melhorar o condicionamento da equação normal. Para simplificar, eles escolheram o mesmo valor de tolerância para os vetores z_i e para as entradas de L que variou entre $0,1$ e $0,5$. Eles utilizaram o escalamento Jacobi e o reordenamento MMD. O RIF foi comparado com os precondicionadores ICNE [15], IMGS [1] e IGR [60] para serem aplicados no CGLS. O lado direito foi calculado usando o vetor solução com todas as entradas iguais a 1. O algoritmo pararia quando a norma euclidiana do resíduo relativo ficasse menor que 10^{-8}. Os autores concluíram que, embora o RIF fosse mais custoso que os outros precondicionadores, os resultados numéricos indicaram que, em muitas vezes, o RIF obteve melhores taxas de convergência e maior estabilidade em relação à falha, dependendo do valor de descarte utilizado. Na Tabela 3.6 exibimos um resumo das principais características deste trabalho.

Quadro Resumitivo	
Autores e Ano:	Benzi e Tüma - 2003
Entrada:	Matriz esparsa A não simétrica e não singular
Saídas:	L, U e D tais que $LDU \approx A$.
Robustez:	Matrizes SPD
Na Literatura	
Método iterativo:	CGLS
Descartes:	Tolerância (normalmente entre 0,1 e 0,5)
Escalamento:	Jacobi
Reordenamento:	MMD

Tabela 3.6: RIF

3.7 Inversa Aproximada Estabilizada Aperfeiçoada (ISAINV)

Em 2004, Seiji Fujino e Yusuki Ikeda apresentaram a Inversa Aproximada Estabilizada Aperfeiçoada (*Improved Stabilized Approximate Inverse* ou simplesmente ISAINV), em [36] para matrizes SPD. Eles se basearam na versão pela direita do SAINV, porém a diferença seria a utilização de uma dupla estratégia de descarte. Antes de serem feitos

descartes nas entradas dos vetores $z_j^{(i-1)}$ por tolerância fixa τ_1, primeiramente era avaliado o valor de $|\frac{r_j^{(i-1)}}{d_{ii}}|$ através de uma tolerância τ_2. Caso $|\frac{r_j^{(i-1)}}{d_{ii}}| \leq \tau_2$, o vetor $z_j^{(i)}$ não seria atualizado. Caso contrário, a atualização do vetor era feita como no SAINV e, então, seriam aplicados os descartes nas entradas de $z_j^{(i)}$. Este processo de duplo descarte pode ser visto no Algoritmo 11, aplicando o descarte na linha 6 em $\frac{r_j^{(i-1)}}{d_{ii}}$ (já que estamos lidando com o caso simétrico). Eles sugeriram essa mesma estratégia de duplo descarte para o RIF, chamando-o de IRIF ("Improved RIF").

Nos testes, os valores das tolerâncias variaram de $0,01$ até $0,75$. Eles utilizaram o escalamento Jacobi. Foram comparados os precondicionadores gerados pelo SAINV, RIF, ISANV e IRIF para o método CG. O lado direito foi calculado usando o vetor solução com todas as entradas iguais a 1 ou um vetor realístico. O algoritmo pararia quando a norma euclidiana do resíduo relativo ficasse menor que 10^{-9}. Os autores concluíram que a estratégia de descarte duplo funcionou bem para matrizes obtidas de uma ampla gama de aplicações, com melhor desempenho em relação ao SAINV e ao RIF. Eles também pontuaram que utilizaram apenas uma mesma máquina para todo os testes, ressaltando que as comparações poderiam ser fortemente influenciadas pela plataforma de computação utilizada. Porém, eles destacaram a melhora de desempenho com a estratégia de descarte duplo se comparada, pelo menos, com outras estratégias de descarte. Na Tabela 3.7 exibimos um resumo das principais características deste trabalho.

Quadro Resumitivo	
Autores e Ano:	Fujino e Ikeda. - 2004
Entrada:	Matriz esparsa A SPD
Saídas:	Z e D tais que $ZD^{-1}Z^{\mathsf{T}} \approx A^{-1}$.
Robustez:	Matrizes SPD
Na Literatura	
Método iterativo:	CG
Descartes:	Descarte duplo
Escalamento:	Jacobi
Reordenamento:	–

Tabela 3.7: ISAINV

3.8 Inversa Aproximada Estabilizada Variada (SAINV-VAR)

A Inversa Aproximada Estabilizada Variada (*Stabilized Approximate Inverse Variety* ou simplesmente SAINV-VAR) foi elaborada por Rafiei e Toutounian em [65], em 2008, para ser aplicado em quaisquer matrizes positivas definidas. Eles se basearam na versão pela direita do SAINV e nas relações dos fatores Z, W com os fatores L e U de A. A ideia proposta é utilizar o SAINV para produzir as aproximações das matrizes W, U e D. A aproximação de U é obtida através da igualdade $u_{ij} = \frac{r_j^{(i-1)}}{d_{ii}}$ (equação (2.3.10)), ou seja, utilizando os multiplicadores gerados pelo algoritmo. Além disso, os multiplicadores $r_j^{(i-1)}$ seriam calculados como $r_j^{(i-1)} = a_{ij} - \sum_{k=1}^{i-1} s_i^{(k-1)} u_{kj}$ (equação (2.3.15)). Dessa forma, não seria necessária a utilização de Z para calculá-los. O algoritmo é feito tomando como base o Algoritmo 5, mas reescrevendo o laço "para"do j pelo laço mostrado no Algoritmo 9, excluindo-se a linha 4 (as entradas da aproximação de L não são calculadas). Os pivôs são calculados como no SAINV.

Para achar a aproximação de Z, foi feita a aproximação da inversa de U, pois $Z = U^{-1}$. Para isso, eles fizeram uso do trucamento da série de Neumman de grau l:

$$Z_l = I + F + F^2 + F^3 + ... + F^l \tag{3.8.3}$$

onde $F = I - U$ e I é a matriz identidade. Por meio de (3.8.3) e do método de Horner [43], obtemos a inversa aproximada $A^{-1} \approx ZD^{-1}W_l$. No trabalho, foi escolhido $l = 4$. A principal vantagem do algoritmo seria evitar aplicar a conjugação em A^T, calculando a aproximação de Z apenas após o processo, a partir de U.

Os descartes foram feitos nas entradas dos vetores de W, em U e, posteriormente, em Z, utilizando tolerâncias fixas dadas por $\tau_1 = 0,001$, $\tau_2 = 0,001$ e $\tau_3 = 0,01$, respectivamente. Os testes numéricos compararam o desempenho de SAINV-VAR com o SAINV (adaptado para matrizes não simétricas, ou seja, sendo equivalente ao AINV-NS mas utilizando $d_{ii} = z_i^T A z_i$ para o calculo dos pivôs) para os métodos QMR, BIGSTAB e GMRES(10). O lado direito foi calculado usando o vetor solução com todas as entradas iguais a 1. O algoritmo pararia quando a norma euclidiana do resíduo ficasse menor que 10^{-6}. Os autores concluíram nos testes que o SAINV-VAR foi eficaz na redução do número de iterações, além de ter sido mais esparso e mais barato (tanto nos custos de construção quanto nos custos totais) do que o SAINV. Na Tabela 3.8 exibimos um resumo das principais características deste trabalho.

Quadro Resumitivo	
Autores e Ano:	Rafiei e Toutonian - 2008
Entrada:	Matriz esparsa A NSPD
Saídas:	U, W e D tais que $U_l^{-1}D^{-1}W^{\mathsf{T}} \approx A^{-1}$, aonde U^{-1} é estimado pela série de Newmann: $U_l^{-1} = I + (I-U) + \ldots + (I-U)^l$.
Robustez:	Matrizes NSPD
Na Literatura	
Método iterativo:	QMR, BICSTAB e GMRES(10)
Descartes:	Tolerância (entre 0,001 e 0,01)
Escalamento:	–
Reordenamento:	–

Tabela 3.8: SAINV-VAR

3.9 Inversa Aproximada Fatorada pela Frente (FFAPINV)

Em 2010, Salkuyeh propôs em [69] a Inversa Aproximada Fatorada pela Frente (*Forward Factored Approximate Inverse* ou simplesmente FFAPINV), baseada no de Lee e Zhang [50]. O FFAPINV pode ser utilizado em qualquer matriz não singular e tem como base a versão pela esquerda do AINV-NS (o termo "pela frente"neste trabalho é equivalente ao "pela esquerda"). A diferença é que os multiplicadores são calculados utilizando as equações $r_j^{(i-1)} = w_i^T A e_j$ e $s_j^{(i-1)} = z_i^T A^T e_j$ (equações (2.3.12) e (2.7.20)). Ou seja, dado o Algoritmo 6, as linhas 6 e 7 são substituídas pelas equações (2.3.12) e (2.7.20), respectivamente. O cálculo do pivô é feito como $d_{jj} = w_j^T A z_j$ se A não for positiva definida. Se ela for positiva definida, então $d_{jj} = A z_j$ se $A z_j$ for diferente de zero. Caso contrário, $d_{jj} = z_j^T A z_j$ (equação (2.7.19)), garantindo que o algoritmo não falhe para qualquer matriz positiva definida. Salkuyeh também mostrou em um trabalho anterior [70] que, sem considerar os descartes, o AINV e o FFAPINV são matematicamente equivalentes.

A estratégia de descarte foi aplicada em duas partes. Caso, $|\ \frac{r_j^{(i-1)}}{d_{ii}}\ | < \tau$ e $|\ \frac{s_j^{(i-1)}}{d_{ii}}\ | < \tau$, para uma dada tolerância τ então as atualizações de $z_j^{(i)}$ e $w_j^{(i)}$ nas linhas 8 e 9 não eram efetuadas. A segunda estratégia foi feita nos vetores $z_j^{(i)}$ e $w_j^{(i)}$, desconsiderando as entradas cujo valor absoluto fossem menores que τ. O valor de tolerância utilizado foi $\tau = 0,1$. Nos testes, foram comparados os precondicionadores FFA-

PINV, SAINV (adaptado para matrizes não simétricas, ou seja, sendo equivalente ao AINV-NS mas utilizando $d_{ii} = z_i^T A z_i$ para o calculo dos pivôs) e SAINV-VAR para o solver GMRES. O lado direito era construído com a solução sendo o vetor com entradas iguais a 1. O critério de parada foi de 10^{-10} para a norma euclidiana do resíduo ou se o número máximo de 1000 iterações foi atingido. Os autores concluíram que o FFAPINV foi eficaz na melhora do número de iterações e do tempo total total de CPU para atingir a convergência. Eles também ressaltaram que o FFAPINV obteve melhores resultados que o SAINV e SAINV-VAR. Na Tabela 3.9 exibimos um resumo das principais características deste trabalho.

Quadro Resumitivo	
Autores e Ano:	Salkuyeh - 2010
Entrada:	Matriz esparsa A NSPD
Saídas:	Z, W e D tais que $ZD^{-1}W^{\mathsf{T}} \approx A^{-1}$.
Robustez:	Matrizes NSPD
Na Literatura	
Método iterativo:	GMRES
Descartes:	Descarte duplo (no valor de $0,1$)
Escalamento:	–
Reordenamento:	–

Tabela 3.9: FFAPINV

3.10 Fatoração Incompleta Robusta Não Simétrica (RIF-NS)

Em 2011, Rafiei e Bollhöfer propuseram a Fatoração Incompleta Robusta Não Simétrica (*Nonsymmetric Robust Incomplete Factorization* ou simplesmente RIF-NS), em [64], que utiliza o AINV para encontrar aproximações dos fatores LDU de qualquer matriz não singular. Eles apontam que a vantagem de se utilizar o RIF-NS reside no fato dele ser não falhar se a matriz for positiva definida. Nesse caso, o pivô seria calculado como em (2.7.18).

Primeiramente, eles tratam da sua versão pela direita que é baseada no AINV pela direita. A ideia é calcular as matrizes W e D utilizando o processo de biconjugação, e as matrizes L e U por meio das fórmulas $l_{ji} = \frac{s_j^{(i-1)}}{d_{ii}}$ e $u_{ij} = \frac{r_j^{(i-1)}}{d_{ii}}$ (equações (2.3.11) e (2.3.10)), com $j \geqslant i$. O cálculo de $s_j^{(i-1)}$ é feito da foma usual como $c_i^T w_j^{(i-1)}$ (linha 7 do Algoritmo 5). Porém, os multiplicadores $r_j^{(i-1)}$ são calculados como

(ver equação (2.3.15)):

$$r_j^{(i-1)} = r_j^{(i-1)} = a_{ij} - \sum_{k=1}^{i-1} s_i^{(k-1)} u_{kj} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} p_k u_{kj}.$$

Portanto, para achar os multiplicadores $r_j^{(i-1)}$ e, consequentemente encontrar U, necessita-se apenas nos valores das entradas de A e das entradas de L e U previamente calculados. Ou seja, não é preciso calcular Z para se obter os multiplicadores, evitando, assim, trabalhar com A^T. Desta forma, a cada iteração i, são encontradas a i-ésima coluna de W, a i-ésima coluna de L e a i-ésima linha de U. Este processo é observado no Algoritmo 9.

Como estratégia de descarte, Rafiei e Bollhöfer usam um mesmo parâmetro ε_{LW} para L e W e um parâmetro ε_U para U, tais que w_{lj}, l_{ji} e u_{ij} são descartados, caso

$$|w_{lj}| < \varepsilon_{LW},$$
$$|l_{ji}| \|e_i^T L^{-1}\|_\infty < \varepsilon_{LW}, \tag{3.10.4}$$
$$|u_{ij}| \|e_i^T U^{-1}\|_\infty < \varepsilon_U, \tag{3.10.5}$$

onde w_{lj} é a entrada l do vetor w_j na iteração i, com $1 \leq l < i < j \leq n$. As relações (3.10.4) e (3.10.5) são obtidas afim de se preservar informações das inversas de L e U, importantes para a qualidade do precondicionador. Mas, como não se tem as inversas dessas matrizes e $W = L^{-T}$, sem os descartes, então eles sugeriram adaptar a relação (3.10.4) para

$$|l_{ji}| \|w_i\|_\infty < \varepsilon_{LW}, \tag{3.10.6}$$

já que $\|w_i\|_\infty \approx \|L^{-T} e_i\|_\infty = \|e_i^T L^{-1}\|_\infty$. Desta forma, a condição (3.10.6) pode ser utilizada, a cada iteração, no descarte das entradas de L. Como as entradas de Z não são calculadas no algoritmo, eles utilizam um método alternativo (ver [64]) que estima o valor aproximado de $\|e_i^T U^{-1}\|_\infty$, necessitando das linhas de U. Esse método é rodado paralelamente ao RIF-NS a fim de se aplicar o descarte nas entradas de U a cada iteração. Na versão pela esquerda, a ideia é semelhante, porém com alguns ajustes em relação aos índices.

Nos testes foram comparados RIF-NS pela direita, RIF-NS pela esquerda, AINV pela esquerda e ILUT para os métodos BICGSTAB, GMRES(30) e TFQMR [34]. Foram feitos escalamento NSSM (ver [64]) e reordenamento MNLD. O valor inicial nos testes foi o vetor nulo e lado direito foi b, em que $b = Ae$, sendo e um vetor com estradas iguais a 1. O algoritmo pararia quando a norma euclidiana do resíduo relativo ficasse menor que 10^{-10}. Os autores concluíram que a versão pela esquerda do RIF-NS, na maioria dos testes, obteve menores tempos

totais e fizeram convergir o método de Krylov com menor número de iterações do que o AINV. Já o ILUT, na maioria dos testes, obteve menores tempos totais e menor número de iterações do que o RIF-NS. Portanto, eles concluíram que a versão pela esquerda do RIF-NS obteve melhores resultados que o AINV, porém com resultados um pouco inferiores ao ILUT. Na Tabela 3.10 exibimos um resumo das principais características deste trabalho.

Quadro Resumitivo	
Autores e Ano:	Rafiei e Bollhöfer - 2011
Entrada:	Matriz esparsa A não simétricas e não singulares
Saídas:	L, U e D tais que $LDU \approx A$.
Robustez:	Matrizes NSPD
Na Literatura	
Método iterativo:	BICGSTAB, GMRES(30) e TFQMR
Descartes:	Tolerância em Z, W, L e U
Escalamento:	NSSM
Reordenamento:	MLND

Tabela 3.10: RIF-NS

3.11 Fatoração Incompleta Robusta com Pivoteamento (RIFP)

Em 2012, Rafiei propôs a Fatoração Incompleta Robusta com Pivoteamento (*Robust Incomplete Factorization with Pivoting* ou simplesmente RIFP) em [62], que é uma adaptação do processo de pivoteamento do AINVP para o RIF. Ela é proposta para matrizes não singulares quaisquer. O pivoteamento é feito de maneira análoga à descrita na Seção 3.5, porém também são executadas permutações nos elementos doa fatores L e U de A.

Aqui comentamos, de forma generalizada, o processo de pivoteamento utilizado. A cada iteração i, se $s_i^{(i-1)}$ não atender à condição de pivoteamento e tiver de ser permutado com o multiplicador $s_k^{(i-1)}$, então, além de serem permutadas as colunas i e k de $W - I$ e as linhas i e k de Π, também devem ser permutadas as linhas de $L - I$ (linha 8 até 11). Da mesma forma, se $r_i^{(i-1)}$ não atender à condição de pivoteamento e tiver que ser permutado com o multiplicador $r_k^{(i-1)}$, então, além de serem permutadas as colunas i e k de $Z - I$ e de Σ, também devem ser permutadas as colunas de $U - I$ (linha 14 até 17). Ao final do processo de pivoteamento, são calculadas as entradas de L

e U através das equações (2.3.11) e (2.3.10) (linhas 22 e 23), respectivamente, e atualizados os vetores de Z e W (linhas 25 e 26). Depois, são efetuadas estratégias de descartes nesses valores (linhas 24 e 27). O algoritmo produz a fatoração $\Pi A \Sigma \approx LDU$. Podemos observar o RIFP no Algoritmo 14.

Nos testes numéricos, os valores escolhidos para α foram de $\alpha = 0,1$, $\alpha = 0,4$, $\alpha = 0,75$ e $\alpha = 1$. Os descartes foram feitos por magnitude usando tolerâncias de $\tau = 0,1$. Eles compararam com RIFP com a versão pela direita do RIF para para os métodos GMRES(30), BICGSTAB e TFQMR. Em todos os testes, o lado direito foi construído com o vetor solução tendo todas as entradas iguais a 1. O critério de parada foi de 10^{-8} para a norma do resíduo relativo ou quando o método atingisse um máximo de 5.000 iterações. Eles utilizaram reordenamento MMD. Os autores concluíram que o RIFP obteve bons resultados ao diminuir o número de iterações dos métodos de Krylov. Eles também ressaltaram que o valor de $\alpha = 0,1$ melhorou a qualidade dos precondicionadores, se comparado a outros valores de α. Na Tabela 3.11 exibimos um resumo das principais características deste trabalho.

Quadro Resumitivo	
Autores e Ano:	Rafiei - 2012
Entrada:	Matriz esparsa A não simétrica e não singular
Saídas:	Π, Σ, L, U e D tais que $LDU \approx \Pi A \Sigma$.
Robustez:	–
Na Literatura	
Método iterativo:	GMRES(30), BICGSTAB e TFQMR
Descartes:	Tolerância (normalmente 0,1)
Escalamento:	–
Reordenamento:	MMD

Tabela 3.11: RIFP

3.12 Inversa Aproximada com Pivoteamento pela Esquerda (LLAINVP)

Em 2014, Rafiei propôs a Inversa Aproximada com Pivoteamento pela Esquerda (*Left Looking Approximate Inverse with Pivoting* ou simplesmente LLAINVP) para matrizes não singulares quaisquer, em [63]. Baseada no trabalho de Bollhofer e Saad [19], que aplica estratégia de pivoteamento completo na versão pela direita do AINV, ele sugere também o pivoteamento completo, só que na versão pela esquerda. Assim como

no AINVP, a estratégia é baseada no pivoteamento completo da versão eliminação gaussiana com ordem IJK [67].

Aqui comentamos, de forma generalizada, o processo de pivoteamento completo para o AINV pela esquerda. O processo é análogo ao AINVP, (descrito na Seção 3.5), cuja ideia é permutar o pivô $r_j(s_j)$ com outro multiplicador $r_j^{(k)}(s_j^{(k)})(i < k < n)$ calculado a cada iteração, para atender condição predeterminada. Para preservar a consistência do método, deve-se também reordenar as linhas e colunas de A e as colunas de $Z - I$ e $W - I$. Para realizar as permutações em A, são utilizadas matrizes de permutação Π e Σ, que são inicializadas como matrizes identidade I. Apesar do processo ser análogo, existem algumas diferenças, inerentes ao fato de se trabalhar com a versão pela esquerda. Uma delas é o cálculo dos multiplicadores $s_j^{(j-1)}, ..., s_j^{(n)}$. Na versão pela direita, esse cálculo se dá através de $\Pi A\Sigma$ e dos vetores de W e Z da iteração anterior. Porém, na versão pela esquerda, esses vetores ainda não foram calculados. Então utiliza-se a fórmula $s_i^{(j-1)} = e_i^T(\Pi A\Sigma)z_j^{(j-1)}$, $(j \leq i)$ (equação (2.7.20)) para achar tais valores. Para isso, os vetores $z_j^{(1)}, ..., z_j^{(i-1)}$ são previamente atualizados, como no Algoritmo 6. Outra diferença, é que as colunas j e k de $W - I$ não são diretamente permutadas, já que a coluna k de W ainda não foi calculada. Portanto, para que isso ocorra, os vetores $w_j^{(1)}, ..., w_j^{(j-1)}$ são atualizados como no Algoritmo 6 depois de se fazer as permutações dos multiplicadores e das linhas de A. Essas diferenças também valem para os multiplicadores $r_j^{(j-1)}, ..., r_j^{(n)}$ e a matriz Z. O algoritmo produz a fatoração $(\Pi A\Sigma)^{-1} \approx ZD^{-1}W^T$. O LLAINVP é executado como no Algoritmo 15.

Abaixo explicamos, de forma geral, as etapas do Algoritmo 15 a cada iteração j:

- São atualizados os vetores $z_j^{(1)}, ..., z_j^{(j-1)}$ como no Algoritmo 6 e é aplicada estratégia de descarte (linhas 7 e 8).

- São obtidos valores de $s_j^{(j-1)}, ..., s_n^{(j-1)}$ através da equação $s_i^{(j-1)} = e_i^T(\Pi A\Sigma)z_j^{(j-1)}, (j \geq i)$ (se não for a primeira iteração, então $s_j^{(j-1)} = r_j^{(j-1)}$) (linha 9 até 14).

- Caso $|s_j^{(j-1)}| < \alpha \max_{m \geq i+1} |s_m^{(j-1)}|$ (condição de pivoteamento), para um parâmetro $\alpha \in (0, 1]$ escolhido previamente, então é selecionado k, tal que $|s_k^{(j-1)}| = \max_{m \geq j+1} |s_m^{(j-1)}|$. Assim, são permutados os elementos $s_j^{(j-1)}$ e $s_k^{(j-1)}$. Além disso, são permutadas as linhas i e k de Π (linha 15 até 20).

- São atualizados os vetores $w_j^{(1)}, ..., w_j^{(j-1)}$ como no Algoritmo 6 e é aplicada estratégia de descarte (linhas 25 até 26).

- É feito $r_j^{(j-1)} = s_j^{(j-1)}$ e são obtidos valores de $r_{j+1}^{(j-1)}, ..., r_n^{(j-1)}$ através da equação $r_i^{(j-1)} = (w_j^{(j-1)})^T(\Pi A \Sigma)e_i^T, (j \geq i)$ (linha 27 até 28).

- Caso $|r_j^{(j-1)}| < \alpha \max_{m \geq i+1} |r_m^{(j-1)}|$ (condição de pivoteamento), para um parâmetro $\alpha \in (0, 1]$ escolhido previamente, então é escolhido k, tal que $|r_k^{(j-1)}| = \max_{m \geq j+1} r_m^{(j-1)}$. Assim, são permutados os elementos $r_j^{(j-1)}$ e $r_k^{(j-1)}$. Além disso, são permutadas as colunas i e k de Σ (linha 30 até 34).

- Se o pivô $r_j^{(j-1)}$ for permutado (e, consequentemente, as linhas e colunas das matrizes), então $z_j^{(1)}, ..., z_j^{(j-1)}$ e $s_j^{(j-1)}, ..., s_n^{(j-1)}$ devem ser recalculados e novamente aplicada a condição de pivoteamento a $s_j^{(j-1)}$. De forma análoga, se o pivô $s_j^{(j-1)}$ for permutado, então os vetores $w_j^{(1)}, ..., w_j^{(j-1)}$ e os valores $r_j^{(j-1)}, ..., r_n^{(j-1)}$ devem ser recalculados e novamente aplicada a condição de pivoteamento a $r_j^{(j-1)}$. Esse processo é executado até que a condição de pivoteamento seja totalmente atendida (isso é feito utilizando as variáveis booleanas satisfied$_r$ e satisfied$_s$).

Observemos que o algoritmo é executado na matriz $\Pi A \Sigma$ e, dessa forma, a inversa aproximada fica como $(\Pi A \Sigma)^{-1} \approx ZD^{-1}W^T$. É enfatizado que a principal motivação do uso de pivoteamento é evitar o aparecimento de pivôs nulos ou de baixa magnitude durante a execução do algoritmo, contribuindo para que as entradas de Z e W não sejam muito grandes, melhorando a qualidade do precondicionador. Porém, o processo de construção do precondicionador é mais complexo que o $AINV$ original, sendo interessante no uso de problemas difíceis que falham na fase de construção do precondicionador ou que não convergem na fase de aplicação do método de Krylov.

A estratégia de descarte foi feita em relação às entradas dos vetores w_i e z_i por tolerância fixa variando entre $0,1$ e $0,01$. O parâmetro α foi testado para os valores $0,1$, $0,5$, $0,8$ e 1. Os resultados foram comparados com o AINV pela esquerda e os precondicionadores usados para os métodos BICGSTAB e TFQMR. O lado direito utilizado foi construído utilizando o vetor solução com todas as entradas iguais a 1. O critério de parada foi de 10^{-8} para a norma do resíduo relativo ou quando o método atingiu um máximo de 5.000 iterações. Foi utilizado ordenamento MNLD. Os autores concluíram que quando a tolerância foi de $0,1$, os testes com o LLAINVP obtiveram melhores resultados

no uso dos métodos iterativos quando $\alpha = 0,1$. Já quando a tolerância de descarte foi de $0,01$, os testes não indicaram uma melhor escolha para α. Eles também ressaltaram que os resultados do LLAINVP sem o MNLD não foram melhores do que utilizando o MNLD, no intuito de diminuir o número de iterações do método iterativo. Na Tabela 3.12 exibimos um resumo das principais características deste trabalho.

Quadro Resumitivo	
Autores e Ano:	Rafiei - 2014
Entrada:	Matriz esparsa A não simétricas e não singulares
Saídas:	Z, W, Π, Σ e D tais que $ZD^{-1}W^{\mathsf{T}} \approx (\Pi A \Sigma)^{-1}$.
Robustez:	–
Na Literatura	
Método iterativo:	BICGSTAB e TFQMR
Descartes:	Tolerância (entre 0,01 e 0,1)
Escalamento:	–
Reordenamento:	MNLD

Tabela 3.12: LLAINVP

3.13 Classificação da Inversa Aproximada

Agrupamos em quatro classes as diversas variantes da Inversa Aproximada encontradas na literatura. Essa separação foi percebida, principalmente, durante a implementação das mesmas e leva em consideração elementos como: tipo de armazenamento das matrizes calculadas nos algoritmos, uso de memória compartilhada e a forma como alguns cálculos são efetuados. Esperamos com a presente seção clarificar as semelhanças e diferenças entre as versões do algoritmo de inversa aproximada e estabelecer estruturas gerais a partir das quais os algoritmos serão considerados casos particulares.

As quatro classes gerais dos algoritmos de Inversa Aproximada encontradas na literatura são: Classe AINV, Classe FFAPINV, Classe AINV-LU e Classe Pivoteamento. Nas subseções a seguir, exploraremos os aspectos de cada uma dessas classes.

3.13.1 Classe AINV

Essa classe é formada pelas variações: AINV, AINV-NS, SAINV e ISAINV. Elas têm como base o Algoritmo 5. Nelas, os vetores de Z e W são atualizados da mesma maneira, como apresentado neste algoritmo. Elas diferenciam-se apenas nos seguintes aspectos:

- simetria - dentre as quatro variações, somente o AINV-NS foi proposto para matrizes não simétricas, operando exatamente como no Algoritmo 5. Já o AINV, SAINV e ISAINV são abordados apenas para matrizes simétricas, excluindo-se então todos os passos relativos aos cálculos de W (já que, nesse caso $Z = W$).

- cálculo do pivô - o AINV e AINV-NS foram propostos calculando-se o pivô da forma exibida no Algoritmo 5. Já o SAINV e ISAINV calculam o pivô como $d_{ii} = z_i^T A z_i$, garantindo ser livre de falha para matrizes SPD.

- estratégia de descarte - os descartes no AINV, AINV-NS e SAINV foram efetuados nas entradas dos vetores $z_j^{(i-1)}$ e $w_j^{(i-1)}$, a cada iteração. Já o ISAINV aplica o descarte duplo, sendo esse o seu diferencial. Mais precisamente, antes de calcular as entradas dos vetores $z_j^{(i-1)}$, primeiramente é avaliado o valor de $|\frac{r_j^{(i-1)}}{d_{ii}}|$ através de uma tolerância τ_2. Caso $|\ \frac{r_j^{(i-1)}}{d_{ii}}\ | \leq \tau_2$, o vetor $z_j^{(i)}$ não é atualizado. Caso contrário, a atualização é feita como no SAINV e, então, são aplicados os descartes nas entradas de $z_j^{(i)}$.

3.13.2 Classe FFAPINV

A classe FFAPINV é formada pelas variações SAINV-NS e FFAPINV e se diferencia da classe AINV na forma como os multiplicadores são calculados, interferindo principalmente no nível de memória compartilhada necessária a cada iteração.

Direcionada unicamente para matrizes não simétricas, a classe FFAPINV utiliza expressões alternativas para o cálculo dos multiplicadores de Z e W: a diferença é que os multiplicadores são calculados utilizando as equações $r_j^{(i-1)} = w_i^T A e_j$ e $s_j^{(i-1)} = z_i^T A^T e_j$, ou seja, para o cálculo dos multiplicadores de Z será necessário acessar os vetores colunas de W e, reciprocamente, para o cálculo dos multiplicadores de W será necessário acessar as colunas da matriz Z. Essa estrutura força um menor grau de paralelismo uma vez que não há como calcular Z e W separadamente como na classe AINV.

O SAINV-NS e o FFAPINV calculam os multiplicadores da forma descrita, diferenciam-se apenas em alguns aspectos. São eles:

- cálculo do pivô - no SAINV-NS este cálculo é feito como $d_{jj} = w_j^T A z_j$. Já no FFAPINV, o pivô é calculado da mesma forma que o SAINV-NS se a matriz não for positiva definida. Caso seja positiva definida, então $d_{jj} = Az_j$, se Az_j for diferente de zero, ou $d_{jj} = z_j^T A z_j$, Az_j for igual a zero. Isso garante que o algoritmo não falhe para matrizes positivas definidas.

- estratégia de descarte - o FFAPINV faz o descarte duplo (o mesmo feito pelo ISAINV), ou seja, quando atualiza os vetores coluna de Z e W, o FFAPINV primeiro avalia a magnitude de $\frac{r_j^{(i-1)}}{d_{ii}}$ e $\frac{s_j^{(i-1)}}{d_{ii}}$ para decidir se vai atualizar os vetores $z_j^{(i)}$ e $w_j^{(i)}$, respectivamente. Caso os tenha atualizado, utiliza o descarte por tolerância em suas entradas. Já o SAINV-NS fornece a opção de se efetuar descarte nas entradas dos vetores $z_i \frac{r_j^{(i-1)}}{d_{ii}}$ e $w_i \frac{s_j^{(i-1)}}{d_{ii}}$ durante a atualização de $z_j^{(i)}$ e $w_j^{(i)}$ e, posteriormente, são feitos descartes em $z_j^{(i)}$ e $w_j^{(i)}$.

3.13.3 Classe AINV-LU

A classe AINV-LU é formada pelas variações RIF, RIF-NS e SAINV-VAR. O ponto central da classe AINV-LU é o cômputo dos fatores LU de A utilizando-se do AINV. Esse cálculo é possível, pelas relações (2.3.10) e (2.3.11) exibidas e demonstradas na Seção 2.3. Essa escolha em armazenar as entradas de L e U pode ter como objetivo calcular uma fatoração ILU ou para obter uma forma alternativa no cálculo dos fatores Z e W do algoritmo da inversa aproximada.

As três variações não falham para matrizes positivas definidas por utilizarem a relação (2.7.18). Elas se diferenciam apenas em alguns aspectos, sendo os principais deles:

- simetria - o RIF é proposto para matrizes simétricas e o RIF-NS e SAINV-VAR para matrizes não simétricas.

- matrizes produzidas - o RIF e o RIF-NS fornecem os fatores da aproximação da matriz A, sendo eles as matrizes L e D, no primeiro caso, e L, D e U, no segundo. Já o SAINV-VAR calcula os fatores da inversa aproximada de A. Primeiramente, o algoritmo fornece as matrizes W, D e U e depois o Z é obtido através da aproximação de U^{-1} pelo somatório de Neumann. Vejamos que, basicamente, a única diferença entre o SAINV-VAR e o RIF-NS são os elementos guardados durante do algoritmo. No SAINV-VAR são guardados os vetores de W e os elementos de U e D, enquanto as entradas de L (que correspondem a $\frac{s_j^{(i-1)}}{d_{ii}}$) são descartadas. Já no RIF-NS, os elementos de L, U e D são armazenados até o final, descartando-se os vetores de W.

3.13.4 Classe Pivoteamento

A classe Pivoteamento é formada pelas variações AINVP, RIFP e LLAINVP, sendo a sua principal característica o uso de pivoteamento completo durante a execução do AINV. A implementação de um pivoteamento

completo no AINV possui algumas características próprias: considere que A seja uma matriz quadrada de ordem n e não singular e não simétrica. Em primeiro lugar, o uso de memória no cômputo de Z e W deve ser compartilhada, já que a busca pelo maior pivô se dará nos multiplicadores tanto de Z quanto de W, o que impede que ambos os fatores sejam calculados paralelamente. Outra característica importante sobre o uso de pivoteamento é que ele busca obter precondicionadores com maior qualidade e evitar a falha, sendo testado na literatura para problemas aonde o AINV sem pivoteamento obteve sérios problemas de desempenho. Essa última característica está relacionada e é análoga ao pivoteamento no contexto da fatoração LU e tenta justificar seu aumento de custo.

Os três métodos são aplicados para quaisquer matrizes quadradas não singulares. As suas principais diferenças são:

- pela direita ou pela esquerda - o AINVP e o RIFP são apresentados na versão pela direita do AINV, já o LLAINVP é mostrado na versão pela esquerda. Essa é a principal diferença entre o AINVP e o LLAINVP, já que a estrutura do algoritmo com pivoteamento na versão pela esquerda é bastante diferente devido algumas peculiaridades detalhadas na Seção 3.12.

- matrizes produzidas - o AINVP e o LLAINVP produzem os fatores Z, D e W da inversa aproximada de $\Pi A \Sigma$. Já o RIFP produz os fatores L, D e U de $\Pi A \Sigma$ aproximados, utilizando o AINVP como base, mas calculando L e U através das fórmulas (2.3.10) e (2.3.11). Além disso, também são efetuadas permutações em L e U para manter a consistência do algoritmo (mais detalhes na Seção 3.11).

Exercício 3.1. *Escreva o algoritmo do FFAPINV. Depois disso, sabendo que ele é baseado na versão pela esquerda do AINV, escreva o algoritmo do FFAPINV caso ele seja baseado na versão pela direita do AINV.*

Exercício 3.2. *Seja A uma matriz não singular e sejam as aproximações $A \approx LDU$ e $A^{-1} \approx ZD^{-1}W^T$, em que Z, W e U são triangulares superiores unitárias, L é triangular inferior unitária e D é diagonal. Considere a variação SAINV-VAR. Então:*

a) Escreva o algoritmo referente a essa variação;

b) Baseado no SAINV-VAR, escreva um algoritmo que calcule Z, L e D. Neste caso, sugira uma forma de se calcular W.

Exercício 3.3. *Seja A seja uma matriz não singular, com menores principais diferentes de zero, tal que*

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

e considere as variações AINVP e LLAINVP que calculam a fatoração $(\Pi A \Sigma)^{-1} \approx ZD^{-1}W^T$, sendo Π e Σ as matrizes permutações referenciadas nas Seções 3.5 e 3.12. Então

a) Utilize o AINVP ou LLAINVP para encontrar a fatoração $(\Pi A \Sigma)^{-1} = ZD^{-1}W^T$ (ou seja, sem efetuar descartes), com $\alpha = 1$.

b) A partir das matrizes obtidas no item anterior, encontre uma fatoração de A^{-1}.

Algoritmo 13: AINVP

Dados: matriz A $n \times n$ não singular

Resultado: D, Z e W tais que $(\Pi A \Sigma)^{-1} = ZD^{-1}W^{\mathsf{T}}$

$z_i^{(0)}, w_i^{(0)} \leftarrow e_i$, $i = 1, \ldots, n$;
$\Pi = \Sigma = I_n$;
Tolerância $\alpha \in (0, 1]$;
para $i \leftarrow 1$ **até** n **faça**
    satisfied_s =Falso, satisfied_r =Falso;
    enquanto *não satisfied_s* **faça**
        $s_j^{(i-1)} = (w_j^{(i-1)})^T (\Pi A \Sigma) z_i^{(i-1)}$ $j \geq i$;
        se $|s_i^{(i-1)}| < \alpha max_{m \geq i+1} |s_m^{(i-1)}|$ **então**
            satisfied_r =Falso;
            Escolher k tal que $|s_k^{(i-1)}| = max_{m \geq i+1} |s_m^{(i-1)}|$;
            Permutar as colunas i e k de $W - I$ e linhas i e k de Π e os elementos $s_i^{(i-1)}$ e $s_k^{(i-1)}$;
        satisfied_s =Verdadeiro;
        $r_j^{(i-1)} = (w_i^{(i-1)})^T (\Pi A \Sigma) z_j^{(i-1)}$ $j \geq i$;
        se $|r_i^{(i-1)}| < \alpha max_{m \geq i+1} |r_m^{(i-1)}|$ **então**
            satisfied_s =Falso;
            Escolher k tal que $|r_k^{(i-1)}| = max_{m \geq i+1} |r_m^{(i-1)}|$;
            Permutar as colunas i e k de $Z - I$ e Σ e os elementos $r_i^{(i-1)}$ e $r_k^{(i-1)}$;
        satisfied_r =Verdadeiro;
    $d_{ii} = r_i^{(i-1)}$ ou $s_i^{(i-1)}$;
    $z_i = z_i^{(i-1)}$; $w_i = w_i^{(i-1)}$;
    para $j \leftarrow i + 1$ **até** n **faça**
        $z_j^{(i)} \leftarrow z_j^{(i-1)} - z_i \frac{r_j^{(i-1)}}{d_{ii}}$;
        $w_j^{(i)} \leftarrow w_j^{(i-1)} - w_i \frac{s_j^{(i-1)}}{d_{ii}}$;
        Aplicar estratégia de descarte em $z_j^{(i)}$ e $w_j^{(i)}$.
$D \leftarrow diag(d_{11}, d_{22}, ..., d_{nn})$, $Z \leftarrow [z_1, z_2, ..., z_n]$ e $W \leftarrow [w_1, w_2, ..., w_n]$

Algoritmo 14: RIFP

Dados: matriz A $n \times n$ não singular

Resultado: L, D, e U tais que $\Pi A \Sigma \approx LDU$

$z_i^{(0)}, w_i^{(0)} \leftarrow e_i$, $i = 1, \ldots, n$;

$\Pi = \Sigma = L = U = I_n$;

Tolerância $\alpha \in (0, 1]$;

para $i \leftarrow 1$ **até** n **faça**

  satisfied_s =Falso, satisfied_r =False;

  enquanto *não satisfied_s* **faça**

    $s_j^{(i-1)} = (w_j^{(i-1)})^T (\Pi A \Sigma) z_i^{(i-1)}$ $j \geq i$;

    se $|s_i^{(i-1)}| < \alpha max_{m \geq i+1} |s_m^{(i-1)}|$ **então**

      satisfied_r =False;

      Escolher k tal que $|s_k^{(i-1)}| = \max_{m \geq i+1} |s_m^{(i-1)}|$;

      Permutar as colunas i and k de $W - I$, as linhas i e k de $L - I$ e de Π e os elementos $s_i^{(i-1)}$ e $s_k^{(i-1)}$;

    satisfied_s =Verdadeiro;

    $r_j^{(i-1)} = (w_i^{(i-1)})^T (\Pi A \Sigma) z_j^{(i-1)}$ $j \geq i$;

    se $|r_i^{(i-1)}| < \alpha \max_{m \geq i+1} |r_m^{(i-1)}|$ **então**

      satisfied_s =Falso;

      Escolher k tal que $|r_k^{(i-1)}| = max_{m \geq i+1} |r_m^{(i-1)}|$;

      Permutar as colunas i e k de $Z - I$, $U - I$ e de Σ e os elementos $r_i^{(i-1)}$ e $r_k^{(i-1)}$;

    satisfied_r =Verdadeiro;

  $d_{ii} = r_i^{(i-1)}$ ou $s_i^{(i-1)}$;

  $z_i = z_i^{(i-1)}$; $w_i = w_i^{(i-1)}$;

  para $j \leftarrow i + 1$ **até** n **faça**

    $u_{ij} = \frac{r_j^{(i-1)}}{p_i}$;

    $l_{ji} = \frac{s_j^{(i-1)}}{q_i}$;

    Aplicar estratégia de descarte em u_{ij} e u_{ji};

    $z_j^{(i)} \leftarrow z_j^{(i-1)} - z_i \frac{r_j^{(i-1)}}{p_i}$;

    $w_j^{(i)} \leftarrow w_j^{(i-1)} - w_i \frac{s_j^{(i-1)}}{q_i}$;

    Aplicar estratégia de descarte em $z_j^{(i)}$ e $w_j^{(i)}$.

$D \leftarrow diag(p_1, p_2, ..., p_n)$, $L \leftarrow [l_{ji}]$ e $U \leftarrow [u_{ij}]$

Algoritmo 15: LLAINVP

Dados: matriz A $n \times n$ não singular
Resultado: Z, W e D tais que $(\Pi A \Sigma)^{-1} \approx ZD^{-1}W^{\mathsf{T}}$

$\Pi = \Sigma = I_n$;Tolerância $\alpha \in (0, 1]$;
para $j \leftarrow 1$ **até** n **faça**
  $m_j = n_j =$ iter $= 0$; satisfied_s = Falso, satisfied_r = Falso;
  enquanto *não satisfied _ s* **faça**
    $z_i^{(0)} = e_i$; $iter = iter + 1$;
    para $i \leftarrow 1$ **até** $j - 1$ **faça**
      $r_j^{(i-1)} = e_i^T(\Pi A \Sigma) z_j^{(i-1)}$; $z_j^{(i)} \leftarrow z_j^{(i-1)} - z_j^{(j-1)} \frac{r_j^{(i-1)}}{r_j^{(j-1)}}$;
      Aplicar estratégia de descarte em $z_j^{(i-1)}$;
    se $iter = 1$ **então**
      $s_j^{(j-1)} = e_j(\Pi A \Sigma) z_j^{(j-1)}$;
      senão
        $s_j^{(j-1)} = r_j^{(j-1)}$;
    para $i \leftarrow j + 1$ **até** n **faça**
      $s_i^{(j-1)} = e_i^T(\Pi A \Sigma) z_j^{(j-1)}$
    se $|s_j^{(j-1)}| < \alpha \max_{m \geq j+1} |s_m^{(j-1)}|$ **então**
      $m_j = m_j + 1$; $\pi_{m_j}^{(j-1)} = I_n$;
      satisfied_r =Falso;
      Escolher k tal que $|s_k^{(j-1)}| = \max_{m \geq j+1} |s_m^{(j-1)}|$;
      Permutar as linhas j e k de $\pi_{m_j}^{(j-1)}$ e os elementos $s_j^{(j-1)}$ e $s_k^{(j-1)}$;
      $\Pi = \pi_{m_j}^{(j-1)} \Pi$;
    satisfied_s =Verdadeiro;
    se *não satisfied_ r* **então**
      $w_i^{(0)} = e_i$;
      para $i \leftarrow 1$ **até** $j - 1$ **faça**
        $s_j^{(i-1)} = e_i^T(\Pi A \Sigma) w_j^{(i-1)}$; $w_j^{(i)} \leftarrow w_j^{(i-1)} - w_j^{(j-1)} \frac{s_j^{(i-1)}}{s_j^{(j-1)}}$;
        Aplicar estratégia de descarte em $w_j^{(i-1)}$;
      $r_j^{(j-1)} = s_j^{(j-1)}$;
      para $i \leftarrow j + 1$ **até** n **faça**
        $r_i^{(j-1)} = w_j^{(j-1)}(\Pi A \Sigma) e_i$
      se $|r_j^{(j-1)}| < \alpha \max_{m \geq j+1} |r_m^{(j-1)}|$ **então**
        $n_j = n_j + 1$; $\sigma_{n_j}^{(j-1)} = I_n$; satisfied_$s$ = Falso;
      Escolher k tal que $|r_k^{(j-1)}| = \max_{m \geq j+1} |r_m^{(j-1)}|$;
      Permutar as linhas j e k de $\sigma_{n_j}^{(j-1)}$ e os elementos $r_j^{(j-1)}$ e $r_k^{(j-1)}$;
      $\Sigma = \Sigma \sigma_{n_j}^{(j-1)}$;
      satisfied_r =Verdadeiro;
  $d_{jj} = s_j^{(j-1)}$; $z_j = z_j^{(j-1)}$; $w_j = w_j^{(j-1)}$;
$D \leftarrow diag(d_{11}, d_{22}, ..., d_{nn})$, $Z \leftarrow [z_1, z_2, ..., z_n]$, e $W \leftarrow [w_1, w_2, ..., w_n]$

Capítulo 4

Complexidade da Inversa Aproximada

Neste capítulo, estudaremos a complexidade em termos de operações de ponto flutuante dos algoritmos de Inversa Aproximada. Do ponto de vista da aplicação em sistemas lineares esparsos e de grande porte, a complexidade tanto da construção quanto da aplicação dos precondicionadores precisa ser linear com respeito ao número de variáveis consideradas. Na primeira seção, verificaremos que, sem a existência de descarte, os custos dos algoritmos são inviáveis. Nas demais seções, apresentaremos diferentes estratégias para a redução do custo computacional.

4.1 Complexidade sem considerar os descartes

Começaremos analisando o algoritmo de A-conjugação aplicado a matrizes simétricas sem levar o descarte em consideração. Primeiramente, consideremos os seguintes pressupostos: A é uma matriz $n \times n$, simétrica, não singular, aonde n é grande. Quanto a complexidade, o AINV possui dois momentos-chave: as atualizações da matriz Z e o cálculo do pivô.

A versão pela direita (a versão pela esquerda possui a mesma complexidade) calcula, na i-ésima iteração, as colunas de Z do seguinte modo: $z_j^{(i)} \leftarrow z_j^{(i)} - z_i \frac{r_j^{(i-1)}}{d_{ii}}$ aonde $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant n$, com $i < j$. Vamos analisar trecho a trecho, fixado j. Primeiramente, analisemos o cálculo do multiplicador $r_j^{(i-1)} \leftarrow a_{i-1}^{\intercal} z_j^{(i-1)}$. Para fazer o produto interno $a_{i-1}^{\intercal} z_j^{(i-1)}$ devemos observar que $z_j^{(i-1)}$ é tal que $nnz(z_j^{(i-1)}) = i$, lembrando que sua j-ésima entrada é 1. Assim, são necessários i produtos e $i-1$ somas, totalizando $2i-1$ operações. Temos que $\frac{r_j^{(i-1)}}{d_{ii}}$ é

uma divisão, sendo igual, então, a 1 operação. Já para fazer $z_i \frac{r_j^{(i-1)}}{d_{ii}}$ devemos notar que o vetor resultante é calculado através de um produto de um vetor por um escalar. O vetor z_i tem preenchimento igual a i, o que implica que são necessários i produtos. Finalmente, para realizar $z_j^{(i)} - z_i \frac{r_j^{(i-1)}}{d_{ii}}$ são necessárias i somas. Ao todo, o custo de produção de $z_j^{(i)}$, representado por $C(z_j^{(i)})$, vale $C(z_j^{(i)}) = 4i$.

Na i-ésima iteração, $n - i$ colunas são atualizadas, o que faria que o custo por iteração seja igual a $(4i)(n - i) = -4i^2 + 4ni$. Assim, considerando as n iterações, o custo do algoritmo para produzir Z é:

$$\sum_{i=1}^{n} -4i^2 + 4ni = -4\sum_{i=1}^{n} i^2 + (4n)\sum_{i=1}^{n} i. \tag{4.1.1}$$

Resolvendo (4.1.1) chegamos a:

$$\frac{2n^3}{3} - \frac{2n}{3}.$$

Desta forma, o algoritmo, sem o cálculo do pivô, tem ordem n^3.

Quanto ao pivô $d_{ii} = a_i^{\mathsf{T}} z_i$, na sua construção são executados i produtos (preenchimento de Z) e $i - 1$ somas. Assim, para produzir d_{ii} teremos um custo de $2i-1$ operações. Assim, o custo total da produção da matriz D será:

$$C(D) = \sum_{i=1}^{n} 2i - 1 = n(n + 1) - n = n^2,$$

que também é de ordem n^2 do sistema. Concluímos que o custo total do algoritmo de A-conjugação é:

$$\frac{2n^3}{3} + n^2 - \frac{2n}{3}.$$

Quanto a versão livre de falha para matrizes simétricas positivas definidas, o pivô é produzido como: $d_{ii} = z_i^{\mathsf{T}} A z_i$. Analisando o produto $z_i^{\mathsf{T}} A z_i$ vemos que, a parte de A que efetivamente estará na conta é a sua submatriz líder principal de ordem i. Assim, cada entrada Az_i é equivalente a um produto interno de dois vetores de ordem i. Desse modo, cada entrada de Az_i custa i produtos e $i - 1$ somas. Dessas i entradas não nulas de Az_i. Assim, podemos dizer que o custo de Az_i é $i(2i-1)$. Por fim, para produzir $z_i^{\mathsf{T}}(Az_i)$ serão efetuados mais i produtos e $i - 1$ somas, gerando um custo total do pivô de:

$$2i^2 - i + 2i - 1 = 2i^2 + i - 1,$$

e o custo total de produção da matriz D fica em:

$$C(D) = \sum_{i=1}^{n} 2i^2 + i - 1 = \frac{2n^3}{3} + \frac{3n^2}{2} - \frac{n}{6}.$$

Assim, o custo total é:

$$\frac{4n^3}{3} + \frac{3n^2}{2} - \frac{5n}{6}.$$

4.2 Complexidade considerando os descartes

4.2.1 Custo da classe AINV

No que segue, estudaremos a complexidade do AINV, levando-se em consideração os seguintes pressupostos: A é uma matriz $n \times n$, não singular, simétrica, aonde n é grande. A partir daqui, consideraremos o uso de descartes em Z. A matriz Z possuirá, então, um preenchimento máximo de linha ou coluna, isto é, existe um valor N tal que $N \geq nnz(z_{:,i})$ e $N \geq nnz(z_{i,:})$ para todo $1 \leqslant i \leqslant n$, em que $nnz(z_{:,i})$ e $nnz(z_{i,:})$ representam o número de não zeros da i-ésima coluna e i-ésima linha de Z, respectivamente. Quanto a complexidade, o AINV possui dois momentos-chave: as atualizações da matriz Z e o cálculo do pivô.

A versão pela direita (a versão pela esquerda possui a mesma complexidade) calcula, na i-ésima iteração, as colunas de Z do seguinte modo: $z_j^{(i)} \leftarrow z_j^{(i-1)} - z_i \frac{r_j^{(i-1)}}{d_{ii}}$ aonde $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant n$, com $i < j$. Vamos analisar trecho a trecho, fixado j. Primeiramente, analisemos o cálculo do multiplicador $r_j^{(i-1)} \leftarrow a_{i-1}^{\intercal} z_j^{(i-1)}$. Para fazer o produto interno $a_{i-1}^{\intercal} z_j^{(i-1)}$ devemos observar que $z_j^{(i-1)}$ é tal que $nnz(z_j^{(i-1)}) = min\{N, i-1\} \leqslant N$. Por isso, são necessários, no máximo, N produtos e $N-1$ somas. Temos que $\frac{r_j^{(i-1)}}{d_{ii}}$ é uma divisão, sendo igual, então, a 1 operação. Já para fazer $z_i \frac{r_j^{(i-1)}}{d_{ii}}$ devemos notar que o vetor resultante é calculado por meio de um produto de um vetor por um escalar. O vetor z_i tem preenchimento igual a N, o que implica que são necessários N produtos. Finalmente, para realizar $z_j^{(i)} - z_i \frac{r_j^{(i-1)}}{d_{ii}}$ são necessárias, no máximo, $2N$ somas (supondo que os preenchimentos de $z_j^{(i-1)}$ e z_i estejam em posições disjuntas). Assim, a posteriori, será necessário definir um critério para o qual sejam eliminados os excedentes do limite de preenchimento das colunas de Z. Ao todo, o custo de produção de $z_j^{(i)}$, representado por $C(z_j^{(i)})$, vale $C(z_j^{(i)}) = 5N$. Na

i-ésima iteração, são $n - i$ colunas atualizadas, o que faria que o custo por iteração seja igual a $C(i) = 5N(n - i)$, no entanto, isso preenche a i-ésima linha de Z mais que se limite N de preenchimento. Mas isso é um problema que, marcadamente, deve ser resolvido a priori, pois, do contrário, o custo do algoritmo para produzir Z seria:

$$\sum_{i=1}^{n} 5N(n-i) = \frac{5}{2}N(n^2 - n),$$

E o algoritmo seria, sem o cálculo do pivô, de ordem n^2. Por isso, é necessário escolher com antecedência, quais são as M colunas $z_j^{(i)}$ que serão atualizadas. Assim, o custo por iteração se reduz a $5NM$ e o custo total para calcular as entradas de Z no Algoritmo será:

$$C(Z) = \sum_{i=1}^{n} 5NM = 5NMn.$$

Quanto ao pivô $d_{ii} = a_i^{\mathsf{T}} z_i$, sua construção indica que são executados N produtos (preenchimento de Z) e $N-1$ somas. Assim, para produzir d_{ii}, teremos um custo de $2N - 1$ operações. Assim, o custo total da produção da matriz D será:

$$C(D) = \sum_{i=1}^{n} 2N - 1 = (2N - 1)n,$$

que também é de ordem n do sistema. Concluímos, então, que o custo total do AINV, dado por $C(AINV)$ é:

$$C(AINV) = 5NMn + (2N - 1)n.$$

Quanto ao SAINV, que é a versão livre de falha para matrizes SPD, o pivô é produzido como: $d_{ii} = z_i^{\mathsf{T}} A z_i$. Assim, ainda teremos uma divisão, embora os produtos sejam realizados de modo alternativo. Analisando o produto $z_i^{\mathsf{T}} A z_i$ vemos que, a parte de A que efetivamente estará presente no cálculo é a sua submatriz líder principal de ordem i. Assim, cada entrada Az_i é equivalente a um produto interno de dois vetores de ordem i, sendo que z_i tem preenchimento igual a Z. Desse modo, cada entrada de Az_i custa N produtos e $N - 1$ somas. Dessas, possíveis i entradas não nulas de Az_i, apenas N delas serão realmente utilizadas no produto interno $z_i^{\mathsf{T}}(Az_i)$. Assim, podemos dizer que o custo de Az_i é $N(2N - 1)$. Por fim, para produzir $z_i^{\mathsf{T}}(Az_i)$ serão efetuados mais N produtos e $N - 1$ somas, gerando um custo total do pivô de:

$$2N^2 - N + 2N - 1 = 2N^2 + N - 1,$$

e o custo total de produção da matriz D fica em:

$$C(D) = \sum_{i=1}^{n} 2N^2 + N - 1 = (2N^2 + N - 1)n.$$

Assim, o custo total do SAINV é:

$$C(SAINV) = 5NMn + 2N^2n + Nn - n = C(AINV) + 2N^2n - Nn.$$

Quanto ao ISAINV, é útil notar que não há nenhuma mudança no que diz respeito as operações algébricas quanto ao SAINV. Novamente, portanto, ao todo, o custo de produção de $z_j^{(i)}$ vale $5N$. No entanto, por iteração serão feitas, no máximo, mais M comparações, já que o vetor $z_j^{(i)}$ só é atualizado depois de comparar a magnitude de $\frac{r_j^{(i-1)}}{d_{ii}}$ com um valor de tolerância previamente escolhido. Logo, o custo total de $C(z_j^{(i)}) = 5NM + M$. Dessa forma, por analogia ao que foi feito anteriormente, temos $C(Z) = \sum_{i=1}^{n} 5NM + M = (5N + 1)Mn$. A complexidade final, incluída o cálculo do pivô será $C(ISAINV) = (5N + 1)Mn + (2N^2 + N - 1)n$.

A versão não simétrica AINV-NS necessita do cálculo de Z e W e seu custo será, precisamente, o dobro do cálculo de Z. Logo, $C(Z) + C(W) = 2 \cdot \sum_{i=1}^{n} 5NM = 2 \cdot 5NMn = 10NMn$. O cálculo do pivô é o mesmo que o cálculo do AINV, então teremos $C(AINV - NS) = 10NMn + (2N - 1)n$.

4.2.2 Custo da classe FFAPINV

No caso do FFAPINV, é oportuno observar que o cálculo de atualização das matrizes Z e W possuem a mesma quantidade de operações que a do AINV-NS: o que muda é qual vetor é utilizado na atualização dos multiplicadores. Além disso, ele faz a mesma avaliação dos multiplicadores que o ISAINV, porém são avaliados tanto os multiplicadores de Z quanto os de W. Temos também que ele faz a mesma quantidade de operações em relação ao cálculo dos pivôs que o do ISAINV (que é o mesmo do SAINV). Logo, $C(FFAPINV) = (5N + 1)2Mn + (2N^2 + N - 1)n$.

Em relação ao SAINV-NS, o cálculo de atualização das matrizes Z e W é feito da mesma forma que o AINV-NS. Porém, o SAINV-NS calcula o pivô como $d_{ii} = w_i^T A z_i$, que faz as mesmas quantidades de operações que o cálculo do pivô do SAINV. Logo, $C(SAINV - NS) = 10NMn + (2N^2 + N - 1)n$.

4.2.3 Custo da classe AINV-LU

Analisando o RIF, vemos que a sua principal diferença em relação ao SAINV é o que será armazenado ao final, obtendo, assim, o mesmo

custo. Portanto, $C(RIF) = C(SAINV)$.

Em relação ao SAINV-VAR, na i-ésima iteração, com $1 \leq i \leq n$, o $w_j^{(i)}$ é calculado com o mesmo número de operações algébricas que o $z_j^{(i)}$ no SAINV. O pivô também é calculado da mesma forma que o SAINV. Porém, além desses cálculos, fixado o j, também deve-se achar as entradas de U com por meio de $u_{ij} = \frac{r_j^{(i-1)}}{d_{ii}}$ e do multiplicador $r_j^{(i-1)} = a_{ij} - \sum_{k=1}^{i-1} s_i^{(k-1)} u_{kj}$. Em relação a $u_{ij} = \frac{r_j^{(i-1)}}{d_{ii}}$, temos que é executada uma operação de divisão. Já em relação a $r_j^{(i-1)}$, no somatório são executados produtos entre os elementos da coluna j de U e os $s_i^{(k-1)}$. Mas, o número de elementos não nulos nas linhas e colunas de U também deve ser limitado por N. Logo são efetuadas, no máximo, N multiplicações. Além disso, também é efetuada uma soma. Portanto, o custo, do algoritmo, por iteração, é de $(5N + 1 + N + 1)M = (6N + 2)M$. Incluindo o custo do pivô, o algoritmo terá um custo total de $C(SAINV - VAR) = (6N + 2)Mn + (2N^2 + N - 1)n$.

Quanto ao RIF-NS, sua principal diferença em relação ao SAINV-VAR é o que será armazenado ao final. Portanto, $C(RIF) = C(SAINV - VAR)$.

4.2.4 Custo da classe Pivoteamento

Dentre todos os algoritmos estudados aqui, os algoritmos da classe Pivoteamento são os mais complicados e caros. Isso se dá pelo pivoteamento. Vejamos, primeiramente, o custo do AINVP. Vamos analisar a i-ésima iteração com mais detalhes: primeiramente o algoritmo executa o processo de pivoteamento por linha. Primeiro ele calcula o pivô e os multiplicadores $s_j^{(i-1)}$ como

$$s_j^{(i-1)} = (w_j^{(i-1)})^T (\Pi A \Sigma) z_i^{(i-1)}$$

para $j \geq i$. Para cada produto interno desses, temos N produtos e $N - 1$ somas. Mas, para preservar a viabilidade do algoritmo, devemos atualizar, no máximo, M multiplicadores. Logo, o custo dessa operação vale $(2N - 1)M$. Depois disso, devemos analisar a condição de pivoteamento através da relação

$$|s_i^{(i-1)}| < \alpha max_{m \geq i+1} |s_m^{(i-1)}|.$$

Como temos um máximo de M multiplicadores, então fazemos, no máximo, M produtos (do multiplicador por α) e M comparações. Então, o processo de pivoteamento por linhas executa $(2N-1)M+2M$ operações. Esse mesmo raciocínio é executado no pivoteamento por colunas, onde se compara os multiplicadores $r_j^{(i-1)}$, fazendo o mesmo número

de operações. Porém, esses cálculos podem acontecer diversas vezes, até que a condição de pivoteamento seja totalmente atendida. No pior das hipóteses, o algoritmo fará, no máximo $2(n-i)+1$ pivoteamentos a cada iteração. Logo, o custo da etapa de pivoteamento será, no máximo,

$$((2N-1)M+2M)(2(n-i)+1) = (2NM+M)(2(n-i)+1).$$

Depois desta etapa, $w_j^{(i)}$ e $z_j^{(i)}$ são atualizados da mesma forma que no AINV tendo custo de $10NM$. Considerando as n iterações, o custo total do algoritmo é de

$$\sum_{i=1}^{n}[(2NM+M)(2(n-i)+1)+10NM] = 2NMn^2+Mn^2+10MNn.$$

Vejamos que o custo total é de ordem n^2. Isso ocorre por conta da quantidade de pivoteamentos que podem ser necessários durante o algoritmo. Ou seja, para contornar este fato deve-se limitar a quantidade de pivoteamentos a serem executados. Isto pode ser feito através do parâmetro α escolhido previamente. Os autores que propuseram o AINVP salientaram que a quantidade de pivoteamentos executados durante o algoritmo tende a ser baixo, permitindo a que ele seja realizável. Na nossa análise, limitaremos o número de pivoteamentos em, no máximo, P vezes (como os multiplicadores devem ser calculados no mínimo uma vez então $2 \leqslant P$). A partir disto, o custo total do AINVP é de $C(AINVP) = (2NM+M)Pn + 10NMn$.

Quanto ao RIFP, sua principal diferença em relação ao AINVP é o que será armazenado ao final. Portanto, $C(RIFP) = C(AINVP)$.

Quanto ao LLAINVP, sua principal diferença em relação ao AINVP é o fato dele ser pela esquerda, o que, neste caso, muda a ordem das operações. Vamos estudar como isso acontece. A j-ésima iteração, inicia-se atualizando os vetores $z_j^{(i)}$, com $1 \leqslant i \leqslant j-1$. Fixado i, o custo será igual a $5N$, da mesma forma que no AINV. Porém, como atualizamos no máximo M vetores, então isso terá um custo máximo de $5NM$. Depois, são calculados o pivô e os multiplicadores $s_j^{(i-1)}$ como

$$s_j^{(i-1)} = (w_j^{(i-1)})^T(\Pi A \Sigma)z_i^{(i-1)}$$

para $j \geq i$. Para cada produto interno desses, temos N produtos e $N-1$ somas. Mas, para preservar a viabilidade do algoritmo, devemos atualizar, no máximo, M multiplicadores. Logo, o custo dessa operação vale $(2N-1)M$. Depois disso, devemos analisar a condição de pivoteamento através da relação

$$|s_i^{(i-1)}| < \alpha max_{m \geq i+1}|s_m^{(i-1)}|.$$

Como temos um máximo de M multiplicadores, então fazemos, no máximo, M produtos (do multiplicador por α) e M comparações. Então, o processo de pivoteamento por linhas executa $(2N-1)M+2M$ operações. Depois disso, fazemos o mesmo raciocínio para construir a matriz W e para o cálculo e pivoteamento em relação aos $r_j^{(i-1)}$. Mas, diferentemente do AINVP, sempre que se faz um pivoteamento, por exemplo, por linha, deve-se recalcular os vetores de W e também os multiplicadores $r_j^{(i-1)}$ para testar novamente a condição de pivoteamento. O mesmo raciocínio pode ser feito para o pivoteamento por coluna. Logo, como a matriz faz, no máximo, $(2(n-j)+1)$ pivoteamentos, teremos um total de, no máximo

$$(5NM+(2N-1)M+2M)(2(n-j)+1) = (7NM+M)(2(n-j)+1)$$

operações por iteração j. Considerando as n iterações, o algoritmo terá um custo total de

$$\sum_{i=1}^{n}(7NM+M)(2(n-j)+1) = 7NMn^2+Mn^2.$$

Vejamos que o custo total também é de ordem n^2, como no AINVP. Isso ocorre por conta da quantidade de pivoteamentos que podem ser necessários durante o algoritmo. Ou seja, para contornar este fato deve-se limitar a quantidade de pivoteamentos a serem executados. Na nossa análise, limitaremos o número de pivoteamentos em, no máximo, P vezes como os multiplicadores devem ser calculados no mínimo uma vez então $2 \leqslant P$). A partir disto, o custo total do LLAINVP é de $C(AINVP) = (7NM+M)Pn$.

4.3 Resumo dos custos dos algoritmos

Diante do estudo aqui apresentado, sintetizamos na Tabela 4.1 os custos de cada um dos algoritmos.

Algoritmo	Custo
A-conjugação	$\frac{2n^3}{3} - \frac{2n}{3}$
A-conjugação sem falha c/ SPD	$\frac{4n^3}{3} + \frac{3n^2}{2} - \frac{5n}{6}$
AINV	$5NMn + (2N-1)n$
SAINV	$5NMn + (2N^2+N-1)n$
ISAINV	$(5N+1)Mn + (2N^2+N-1)n$
RIF	$(5N+1)Mn + (2N^2+N-1)n$
AINV-NS	$10NMn + (2N-1)n$
SAINV-NS	$10NM + (2N^2+N-1)n$
FFPAPINV	$(5N+1)2Mn + (2N^2+N-1)n$
SAINV-VAR	$(6N+2)Mn + (2N^2+N-1)n$
RIF-NS	$(6N+2)Mn + (2N^2+N-1)n.$
AINVP	$(2NM+M)Pn + 10NMn$
RIFP	$(2NM+M)Pn + 10NMn$
LLAINVP	$(7NM+M)Pn$

Tabela 4.1: Custo dos algoritmos

Vejamos que, para que o AINV e suas variações não tenham ordem polinomial e, assim, sejam algoritmos viáveis, deve-se limitar o número de entradas de Z e W e a quantidade de linhas e colunas a serem atualizadas a cada iteração. Isso pode ser feito através das estratégias de descartes a serem empregadas. No caso específico dos algoritmos que fazem uso de pivoteamento completo (AINVP, RIFP e LLAINVP), além dessas condições, o número de vezes em que se executa o pivoteamento também deve se limitado. Isto pode ser feito com o auxílio do parâmetro α e também adotando-se estratégias de descarte adequadas. Tais ferramentas são feitas considerando que A seja esparsa, condição essencial para a viabilidade do algoritmo.

Exercício 4.1. *Determine o custo de uma iteração do Algoritmo de Biconjugação (sem descarte) versão pela esquerda em função da i-ésima iteração.*

Exercício 4.2. *Notoriamente, a versão pela esquerda do Algoritmo de Biconjugação inicia realizando menos operações por iteração que a versão pela direita. No entanto, termina realizando mais operações por iteração. Determine, se existir, em qual iteração ambas as versões do Algoritmo de Biconjugação realizam o mesmo número de operações.*

Exercício 4.3. *Mostre que a complexidade do Algoritmo de Inversa Aproximada (AINV), versão pela esquerda é igual a complexidade da versão pela direita.*

Exercício 4.4. *Determine, separadamente, o número de somas, multiplicações e divisões operadas por iteração, em cada variação do Algoritmo de Inversa Aproximada a seguir:*

a) AINV;

b) SAINV;

c) AINV-NS;

d) SAINV-NS;

e) FFAPINV;

f) SAINV-VAR;

g) LLAINVP.

Capítulo 5

Inversa Aproximada em Blocos para Matrizes Simétricas

Neste capítulo, apresentaremos os resultados do trabalho que desenvolvemos em [2], a respeito do suporte teórico para a Inversa Aproximada em Blocos de Matrizes Simétricas (*Block Approximate Inverse* ou simplesmente BAINV), precondicionador proposto por Benzi, Meyer e Tůma, em 2001 [11]. Nós provaremos sua consistência e que para as classes de matrizes do tipo M e H o algoritmo não falha, independente da estratégia de descarte adotada.

5.1 Notação

5.1.1 Estrutura em blocos homogêneos

Primeiramente, descreveremos a estrutura de matrizes em blocos $N \times N$ homogêneos com blocos de tamanho t. Seja A uma matriz $n \times n$, com $n = Nt$. Nós consideraremos a estrutura em blocos homogêneos $N \times N$ de A:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1N} \\ A_{21} & A_{22} & \cdots & A_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N1} & A_{N2} & \cdots & A_{NN} \end{bmatrix},$$

onde, para $1 \leq I, J \leq N$, o bloco A_{IJ} é uma submatriz $t \times t$ de A tal que

$$A_{IJ} = \begin{bmatrix} a_{(I-1)t+1,(J-1)t+1} & a_{(I-1)t+1,(J-1)t+2} & \cdots & a_{(I-1)t+1,Jt} \\ a_{(I-1)t+2,(J-1)t+1} & a_{(I-1)t+2,(J-1)t+2} & \cdots & a_{(I-1)t+2,Jt} \\ \vdots & \vdots & \ddots & \vdots \\ a_{It,(J-1)t+1} & a_{It,(J-1)t+2} & \cdots & a_{It,Jt} \end{bmatrix}.$$

Na sequência, se não houver alguma indicação explícita, nós assumiremos que os blocos têm tamanho t fixado. Neste capítulo, o termo "matriz bloco" ou "matriz-bloco $N \times N$"denotará a matriz com estrutura em blocos homogêneos e tamanho de bloco t, ao menos que se assuma algo diferente explicitamente.

Nós definimos como N-vetor-bloco, ou vetor-bloco a matriz-bloco $N \times 1$ (i.e., uma matriz $Nt \times t$) e uma N-vetor-bloco, ou linha-bloco de tamanho N, como uma matriz em blocos $1 \times N$ (i.e., uma matriz $t \times Nt$). Denotamos por A_J o vetor-bloco correspondente à J-ésima coluna-bloco de A e por A_{J*} a linha-bloco correspondente à J-ésima linha-bloco de A, ou seja,

$$A_J = \begin{bmatrix} A_{1J} \\ A_{2J} \\ \vdots \\ A_{NJ} \end{bmatrix}, \qquad A_{J*} = \begin{bmatrix} A_{J1} & A_{J2} & \cdots & A_{JN} \end{bmatrix}.$$

Nós também definimos os N-vetores-bloco canônicos E_I, $I = 1, 2, \ldots, N$, como o produto de Kronecker dos vetores canônicos $e_I \in \mathbb{R}^N$ e a matriz identidade $t \times t$.

Finalmente, considerando a matriz-bloco A, denotamos como $A_{I_0:I_f, J_0:J_f}$ a submatriz-bloco

$$A_{I_0:I_f, J_0:J_f} = \begin{bmatrix} A_{I_0,J_0} & A_{I_0,J_0+1} & \cdots & A_{I_0,J_f} \\ A_{I_0+1,J_0} & A_{I_0+1,J_0+1} & \cdots & A_{I_0+1,J_f} \\ \vdots & \vdots & \ddots & \vdots \\ A_{I_f,J_0} & A_{I_f,J_0+1} & \cdots & A_{I_f,J_f} \end{bmatrix}.$$

Por conveniência, denotamos "$1 : N$" simplesmente por ":". Por exemplo, o vetor-bloco A_J e a linha-bloco A_{J*}, definidos em (5.1.1), também podem ser expressos como $A_{:,J:J}$ e $A_{J:J,:}$, respectivamente.

Neste capítulo, o termo "matriz bloco" ou "matriz-bloco $N \times N$"denotará a matriz com estrutura em blocos homogêneos e tamanho de bloco t, ao menos que se assuma algo diferente explicitamente.

Uma matriz bloco D é dita "bloco-diagonal"se $D_{IJ} = 0$, para todo $I \neq J$. Uma matriz bloco A é dita "bloco-triangular superior"se $A_{IJ} = 0$, para todo $I > J$. Analogamente, uma matriz bloco A é dita "bloco-triangular inferior"se $A_{IJ} = 0$, para todo $I < J$.

5.1.2 Estrutura em blocos heterogêneos (tamanho de blocos variável)

Em algumas aplicações (por exemplo, o método AIM - *Adaptive Implicit Method* na sigla em inglês - de simulação de reservatório de petróleo,

ver [33, 75], em que algumas células de grades são tratadas implicitamente e outras explicitamente sobre em relação ao tempo de discretização do termo de fluxo), a matriz do problema possui uma estrutura de blocos heterogêneos, onde o número de linhas/colunas de cada bloco varia de acordo com a sua posição. No exemplo dado, isto está relacionado com o número variável de incógnitas por célula de grade.

Neste caso, nós suprimimos a condição de que n deve ser divisível por t e, ao invés disso, temos que $n = \sum_{K=1}^{N} n_K$. A estrutura em blocos será representada por (5.1.1), porém, neste caso, cada A_{IJ} é (possivelmente) um bloco retangular $n_I \times n_J$. Esta estrutura é apresentada mais detalhadamente em [11].

5.1.3 Outras notações

Usaremos as desigualdades $A \geq B$ (ou $A \leq B$) fazendo referência às entradas, ou seja, significando que $a_{ij} \geq b_{ij}$ (ou $a_{ij} \leq b_{ij}$), para todo i e j. O valor absoluto de A, $|A|$, também é definido fazendo referência às entradas de A, ou seja, $|A|_{ij} = |a_{ij}|$.

5.2 A-conjugação em blocos

Seja A uma matriz simétrica $n \times n$, com $n = Nt$, N e t inteiros.

Definição 5.1 (A-conjugado)**.** *Dois vetores-bloco U e V são A-conjugados se $U^{\mathsf{T}}AV$ é o bloco zero (i.e., uma matriz $t \times t$ formada só de zeros). Um conjunto de vetores-bloco $\{V_1, \ldots, V_K\}$ é dito A-conjugado se V_I e V_J são A-conjugados para todo I, J, em que $0 \leq I \leq K$ e $0 \leq J \leq K$.*

Notemos que, se Z é uma matriz-bloco $N \times N$, então $\{Z_1, \ldots, Z_N\}$ é A-conjugado se e somente se $Z^{\mathsf{T}}AZ = D$, em que D é bloco-diagonal. Note que, neste caso, se A e Z são não singulares, então

$$A^{-1} = ZD^{-1}Z^{\mathsf{T}}. \tag{5.2.1}$$

A decomposição (5.2.1) pode ser obtida através do algoritmo da versão pela direita da A-conjugação em blocos, Algoritmo 16 (que é essencialmente o mesmo algoritmo em [11]). Nós também apresentamos a versão pela esquerda, Algoritmo 17. É fácil verificar que ambos Algoritmos 16 e 17 produzem os mesmos resultados finais e parciais, podendo ser provado de forma análoga à versão escalar, no Capítulo 2.

Desejamos agora demonstrar o Teorema 5.1, o qual afirma que, sob determinadas condições, o Algoritmo 16 de fato produz a decomposição (5.2.1). Este teorema também estabelece as bases para nossos principais resultados, Teoremas 5.2 e 5.3. Vejamos, primeiramente, dois resultados auxialares, Lemas 5.1 e 5.2.

Algoritmo 16: A-conjugação em blocos pela direita

Dados: matriz A em blocos $N \times N$ não singular.
Resultado: D e Z não singulares tais que $A^{-1} = ZD^{-1}Z^T$.

$Z_I^{(0)} \leftarrow E_I, \qquad I \in \{1, ..., N\}$;
para $I \leftarrow 1$ **até** N **faça**
  $Z_I \leftarrow Z_I^{(I-1)}$;
  $D_{II} \leftarrow A_I^{\mathsf{T}} Z_I$;
  para $J \leftarrow I + 1$ **até** N **faça**
    $R_J^{(I-1)} \leftarrow A_I^{\mathsf{T}} Z_J^{(I-1)}$;
    $Z_J^{(I)} \leftarrow Z_J^{(I-1)} - Z_I D_{II}^{-1} R_J^{(I-1)}$;
$D \leftarrow diag(D_{11}, ..., D_{NN})$ e $Z \leftarrow \left[\; Z_1, \quad \cdots \quad , Z_N \; \right]$;

Algoritmo 17: A-conjugação em blocos pela esquerda

Dados: matriz A em blocos $N \times N$ não singular.
Resultado: D e Z não singulares tais que $A^{-1} = ZD^{-1}Z^T$.

$Z_1 \leftarrow E_1$;
$D_{11} \leftarrow A_{11}$;
para $J \leftarrow 2$ **até** N **faça**
  $Z_J^{(0)} \leftarrow E_J$;
  para $I \leftarrow 1$ **até** $J - 1$ **faça**
    $R_J^{(I-1)} \leftarrow A_I^{\mathsf{T}} Z_J^{(I-1)}$;
    $Z_J^{(I)} \leftarrow Z_J^{(I-1)} - Z_I D_{II}^{-1} R_J^{(I-1)}$;
  $Z_J \leftarrow Z_J^{(J-1)}$;
  $D_{JJ} \leftarrow A_J^{\mathsf{T}} Z_J$;
$D \leftarrow diag(D_{11}, ..., D_{NN})$ e $Z \leftarrow \left[\; Z_1, \quad \cdots \quad , Z_N \; \right]$;

Lema 5.1. *Caso o Algoritmo 16 não falhe (i.e., todos as matrizes D_{II}'s produzidas são não-singulares), ele produz uma matriz bloco-triangular superior Z. Além disso, a diagonal em blocos (ou bloco-diagonal) de Z é formada por matrizes identidade.*

Demonstração. Façamos indução em I para provar que, para $I = 0, 1, \ldots, N-1$,

$$E_L^T Z_J^{(I)} = \delta_{LJ}, \quad \forall\, J, L \ \text{ tal que } \ I < J \leq L \leq N, \tag{5.2.2}$$

onde δ_{LJ} é dado por

$$\delta_{LJ} = \begin{cases} \text{bloco identidade}, & \text{if } L = J \\ 0, & \text{if } L \neq J \end{cases}.$$

A igualdade (5.2.2) é claramente verdadeira para $I = 0$, já que $Z_J^{(0)} = E_J$. Agora, assumamos que (5.2.2) é verdadeira para algum $0 \leq I < N - 1$. A partir dessa hipótese, somado a linha 7 do Algoritmo 16,

$$E_L^T Z_J^{(I+1)} = E_L^T Z_J^{(I)} - E_L^T Z_{I+1}^{(I)} D_{(I+1)(I+1)}^{-1} R_J^{(I)} = E_L^T Z_J^{(I)} = \delta_{LJ},$$

completando a indução. □

Lema 5.2. *Caso o Algoritmo 16 não falhe, temos que, para algum $1 \leq J \leq N$ fixado,*

$$Z_J = Z_J^{(K)} - \sum_{I=K+1}^{J-1} Z_I D_{II}^{-1} R_J^{(I-1)},$$

para todo $0 \leq K \leq I - 1$.

Demonstração. Segue diretamente das linhas 3 e 7 do Algoritmo 16. □

Teorema 5.1. *Seja A uma matriz-bloco $N \times N$ simétrica tal que $A_{1:J,1:J}$ é não-singular para $J = 1, 2, \ldots, N$. Então o Algoritmo 16 não falha e produz uma matriz D bloco-diagonal não singular e uma matriz-bloco Z tal que $A^{-1} = ZD^{-1}Z^{\mathsf{T}}$ (ou, de forma equivalente, $Z^{\mathsf{T}}AZ = D$).*

Demonstração. Provaremos por indução. É suficiente demonstrar que H_J é verdadeira para cada $J \in \{1, \ldots, N\}$.

$$H_J : \begin{cases} A_I^{\mathsf{T}} Z_J = 0, & I \in \{1, 2, \ldots, J-1\}; \quad (5.2.3a) \\ Z_I^{\mathsf{T}} A Z_J = Z_J^{\mathsf{T}} A Z_I = 0, & I \in \{1, 2, \ldots, J-1\}; \quad (5.2.3b) \\ Z_J^{\mathsf{T}} A Z_J = D_{JJ}; & (5.2.3c) \\ D_{JJ} \text{ é não singular}; & (5.2.3d) \end{cases}$$

As hipóteses (5.2.3b), (5.2.3c), e (5.2.3d) demonstram o resultado pretendido, enquanto a hipótese (5.2.3a) serve como suporte técnico para a demonstração.

Para $J = 1$, as condições (5.2.3a) e (5.2.3b) são verdadeiras por vacuidade e as condições (5.2.3c) e (5.2.3d) também são já que $Z_1 = E_1$ e $D_{11} = A_{11}$. Então, para o passo de indução, assumimos que $H_1, H_2, \ldots, H_{J-1}$ são verdadeiras no intuito de provar H_J.

Para demonstrar (5.2.3a), multiplicamos pela esquerda a linha 7 do Algoritmo 16 por A_I^T, com $I < J$, obtendo

$$A_I^\mathsf{T} Z_J^{(I)} = A_I^\mathsf{T} Z_J^{(I-1)} - A_I^\mathsf{T} Z_I D_{II}^{-1} R_J^{(I-1)}.$$

Vejamos que o primeiro termo do lado direito da equação é a matriz $R_J^{(I-1)}$ (linha 6 do Algoritmo 16) e, dado que $A_I^\mathsf{T} Z_I = D_{II}$ (linha 4 do Algoritmo 16), concluímos que

$$A_I^\mathsf{T} Z_J^{(I)} = 0 \quad \forall\, I < J. \tag{5.2.4}$$

Agora, de acordo com o Lema 5.2, para $1 \leqslant K < J$,

$$A_K^\mathsf{T} Z_J = A_K^\mathsf{T} Z_J^{(K)} - \sum_{I=K+1}^{J-1} A_K^\mathsf{T} Z_I D_{II}^{-1} R_J^{(I-1)}.$$

Nós acabamos de provar em (5.2.4) que o primeiro termo do lado direito é zero. Como a hipótese de indução garante que $A_K^\mathsf{T} Z_I$ é zero para $1 \leqslant K < I < J$, o somatório no lado direito é zero, provando (5.2.3a).

O que demonstramos até agora garante que a matriz-bloco $Z_{:,1:J}$, de ordem $N \times J$, é bloco-triangular superior e que $AZ_{:,1:J}$ é bloco-triangular inferior. Notemos que $Z_{:,1:J}^\mathsf{T} AZ_{:,1:J}$ é um produto de duas matrizes bloco-triangular inferiores, ou seja $Z_{:,1:J}^\mathsf{T}(AZ_{:,1:J})$, e, portanto, é bloco-triangular inferior. Do mesmo modo, $Z_{:,1:J}^\mathsf{T} AZ_{:,1:J}$ é o produto de duas matrizes bloco-triangular superiores, ou seja $(AZ_{:,1:J})^\mathsf{T} Z_{:,1:J}$, e, portanto, é bloco-triangular superior também. Isto prova (5.2.3b).

Escrevendo a matriz identidade como $I = \sum_{K=1}^{N} E_K E_K^\mathsf{T}$, temos que

$$Z_J^\mathsf{T} A Z_J = \sum_{K=1}^{N} (E_K^\mathsf{T} Z_J)^\mathsf{T} (E_K^\mathsf{T} A) Z_J =$$

$$= \sum_{K=1}^{J-1} (E_K^\mathsf{T} Z_J)^\mathsf{T} A_K^\mathsf{T} Z_J + (E_J^\mathsf{T} Z_J)^\mathsf{T} A_J^\mathsf{T} Z_J + \sum_{K=J+1}^{N} (E_K^\mathsf{T} Z_J)^\mathsf{T} A_K^\mathsf{T} Z_J.$$

Vejamos que o primeiro somatório no último termo é zero, dado pela última hipótese de indução (5.2.3a), e que o segundo somatório vale

zero, de acordo com o Lema 5.1. Então (5.2.3c) segue do Lema 5.1 e da linha 4 do Algoritmo 16.

Notemos que as hipóteses (5.2.3b) e (5.2.3c) implicam que

$$Z_{:,1:J}^{\mathsf{T}} A Z_{:,1:J} = \operatorname{diag}(D_{11}, \ldots, D_{JJ})$$

é uma matriz bloco diagonal com D_{II}, $1 \leqslant I \leqslant J$, na sua bloco-diagonal. Além disso, o Lema 5.1 garante que $Z_{J+1:N,1:J}$ é zero e, com isso,

$$\operatorname{diag}(D_{11}, \ldots, D_{JJ}) = Z_{:,1:J}^{\mathsf{T}} A Z_{:,1:J} = Z_{1:J,1:J}^{\mathsf{T}} A_{1:J:,1:J} Z_{1:J:,1:J}.$$

A matriz $Z_{1:J,1:J}$ é bloco triangular e sua bloco-diagonal é formada por matrizes identidade, e, portanto, é não singular. Como $A_{1:J:,1:J}$, por hipótese, é não singular, então (5.2.3d) é verdadeira.

□

Esta demonstração pode ser adaptada para matrizes formadas por blocos heterogêneos com apenas algumas pequenas modificações. Por questão de simplicidade, não apresentaremos essa demonstração aqui. Esta observação também vale para os Teoremas 5.2 e 5.3.

5.3 BAINV para M-matrizes não singulares

Mesmo que A seja uma matriz esparsa, a matriz Z produzida pelos Algoritmos 16 e 17 pode ser densa. Como em [11], introduzimos no Algoritmo 16 o descarte de blocos (ou entradas) de Z, ver Definição 5.2. O método resultante, apresentado no Algoritmo 18, é chamado de BAINV (block approximate inverse).

Nesta seção, provaremos que o BAINV não falha quando aplicado a M-matrizes. Para isso, inicialmente, precisaremos de algumas definições e propriedades desta classe de matrizes, estudada sistematicamente por Ostrowski [59].

Em [61], Plemmons apresentou 40 caracterizações equivalentes de M-matrizes não singulares. Nós usaremos três delas:

Lema 5.3. *Seja A uma Z-matriz $n \times n$. Então as seguintes condições equivalem à sentença "A é uma M-matriz não singular":*

1. $\det(A_{p:q,p:q}) > 0$ *para todo* $1 \leq p \leq q \leq n$.
2. *A é não singular e* $A^{-1} \geqslant 0$.
3. *As entradas da diagonal de A são positivas e existe uma matriz diagonal D com entradas diagonais positivas tais que AD é estritamente diagonal dominante, isto é,*

$$a_{ii}d_{ii} > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}|d_{jj}, \quad i = 1, 2, \ldots, n.$$

Comentário 3. *Usando o critério de Sylvester, ver [44], se A é uma Z-matriz simétrica então A é M-matriz não singular se e somente se A for simétrica positiva definida.*

Lema 5.4. *Sejam $A,\ B,\ C \in \mathbb{R}^{n\times n}$ tais que $B \leq C$. Se $A \geqslant 0$, então $AB \leq AC$ e $BA \leq CA$. Se $A \leqslant 0$, então $AB \geqslant AC$ e $BA \geqslant CA$.*

Demonstração. A demonstração é direta e portanto será omitida. □

Lema 5.5. *Sejam $A,\ B \in \mathbb{R}^{m\times p}$ e $C,\ D \in \mathbb{R}^{p\times n}$ tais que $|A| \leqslant B$ e $|C| \leqslant D$. Então $-BD \leqslant AC \leqslant BD$.*

Demonstração. A demonstração é direta e portanto será omitida. □

Lema 5.6. *Seja B uma M-matriz não singular e A uma Z-matriz tal que $A \geq B$. Então A é uma M-matriz não singular.*

Demonstração. A prova é uma aplicação direta da Condição 3 do Lema 5.3 e do Lema 5.4.

□

Lema 5.7. *Seja A e B M-matrizes não singulares tais que $A \leq B$. Então $B^{-1} \leq A^{-1}$.*

Demonstração. Dado que $A \leq B$ e, recordando que, da Condição 2 do Lema 5.3, $A^{-1} \geqslant 0$ e $B^{-1} \geqslant 0$. Usando o Lema 5.4, dado que $A \leq B$, temos que $I = A^{-1}A \leq A^{-1}B$. Usando o mesmo lema novamente, $B^{-1} = IB^{-1} \leq A^{-1}BB^{-1} = A^{-1}$.

□

Os dois próximos lemas apresentam algumas propriedades das matrizes D_{JJ} produzidas pelo algoritmo de A-conjugação quando A é SPD ou uma M-matriz.

Lema 5.8. *Se A for SPD, a matriz diagonal em blocos D produzida pelo Algoritmo 16 é SPD.*

Demonstração. O Teorema 5.1 garante que D é igual a $Z^{\mathsf{T}}AZ$ e que Z é não singular. □

Lema 5.9. *Seja A uma M-matriz SPD e D e Z as matrizes resultantes da aplicação do Algoritmo 16 em A. Se $Z \geqslant 0$, então as matrizes D_{JJ}, $J = 1, \ldots, N$, M-matrizes não singulares.*

Demonstração. Pela linha 4 do Algoritmo 16 e Lema 5.1, nós temos

$$D_{JJ} = \sum_{L=1}^{J-1} A_{JL} Z_{LJ} + A_{JJ}.$$

A é Z-matriz, portanto A_{JJ} também é Z-matriz e $A_{JL} \leq 0$, para $J \neq L$. Logo, dado que $Z_{LJ} \geqslant 0$, nós concluímos que D_{JJ} é uma soma de Z-matrizes e, portanto, também é uma Z-matriz. D_{JJ} também é SPD (pois é uma submatriz principal de uma matriz SPD, ver Lema 5.8). Usando a Condição 1 do Lema 5.3, nós concluímos que D_{JJ} é uma M-matriz não singular.

□

Agora apresentaremos o Algoritmo 18, que implementa o BAINV. A única diferença respeito do Algoritmo 16 é a linha 7, aonde são executados descartes para promover esparsidade. A seguir nós definimos tal procedimento.

Definição 5.2. *Para um N-vetor-bloco V, a notação $descarte_I(V)$ indica o procedimento de descartar certas entradas de V, exceto aquelas que pertençam ao I-ésimo bloco, i.e.,*

$$(descarte_I(V))_{ij} = \begin{cases} v_{ij}, & se\ (I-1)b < i \leq Ib \\ v_{ij}\ ou\ 0, & para\ as\ demais\ entradas. \end{cases}$$

De forma análoga, para uma N-linha-bloco V_*, a notação $descarte_I(V_*)$ indica o procedimento de descartar certas entradas de V_*, exceto aquelas que pertençam ao I-ésimo bloco, i.e.,

$$(descarte_I(V_*))_{ij} = \begin{cases} v_{*ij}, & se\ \ (I-1)b < j \leq Ib, \quad 1 \leqslant i \leqslant b, \\ v_{*ij}\ \ ou\ \ 0, & para\ as\ demais\ entradas. \end{cases}$$

O objetivo é produzir um vetor-bloco esparso, mas ainda próximo ao original. Uma estratégia natural é descartar entradas de baixa magnitude. No entanto, outras estratégias podem ser empregadas como descartar de acordo com um padrão de zeros pré-definidos, tolerância absoluta e relativa, descartes por elementos ou por blocos. Nossos resultados teóricos valem para qualquer uma destas estratégias de descarte.

Comentário 4. *Se $A \leq 0$ então $A \leq descarte(A) \leq 0$. Se $B \geq 0$ então $B \geq descarte(B) \geq 0$. Se A é uma Z-matriz então $(descarte(A))_{ij} \geq a_{ij}$, para $i \neq j$.*

Lema 5.10. *Se o Algoritmo 18 não falhar, ele produz uma matriz bloco-triangular superior $\widetilde{Z}$. Além disso, sua bloco diagonal é formada por matrizes identidade.*

Algoritmo 18: BAINV pela direita com descarte

Data: A é matriz em blocos de ordem $N \times N$.
Result: uma matriz não singular diagonal em blocos $\widetilde{D}$ e uma matriz em blocos não singular $\widetilde{Z}$ tais que $A^{-1} \approx \widetilde{Z}\widetilde{D}^{-1}\widetilde{Z}^{\mathsf{T}}$.

1 $\widetilde{Z}_J^{(0)} \leftarrow E_J, \qquad J \in \{1, ..., N\}$;
2 **para** $J \leftarrow 1$ **até** N **faça**
3 $\widetilde{Z}_J \leftarrow \widetilde{Z}_J^{(J-1)}$;
4 $\widetilde{D}_{JJ} \leftarrow A_J^{\mathsf{T}}\widetilde{Z}_J$;
5 **para** $I \leftarrow J+1$ **até** N **faça**
6 $\widetilde{R}_I^{(J-1)} \leftarrow A_J^{\mathsf{T}}\widetilde{Z}_I^{(J-1)}$;
7 $\widetilde{Z}_I^{(J)} \leftarrow descarte_I(\widetilde{Z}_I^{(J-1)} - \widetilde{Z}_J\widetilde{D}_{JJ}^{-1}\widetilde{R}_I^{(J-1)})$;
8 $\widetilde{D} \leftarrow diag(\widetilde{D}_{11}, ..., \widetilde{D}_{NN})$ e $\widetilde{Z} \leftarrow \begin{bmatrix} \widetilde{Z}_1, & \cdots & , \widetilde{Z}_N \end{bmatrix}$;

Demonstração. A prova é análoga à do Lema 5.1. As únicas observações adicionais são que $E_I^T descarte_I(V) = E_I^T V$ e $E_L^T V = 0 \Rightarrow E_L^T descarte_I(V) = 0$. □

Nós estabelecemos todos os elementos necessários para apresentar e demonstrar o primeiro dos dois principais resultados desse capítulo.

Teorema 5.2. *Seja $A \in \mathcal{M}$ uma matriz em blocos $N \times N$, simétrica e não singular. Então o Algoritmo 18 não falha, pois produz blocos $\widetilde{D}_{JJ}$ que são M-matrizes não singulares.*

Demonstração. Pelo Comentário 3, A é SPD. Provaremos por indução. é suficiente provar que as hipóteses de H_J são verdadeiras para cada $J \in \{1, \ldots, N\}$.

$$H_J : \begin{cases} 0 \leq \widetilde{Z}_I^{(J-1)} \leq Z_I^{(J-1)}, & I \in \{J, \ldots, N\}; \quad (5.3.5a) \\ R_I^{(J-1)} \leq \widetilde{R}_I^{(J-1)} \leq 0, & I \in \{J+1, \ldots, N\}; \quad (5.3.5b) \\ \widetilde{D}_{JJ} \geq D_{JJ}; & (5.3.5c) \\ \widetilde{D}_{JJ} \text{ é uma } M\text{-matriz não singular}; & (5.3.5d) \end{cases}$$

Enquanto a hipótese (5.3.5d) é, de fato, o resultado pretendido, as hipóteses (5.3.5a), (5.3.5b), e (5.3.5c) servem como suporte técnico para a demonstração.

Primeiro verifiquemos se H_1 é verdadeira. A linha 1 dos Algoritmos 16 e 18 mostram que $Z_I^{(0)} = \widetilde{Z}_I^{(0)} = E_I$ para $I = 1, \ldots, N$, o que garante (5.3.5a). Para provar (5.3.5b), obervemos que pelas linhas 1 e 6 do Algoritmo 18, $\widetilde{R}_I^{(0)} = A_{1I}$, $I = 2, \ldots, N$, enquanto pelas linhas 1

e 6 do Algoritmo 16, $R_I^{(0)} = A_{1I}$, $I = 2, \ldots, N$. A Equação (5.3.5b) é verdadeira, pois como A é uma M-matriz, então também é uma Z-matriz, logo as entradas de A_{1I} são todas não positivas para $I > 1$. Como $\widetilde{D}_{11} = D_{11}$ e D_{11} é uma M-matriz não singular (Lema 5.9), nós obtemos (5.3.5c) e (5.3.5d).

Agora assumamos que as hipóteses de H_{J-1} são verdadeiras para demonstrar H_J.

Para provarmos (5.3.5a), primeiro vejamos que a linha 7 dos Algoritmos 16 e 18 implica que

$$Z_I^{(J-1)} = Z_I^{(J-2)} - Z_{J-1} D_{(J-1)(J-1)}^{-1} R_I^{(J-2)} \quad \text{e}$$

$$\widetilde{Z}_I^{(J-1)} = descarte_I(\widetilde{Z}_I^{(J-2)} - \widetilde{Z}_{J-1} \widetilde{D}_{(J-1)(J-1)}^{-1} \widetilde{R}_I^{(J-2)}).$$

Temos, pela hipótese de indução, que $0 \leq \widetilde{Z}_I^{(J-2)} \leq Z_I^{(J-2)}$, então, a partir do Comentário 4, basta verificar que

$$0 \geq \widetilde{Z}_{J-1} \widetilde{D}_{(J-1)(J-1)}^{-1} \widetilde{R}_I^{(J-2)} \geq Z_{J-1} D_{(J-1)(J-1)}^{-1} R_I^{(J-2)}.$$

Pela hipótese de indução,

$$Z_{J-1} \geq \widetilde{Z}_{J-1} \geq 0,\ R_I^{(J-2)} \leq \widetilde{R}_I^{(J-2)} \leq 0,\ \widetilde{D}_{(J-1)(J-1)} \geq D_{(J-1)(J-1)}$$

e, pelo Lema 5.7 e pela condição 2 do Lema 5.3, $D_{(J-1)(J-1)}^{-1} \geq \widetilde{D}_{(J-1)(J-1)}^{-1} \geq 0$. Agora, pelo Lema 5.4 e pela definição de descarte, temos

$$\begin{aligned} 0 \leqslant \widetilde{Z}_I^{(J-1)} &= descarte_I(\widetilde{Z}_I^{(J-2)} - \widetilde{Z}_{J-1} \widetilde{D}_{(J-1)(J-1)}^{-1} \widetilde{R}_I^{(J-2)}) \leqslant \\ &\leqslant \widetilde{Z}_I^{(J-2)} - \widetilde{Z}_{J-1} \widetilde{D}_{(J-1)(J-1)}^{-1} \widetilde{R}_I^{(J-2)} \leqslant \\ &\leqslant Z_I^{(J-2)} - Z_{J-1} D_{(J-1)(J-1)}^{-1} R_I^{(J-2)} = Z_I^{(J-1)},\ \text{para } I \in J, \ldots, N, \end{aligned}$$

provando (5.3.5a).

Para provar (5.3.5b), vejamos que pela linha 6 dos Algoritmos 16 e 18, junto com os Lemas 5.1 e 5.10, temos que

$$R_I^{(J-1)} = A_{JI} + \sum_{L=1}^{J-1} A_{JL} Z_{LI}^{(J-1)} \quad \text{e}$$

$$\widetilde{R}_I^{(J-1)} = A_{JI} + \sum_{L=1}^{J-1} A_{JL} \widetilde{Z}_{LI}^{(J-1)}.$$

Como $J < I$ e A é uma Z-matriz, $A_{JI} \leq 0$, e portanto é suficiente mostrar que $A_{JL} Z_{LI}^{(J-1)} \leq A_{JL} \widetilde{Z}_{LI}^{(J-1)} \leq 0$. Já que, $A_{JL} \leq 0$ (entradas

fora da diagonal de uma Z-matriz) e, de (5.3.5a) , $Z_{LI}^{(J-1)} \geq \widetilde{Z}_{LI}^{(J-1)} \geq 0$. Com estes resultados e o Lema 5.4 provamos (5.3.5b).

Para provar (5.3.5c), vejamos que as linhas 4 dos Algoritmos 16 e 18 implicam que

$$D_{JJ} = A_{JJ} + \sum_{L=1}^{J-1} A_{JL} Z_{LJ}^{(J-1)} \quad \text{e}$$

$$\widetilde{D}_{JJ} = A_{JJ} + \sum_{L=1}^{J-1} A_{JL} \widetilde{Z}_{LJ}^{(J-1)}.$$

Notemos que, como $A_{JL} \leq 0$ e $Z_{LJ}^{(J-1)} \geq \widetilde{Z}_{LJ}^{(J-1)} \geq 0$, usando o Lema 5.4, temos que

$$\sum_{L=1}^{J-1} A_{JL} Z_{LJ}^{(J-1)} \leq \sum_{L=1}^{J-1} A_{JL} \widetilde{Z}_{LJ}^{(J-1)} \leq 0 \Rightarrow D_{JJ} \leqslant \widetilde{D}_{JJ}.$$

Finalmente, para provar (5.3.5d), vejamos primeiramente que o Lema 5.9 garante que D_{JJ} é uma M-matriz não singular. No parágrafo anterior, mostramos que $\sum_{L=1}^{J-1} A_{JL} Z_{LJ}^{(J-1)} \leq \sum_{L=1}^{J-1} A_{JL} \widetilde{Z}_{LJ}^{(J-1)} \leq 0$. Além disso, como A é uma M-matriz, então A_{JJ} é uma Z-matriz, portanto suas entradas fora da diagonal são não positivas. Logo as entradas fora da diagonal de $\widetilde{D}_{JJ}$ são não positivas, dessa forma $\widetilde{D}_{JJ}$ também é uma Z-matriz. Como D_{JJ} é uma M-matriz não singular, pela assertiva (5.3.5c) e pelo Lema 5.6, concluímos que $\widetilde{D}_{JJ}$ é uma M-matriz não singular. □

5.4 BAINV para H-matrizes não singulares

Nesta seção, provaremos que o Algoritmo 18 também produz matrizes não singulares quando aplicado a H-matrizes com matriz comparação não singular.

Lema 5.11. *Se A é uma H-matriz e $\mathcal{C}(A)$ é não singular então A é não singular. Ver [23].*

Na sequência, seguindo Plemmons em [61], consideraremos apenas H-matrizes que possuem M-matrizes não singulares como suas matrizes comparação.

Lema 5.12 (Desigualdade de Ostrowsky's)**.** *Se A for uma H-matriz não singular tal que $\mathcal{C}(A)$ é uma M-matriz não singular então $|A^{-1}| \leq \mathcal{C}(A)^{-1}$. Ver [48].*

Lema 5.13. *Seja $A \in \mathbb{R}^{n\times n}$ uma H-matriz simétrica com diagonal principal de entradas positivas e tal que $\mathcal{C}(A)$ é uma M-matriz não singular. Então A é simétrica positiva definida. Ver [24].*

A seguir demonstraremos um dos principais resultados deste capítulo.

Teorema 5.3. *Seja a matriz em blocos A uma H-matriz simétrica com diagonal principal formada por entradas positivas e tal que $\mathcal{C}(A)$ é M-matriz não singular. Então o Algoritmo 18 não falha, pois produz blocos $\widetilde{D}_{JJ}$ que são H-matrizes não singulares.*

Demonstração. Primeiramente vejamos algumas notações. Além dos símbolos D_{JJ}, $Z_J^{(K)}$ e $S_J^{(K)}$ para as matrizes geradas pelo Algoritmo 16 aplicado a A e $\widetilde{D}_{JJ}$, $\widetilde{Z}_J^{(K)}$ e $\widetilde{R}_J^{(K)}$ para aquelas geradas pelo Algoritmo 18 aplicado a A, nós também introduzimos as notações $\widehat{D}_{JJ}$, $\widehat{Z}_J^{(K)}$ e $\widehat{M}_J^{(K)}$ para as matrizes geradas pelo Algoritmo 18 sem descartes[1] aplicado à matriz comparação $\mathcal{C}(A)$. Finalmente, $\mathcal{C}(A)_{IJ}$ denota $(\mathcal{C}(A))_{IJ}$, não $\mathcal{C}(A_{IJ})$.

Provaremos por indução. é suficiente provar que as hipóteses de H_J são verdadeira para cada $J \in \{1,\ldots,N\}$.

$$H_J : \begin{cases} |\widetilde{Z}_I^{(J-1)}| \le \widehat{Z}_I^{(J-1)}, & I \in \{J,\ldots,N\}; \quad (5.4.6a)\\ |\widetilde{R}_I^{(J-1)}| \le -\widehat{M}_I^{(J-1)}, & I \in \{J+1,\ldots,N\}; \quad (5.4.6b)\\ \widehat{D}_{JJ} \le \mathcal{C}(\widetilde{D}_{JJ}); & (5.4.6c)\\ \widetilde{D}_{JJ} \text{ é uma } H\text{-matriz não singular.} & (5.4.6d) \end{cases}$$

A hipótese (5.4.6d) demonstra o resultado pretendido, enquanto as outras hipóteses servem como suporte técnico para a demonstração.

Primeiro verifiquemos se H_1 é verdadeira. A hipótese (5.4.6a) é verdadeira, tornando-se uma igualdade, para $J = 1$, pois $\widetilde{Z}_I^{(0)} = \widehat{Z}_I^{(0)} = E_I$, $1 \le I \le N$. Também a hipótese (5.4.6b) é verdadeira, tornando-se uma igualdade, para $J = 1$, pois para $I \ge 2$ we temos que $\mathcal{C}(A)_{1I} = -|A_{1I}|$ e, a partir das linhas 1 e 6 do Algoritmo 18, $\widetilde{R}_I^{(0)} = A_{1I}$ e $\widehat{M}_I^{(0)} = \mathcal{C}(A)_{1I}$. Da linha 4 do Algoritmo 18, $\widetilde{D}_{11} = A_{11}$ e $\widehat{D}_{11} = \mathcal{C}(A)_{11}$. Para provar que (5.4.6c) é verdadeira, como uma igualdade, note que $\widehat{D}_{11} = \left(\mathcal{C}(A)\right)_{11} = \mathcal{C}(A_{11}) = \mathcal{C}\left(\widetilde{D}_{11}\right)$. Provando a hipótese (5.4.6d): primeiramente, $\widetilde{D}_{11} = A_{11}$ é uma matriz não singular e, pelo Lema 5.13, é uma submatriz principal de uma matriz SPD. $\mathcal{C}(A)$ é M-matriz não

[1] Vejamos que não executar nenhum descarte, isto é, $descarte_I(V) = V$, é uma estratégia de descarte válida, de acordo com a equação (5.2). Vejamos, então, que o Algoritmo 18 com esta estratégia de descarte (sem executar descartes) equivale ao Algoritmo 16. Porém, preferimos mostrá-los separadamente, já que usaremos alguns resultados da Seção 5.3.

singular e, pelo Comentário 3, é uma matriz SPD, portanto $\mathcal{C}(A)_{11}$ é uma matriz SPD pois é uma submatriz principal de $\mathcal{C}(A)$, e, como dito anteriormente, $\mathcal{C}(A)_{11} = \mathcal{C}(A_{11})$. $\mathcal{C}(A_{11})$ é SPD e uma Z-matriz, e, pelo Comentário 3, é uma M-matriz não singular. Portanto $A_{11} = \widetilde{D}_{11}$ é uma H-matriz.

Agora para o passo de indução, assumamos que as hipóteses H_{J-1} são verdadeiras para demonstrar H_J.

O bloco $\widehat{D}_{(J-1)(J-1)}$ é uma M-matriz não singular (ver demonstração do Teorema 5.2). Pela hipótese de indução (5.4.6c), $\widehat{D}_{(J-1)(J-1)} \leq \mathcal{C}(\widetilde{D}_{(J-1)(J-1)})$, então, pelo Lema 5.6, $\mathcal{C}(\widetilde{D}_{(J-1)(J-1)})$ é M-matriz não singular. Temos que $\left(\mathcal{C}(\widetilde{D}_{(J-1)(J-1)})\right)^{-1} \leqslant \widehat{D}^{-1}_{(J-1)(J-1)}$, pelo Lema 5.7, e $\left|\widetilde{D}^{-1}_{(J-1)(J-1)}\right| \leq \left(\mathcal{C}(\widetilde{D}_{(J-1)(J-1)})\right)^{-1}$, pelo Lema 5.12.

Então

$$\left|\widetilde{D}^{-1}_{(J-1)(J-1)}\right| \leqslant \widehat{D}^{-1}_{(J-1)(J-1)}. \tag{5.4.7}$$

Como $|descarte_I(V)| \leq |V|$ para qualquer índice I e qualquer vetor-bloco V e considerando as hipóteses de indução (5.4.6a) e (5.4.6b) assim como a equação (5.4.7), podemos provar (5.4.6a) ao examinarmos a linha 7 do Algoritmo 18:

$$\begin{aligned}
\left|\widetilde{Z}_I^{(J-1)}\right| &= \left|descarte_I(\widetilde{Z}_I^{(J-2)} - \widetilde{Z}_{J-1}\widetilde{D}^{-1}_{(J-1)(J-1)}\widetilde{R}_I^{(J-2)})\right| \\
&\leq \left|\widetilde{Z}_I^{(J-2)} - \widetilde{Z}_{J-1}\widetilde{D}^{-1}_{(J-1)(J-1)}\widetilde{R}_I^{(J-2)}\right| \\
&\leq \left|\widetilde{Z}_I^{(J-2)}\right| + \left|\widetilde{Z}_{J-1}\right| \left|\widetilde{D}^{-1}_{(J-1)(J-1)}\right| \left|\widetilde{R}_I^{(J-2)}\right| \\
&\leq \widehat{Z}_I^{(J-2)} - \widehat{Z}_{J-1}\widehat{D}^{-1}_{(J-1)(J-1)}\widehat{M}_I^{(J-2)} = \widehat{Z}_I^{(J-1)}.
\end{aligned}$$

Para provar (5.4.6b), primeiro recordemos que, para $I > J$,

$$\left|\widetilde{R}_I^{(J-1)}\right| = \left|A_{JI} + \sum_{L=1}^{J-1} A_{JL}\widetilde{Z}_{LI}^{(J-1)}\right| \leq |A_{JI}| + \sum_{L=1}^{J-1} |A_{JL}| \left|\widetilde{Z}_{LI}^{(J-1)}\right|.$$

Como os termos $|A_{JI}|$ e $|A_{JL}|$ acima não são blocos diagonais, eles podem ser substituídos por $-\mathcal{C}(A)_{JI}$ e $-\mathcal{C}(A)_{JL}$, respectivamente. Em relação ao termo $\left|\widetilde{Z}_{LI}^{(J-1)}\right|$ acima, ele é o L-ésimo bloco de $\left|\widetilde{Z}_I^{(J-1)}\right|$, então nós podemos usar (5.4.6a) para concluir que $\left|\widetilde{Z}_{LI}^{(J-1)}\right| \leq \widehat{Z}_{LI}^{(J-1)}$. Portanto

$$\left|\widetilde{R}_I^{(J-1)}\right| \leq -\mathcal{C}(A)_{JI} - \sum_{L=1}^{J-1} \mathcal{C}(A)_{JL}\, \widehat{Z}_{LI}^{(J-1)} = -\widehat{M}_I^{(J-1)}.$$

Para provar (5.4.6c), devemos comparar os blocos

$$\begin{aligned}\widehat{D}_{JJ} &= \mathcal{C}(A)_{JJ} + \sum_{I=1}^{J-1} \mathcal{C}(A)_{JI}\,\widehat{Z}_{IJ} \quad \text{e} \\ \mathcal{C}\left(\widetilde{D}_{JJ}\right) &= \mathcal{C}\left(A_{JJ} + \sum_{I=1}^{J-1} A_{JI}\widetilde{Z}_{IJ}\right).\end{aligned}$$

Primeiramente percebamos que, usando $|A_{JI}| = -\mathcal{C}(A)_{JI}$ e a hipótese de indução (5.4.6a), o Lema 5.5 garante que

$$\mathcal{C}(A)_{JI}\,\widehat{Z}_{IJ} \leq A_{JI}\widetilde{Z}_{IJ} \leq -\mathcal{C}(A)_{JI}\,\widehat{Z}_{IJ}.$$

Agora comparemos as entradas da diagonal principal. Como $\mathcal{C}(A)$ é uma M-matriz não singular, (5.3.5d) implica que $\widehat{D}_{JJ}$ é uma M-matriz não singular e portanto sua diagonal é formada de entradas positivas, de acordo com o Lema 5.3. Com isso temos:

$$\begin{aligned}0 < \left(\widehat{D}_{JJ}\right)_{ii} &= \left(\mathcal{C}(A)_{JJ}\right)_{ii} + \sum_{I=1}^{J-1} \left(\mathcal{C}(A)_{JI}\,\widehat{Z}_{IJ}\right)_{ii} \\ &\leq \left(A_{JJ}\right)_{ii} + \sum_{I=1}^{J-1} \left(A_{JI}\widetilde{Z}_{IJ}\right)_{ii} \\ &= \left(\widetilde{D}_{JJ}\right)_{ii} = \left(\mathcal{C}\left(\widetilde{D}_{JJ}\right)\right)_{ii}.\end{aligned}$$

Para as entradas fora da diagonal, primeiro vejamos que, pela definição de matriz comparação, $\left(\mathcal{C}(A)_{JJ}\right)_{ij} \leq (A_{JJ})_{ij} \leq -\left(\mathcal{C}(A)_{JJ}\right)_{ij}$ para $i \neq j$. Junto com as desigualdades em (5.4) e as fórmulas (5.4.8) e (5.4.8), concluímos que $\left(\widehat{D}_{JJ}\right)_{ij} \leqslant \left(\widetilde{D}_{JJ}\right)_{ij} \leqslant -\left(\widehat{D}_{JJ}\right)_{ij}$ e portanto $\left(\widehat{D}_{JJ}\right)_{ij} \leq -\left|\left(\widetilde{D}_{JJ}\right)_{ij}\right|$. Finalmente, notemos que $-\left|\left(\widetilde{D}_{JJ}\right)_{ij}\right| = \left(\mathcal{C}\left(\widetilde{D}_{JJ}\right)\right)_{ij}$ e, assim, provamos (5.4.6c).

Nós acabamos de provar que $\mathcal{C}\left(\widetilde{D}_{JJ}\right) \geq \widehat{D}_{JJ}$. Como $\mathcal{C}\left(\widetilde{D}_{JJ}\right)$ é uma Z-matriz e por (5.3.5d) $\widehat{D}_{JJ}$ M-matriz não singular, o Lema 5.6 garante que $\mathcal{C}\left(\widetilde{D}_{JJ}\right)$ é uma M-matriz não singular e portanto $\widetilde{D}_{JJ}$ é uma H-matriz. Além disso, o Lema 5.11 garante que $\widetilde{D}_{JJ}$ é não singular e, então, (5.4.6d) é provado.

□

Exercício 5.1. *Considere a matriz A tal que:*

$$A = \begin{bmatrix} 2,00 & 0,40 & 0,10 & 0,00 \\ 0,40 & 1,08 & 2,00 & 0,00 \\ 0,10 & 2,00 & 3,96 & 0,00 \\ 0,00 & 0,00 & 0,00 & 1,00 \end{bmatrix}$$

Mostre que A é uma matriz SPD. Mostre que A não é uma H-matriz.

Exercício 5.2. *Mostre que o Algoritmo de Inversa Aproximada por Blocos (BAINV), com descarte por tolerância de 0,06 falha para tamanho de bloco igual a 1, isto é, produz um pivô nulo. Mais explicitamente, mostre que $\tilde{p}_3 = 0$. Mostre que, para o tamanho de bloco igual a 2, para o descarte de bloco (ou de entrada a entrada) com tolerância de 0,06 o BAINV não falha.*

Exercício 5.3. *Mostre que a submatriz principal de uma M-matriz é uma M-matriz. Conclua que a submatriz principal de uma H-matriz também é uma H-matriz.*

Capítulo 6

Inversa Aproximada em Blocos para Matrizes Não Simétricas

Neste capítulo, propomos algumas adaptações do BAINV para matrizes não simétricas, demonstrando sua consistência. Uma delas baseia-se no SAINV para o caso escalar, sendo livre de falha para matrizes positivas definidas. Além dessa, também propomos versões em blocos para o FFAPINV, SAINV-VAR e RIF-NS.

6.1 A-biconjugação em blocos

Nesta seção, propomos o método de A-biconjugação em blocos para matrizes não simétricas, sendo esse método a base para a inversa aproximada em blocos para matrizes não simétricas. Seja $A \in \mathbb{R}^{n\times n}$ uma matriz não simétrica e não singular em blocos $N \times N$. O objetivo é encontrar as matrizes não singulares Z, W e D tais que $WAZ = D$, ou $A^{-1} = ZD^{-1}W$, em que D é bloco-diagonal, e Z e W são bloco triangulares-superiores e inferiores, respectivamente. O método de A-biconjugação em blocos busca fornecer dois conjuntos de bloco-vetores $\{Z_I\}_{I=1}^{N}$ e bloco-linhas $\{W_I*\}_{I=1}^{N}$ que sejam A-biconjugados, ou seja, tais que $W_I * AZ_J = 0$ se $I \neq J$. $\{Z_I\}_{I=1}^{N}$ e $\{W_I*\}_{I=1}^{N}$ são os bloco-vetores e bloco-linhas de Z e W, respectivamente.

A seguir, temos os Algoritmos 19 e 20 que são as versões pela direita e pela esquerda, respectivamente, do algoritmo de A-biconjugação em blocos. É fácil verificar que ambos os algoritmos produzem os mesmos resultados finais e parciais, podendo ser provado conforme à versão escalar, no Capítulo 2. Para demonstrar os resultados deste capítulo, nos basearemos apenas na versão pela direita, já que para a versão pela esquerda isso pode ser feito de forma análoga. Vejamos alguns resultados.

Algoritmo 19: A-biconjugação em blocos pela direita

Dados: matriz A em blocos $N \times N$ não singular.
Resultado: D, Z e W não singulares com $A^{-1} = ZD^{-1}W$.

$Z_I^{(0)} \leftarrow E_I, \qquad I \in \{1, ..., N\}$;
$W_{I*}^{(0)} \leftarrow E_I^T, \qquad I \in \{1, ..., N\}$
para $I \leftarrow 1$ **até** N **faça**
  $Z_I \leftarrow Z_I^{(I-1)}$; $W_{I*} \leftarrow W_{I*}^{(I-1)}$;
  $D_{II} \leftarrow A_{I*}Z_I$;
  para $J \leftarrow I+1$ **até** N **faça**
    $R_J^{(I-1)} \leftarrow A_{I*}Z_J^{(I-1)}$;
    $S_J^{(I-1)} \leftarrow W_{J*}^{(I-1)}A_I$;
    $Z_J^{(I)} \leftarrow Z_J^{(I-1)} - Z_I D_{II}^{-1} R_J^{(I-1)}$;
    $W_{J*}^{(I)} \leftarrow W_{J*}^{(I-1)} - S_J^{(I-1)} D_{II}^{-1} W_{I*}$;
$D \leftarrow \text{diag}(D_{11}, \ldots, D_{NN})$, $Z \leftarrow \left[\; Z_1, \quad \cdots \quad , Z_N \; \right]$ e

$$W \leftarrow \begin{bmatrix} W_{1*} \\ \vdots \\ W_{N*} \end{bmatrix};$$

Algoritmo 20: A-biconjugação em blocos pela esquerda

Dados: matriz A em blocos $N \times N$ não singular.
Resultado: D, Z e W não singulares com $A^{-1} = ZD^{-1}W$.

$Z_1 \leftarrow E_1$;
$W_{1*} \leftarrow E_1^T$;
$D_{11} \leftarrow A_{11}$;
para $J \leftarrow 2$ **até** N **faça**
  $Z_J^{(0)} \leftarrow E_J$;
  $W_{J*}^{(0)} \leftarrow E_J^T$;
  para $I \leftarrow 1$ **até** $J-1$ **faça**
    $R_J^{(I-1)} \leftarrow A_{I*}Z_J^{(I-1)}$;
    $S_J^{(I-1)} \leftarrow W_{J*}^{(I-1)}A_I$;
    $Z_J^{(I)} \leftarrow Z_J^{(I-1)} - Z_I D_{II}^{-1} R_J^{(I-1)}$;
    $W_{J*}^{(I)} \leftarrow W_{J*}^{(I-1)} - S_J^{(I-1)} D_{II}^{-1} W_{I*}$;
  $Z_J \leftarrow Z_J^{(J-1)}$; $W_{J*} \leftarrow W_{J*}^{(J-1)}$
  $D_{JJ} \leftarrow A_{J*}Z_J$;
$D \leftarrow \text{diag}(D_{11}, \ldots, D_{NN})$, $Z \leftarrow \left[\; Z_1, \quad \cdots \quad , Z_N \; \right]$ e

$$W \leftarrow \begin{bmatrix} W_{1*} \\ \vdots \\ W_{N*} \end{bmatrix};$$

Lema 6.1. *Caso o Algoritmo 16 não falhe (i.e., D_{II}'s singulares não são gerados), ele fornece uma matriz Z bloco-triangular superior e uma matriz W bloco triangular-inferior. Além disso, os blocos-diagonais de Z e W são compostos pela identidade.*

Demonstração. Façamos por indução em I, com $I = 0, 1, \ldots, N-1$, para provar que

$$E_L^T Z_J^{(I)} = W_{J*}^{(I)} E_L = \delta_{LJ} \quad \forall\, J, L \text{ tal que } I < J \leq L \leq N, \qquad (6.1.1)$$

onde δ_{LJ} é dado como

$$\delta_{LJ} = \begin{cases} \text{bloco identidade}, & \text{se } L = J \\ 0, & \text{se } L \neq J \end{cases}.$$

A expressão (5.2.2) é notavelmente verdadeira para $I = 0$, já que $Z_J^{(0)} = E_J$ e $W_{J*}^{(0)} = E_J^T$. Agora, assumamos que (6.1.1) é verdadeira para $0 \leq I < N-1$. A partir disso e da linha 9 do Algoritmo 19,

$$E_L^T Z_J^{(I+1)} = E_L^T Z_J^{(I)} - E_L^T Z_{I+1}^{(I)} D_{(I+1)(I+1)}^{-1} R_J^{(I)} = E_L^T Z_J^{(I)} = \delta_{LJ},$$

e

$$W_{J*}^{(I+1)} E_L = W_{J*}^{(I)} E_L - S_J^{(I)} D_{(I+1)(I+1)}^{-1} W_{I+1*}^{(I)} E_L = W_{J*}^{(I)} E_L = \delta_{LJ},$$

o que completa a indução. □

Lema 6.2. *Se o Algoritmo 19 não falhar, temos que, para qualquer $1 \leq J \leq N$ fixo,*

$$Z_J = Z_J^{(K)} - \sum_{I=K+1}^{J-1} Z_I D_{II}^{-1} R_J^{(I-1)}, \qquad W_{J*} = W_{J*}^{(K)} - \sum_{I=K+1}^{J-1} S_J^{(I-1)} D_{II}^{-1} W_{I*}$$

para todo $0 \leq K \leq I-1$.

Demonstração. Segue diretamente das linhas 4, 9 e 10 do Algoritmo 19. □

A seguir demonstramos, no Teorema 6.1, que sob determinadas condições, o Algoritmo 19 realmente produz a decomposição $A^{-1} = ZD^{-1}W$.

Teorema 6.1. *Seja A uma matriz $N \times N$ em bloco tal que $A_{1:J,1:J}$ é não singular para $J = 1, 2, \ldots, N$. Então, o Algoritmo 19 não falha e retorna uma matriz bloco-diagonal D não singular e matrizes em bloco Z e W não singulares tais que $A^{-1} = ZD^{-1}W$ (ou, de forma equivalente, $WAZ = D$).*

Demonstração. Vamos fazer por indução. É suficiente provar que as hipóteses em H_J são verdadeiras para todo $J \in \{1, \ldots, N\}$.

$$H_J : \begin{cases} A_{I*}Z_J = 0, & I \in \{1, 2, \ldots, J-1\}; \quad (6.1.2a) \\ W_{J*}A_I = 0, & I \in \{1, 2, \ldots, J-1\}; \quad (6.1.2b) \\ W_{I*}AZ_J = W_{J*}AZ_I = 0, & I \in \{1, 2, \ldots, J-1\}; \quad (6.1.2c) \\ W_{J*}AZ_J = D_{JJ}; & (6.1.2d) \\ W_{J*}A_J = D_{JJ}; & (6.1.2e) \\ D_{JJ} \text{ é não singular}; & (6.1.2f) \end{cases}$$

As hipóteses (6.1.2c), (6.1.2d), e (6.1.2f) são suficientes para chegar no resultado desejado. Já as hipóteses (6.1.2a), (6.1.2b), e (6.1.2e) servem como suporte técnico para a demonstração.

Para $J = 1$, as condições (6.1.2a), (6.1.2b) e (6.1.2c) são alcançadas por vacuidade e as condições (6.1.2d), (6.1.2e) e (6.1.2f) seguem pelos fatos de que $Z_1 = E_1$, $W_{1*} = E_1^T$ e $D_{11} = A_{11}$. Para o passo de indução, assumimos que $H_1, H_2, \ldots, H_{J-1}$ são verdadeiras e utilizadas para demonstrar H_J.

Para provar (6.1.2a), nós primeiramente multiplicamos a linha 9 do Algorimto 19 por A_{I*} para $I < J$, obtendo

$$A_{I*}Z_J^{(I)} = A_{I*}Z_J^{(I-1)} - A_{I*}Z_I D_{II}^{-1} R_J^{(I-1)}.$$

Temos que o primeiro termo do lado direito é $R_J^{(I-1)}$ (linha 7 do Algoritmo 19) e, sendo $A_{I*}Z_I = D_{II}$ (linha 5 do Algoritmo 19), nós concluímos que

$$A_{I*}Z_J^{(I)} = 0 \quad \forall\, I < J.$$

Agora, a partir do Lema 6.2, para $1 \leqslant K < J$,

$$A_{K*}Z_J = A_{K*}Z_J^{(K)} - \sum_{I=K+1}^{J-1} A_{K*}Z_I D_{II}^{-1} R_J^{(I-1)}.$$

Foi provado em (14) que o primeiro termo do lado direito da expressão é zero. Como a hipótese de indução garante que $A_{K*}Z_I$ é zero para $1 \leqslant K < I < J$, o somatório do lado direito também é zero, provando (6.1.2a).

Analogamente, para provar (6.1.2b) primeiramente multiplicamos do lado direito a linha 10 do Algoritmo 19 por A_I para $I < J$, obtendo

$$W_{J*}^{(I)}A_I = W_{J*}^{(I-1)}A_I - S_J^{(I-1}D_{II}^{-1}W_{I*}A_I.$$

Temos que o primeiro termo do lado direito é $S_J^{(I-1)}$ (linha 8 do Algoritmo 19) e, sendo $W_{I*}A_I = D_{II}$ (pelas hipóteses de indução (6.1.2e)),

concluímos que

$$W_{J*}^{(I)} A_I = 0 \quad \forall\, I < J. \tag{6.1.3}$$

Agora, a partir do Lema 6.2, para $1 \leqslant K < J$,

$$W_{J*} A_K = W_{J*}^{(K)} A_K - \sum_{I=K+1}^{J-1} S_J^{(I-1)} D_{II}^{-1} W_{I*} A_K.$$

Foi provado em (6.1.3) que o primeiro termo do lado direito da expressão é zero. Como a hipótese de indução garante que $W_{I*}A_K$ é zero para $1 \leqslant K < I < J$, o somatório do lado direito também é zero, provando (6.1.2b).

A partir do que já foi provado, temos que a $N \times J$ bloco-matriz Z_J e $W_{1:J,:}A$ são bloco-triangulares superiores e a $J \times N$ bloco matriz $W_{1:J,:}$ e AZ_J são bloco triangular inferiores. Notemos que $W_{1:J,:}AZ_J$ é o produto de duas matrizes bloco-triangular inferiores, dado por $W_{1:J,:}(AZ_J)$, resultando em uma matriz bloco-triangular inferior. Da mesma forma, $W_{1:J,:}AZ_J$ é o produto de duas matrizes bloco-triangular superiores, dado por $(W_{1:J,:}A)Z_J$, resultando em uma matriz bloco-triangular superior. Isto prova (6.1.2c).

Escrevendo a matriz identidade como $I = \sum_{K=1}^{N} E_K E_K^T$, temos que

$$W_{J*} A Z_J = \sum_{K=1}^{N} (W_{J*}E_K)(E_K^T A) Z_J =$$

$$= \sum_{K=1}^{J-1} (W_{J*}E_K) A_{K*} Z_J + (W_{J*}E_J) A_{J*} Z_J + \sum_{K=J+1}^{N} (W_{J*}E_K) A_{K*} Z_J.$$

Notemos que o primeiro somatório do último termo é igual a zero por conta da hipótese de indução (6.1.2a) e que o segundo somatório é zero de acordo com o Lema 6.1. Então chegamos ao (6.1.2d) pelo Lema 6.1 e linha 5 do Algoritmo 19.

A partir de (6.1.2d), Lema 6.1 e linha 1 do Algoritmo 19 nós temos

$$D_{JJ} = W_{J*} A Z_J = W_{J*} A E_J - \sum_{I=1}^{J-1} W_{J*} A Z_I D_{II}^{-1} R_J^{(I-1)}.$$

O somatório no último termo é zero por conta de (6.1.2c). Portanto, temos $D_{JJ} = W_{J*}A_J$, provando (6.1.2e).

A hipóteses (6.1.2c) e (6.1.2d) resultam que

$$W_{1:J,:} A Z_{:,1:J} = diag(D_{11}, \ldots, D_{JJ}),$$

uma matriz bloco-diagonal com D_{II}, $1 \leqslant I \leqslant J$, na sua diagonal em blocos. Além disso, o Lema 6.1 garante que $Z_{J+1:N,1:J}$ e $W_{1:J,J+1;N}$ são zero. Desta forma,

$$diag(D_{11}, \ldots, D_{JJ}) = W_{1:J,:}AZ_{:,1:J} = W_{1:J,1:J}A_{1:J:,1:J}Z_{1:J:,1:J}.$$

As matrizes $Z_{1:J,1:J}$ e $W_{1:J,1:J}$ são bloco-triangulares e seus blocos diagonais são formados por identidades, portanto elas são não singulares. Como, por hipótese, $A_{1:J:,1:J}$ é também não singular, chegamos a (6.1.2f).

□

No Corolário 6.1 temos mais uma opção para o cálculo dos blocos D_{II} do Teorema 6.1.

Corolário 6.1. *Considerando os elementos do Algoritmo 19, temos que* $D_{II} = W_{I*}A_I$.

O resultado a seguir verifica que as relações (2.3.12) e (2.7.20) do caso escalar também valem para a versão em blocos.

Proposição 7. *Seja A matriz em blocos* $N \times N$ *esparsa e não singular. Considerando o Algoritmo 19 aplicado a A, temos*

$$R_J^{(K-1)} = W_{K*}A_J, \tag{6.1.4}$$

$$S_J^{(K-1)} = A_{J*}Z_K. \tag{6.1.5}$$

Demonstração. Pelas linhas 1, 2, 9 e 10 do Algoritmo 19, temos que

$$Z_J = E_J - \sum_{I=1}^{J-1} Z_I D_{II}^{-1} R_J^{(I-1)} \text{ e } W_{J*} = E_J^T - \sum_{I=1}^{J-1} S_J^{(I-1)} D_{II}^{-1} W_{I*}.$$

Seja $2 \leq J \leq N$ fixo. Como $W_{K*}AZ_J = 0$ para todo $K \neq J$ (pois $D = WAZ$), então, para $K = 1, ..., J-1$,

$$\begin{array}{rcl} 0 = W_{K*}AZ_J & \Rightarrow & 0 = W_{K*}A(E_J - \sum_{I=1}^{J-1} Z_I D_{II}^{-1} R_J^{(I-1)}) \\ & \Rightarrow & 0 = W_{K*}A_J - \sum_{I=1}^{J-1} W_{K*}AZ_I D_{II}^{-1} R_J^{(I-1)} \\ & \Rightarrow & 0 = W_{K*}A_J - W_{K*}AZ_K D_{KK}^{-1} R_J^{(K-1)} \\ & \Rightarrow & 0 = W_{K*}A_J - R_J^{(K-1)} \\ & \Rightarrow & R_J^{(K-1)} = W_{K*}A_J \end{array} .$$

Analogamente,

$$\begin{aligned} 0 = W_{J*}AZ_K \quad &\Rightarrow \quad 0 = (E_J^T - \textstyle\sum_{I=1}^{J-1} S_J^{(I-1)} D_{II}^{-1} W_{I*})AZ_K \\ &\Rightarrow \quad 0 = A_{J*}Z_K - \textstyle\sum_{I=1}^{J-1} S_J^{(I-1)} D_{II}^{-1} W_{I*}AZ_K \\ &\Rightarrow \quad 0 = A_{J*}Z_K - S_J^{(K-1)} D_{KK}^{-1} W_{K*}AZ_K \\ &\Rightarrow \quad 0 = A_{J*}Z_K - S_J^{(K-1)} \\ &\Rightarrow \quad S_J^{(K-1)} = A_{J*}Z_K \end{aligned} \quad ,$$

obtendo (6.1.5). □

Os próximos resultados mostram que, assim como no caso escalar, podemos relacionar os fatores da inversa aproximada em blocos de A com os fatores LDU em blocos de A.

Proposição 8. *Seja A uma matriz em blocos $N \times N$ não singular. Considere a fatoração bloco LDU de A como $A = LDU$, tal que L, D e U são matrizes em blocos $N \times N$ não singulares, em que L e U são bloco triangulares inferior e superior, respectivamente, cujos blocos diagonais são iguais à identidade e D é bloco diagonal. Então L, D e U são únicas.*

Demonstração. Ver exercício 6.2.

□

Seja a decomposição

$$A = W^{-1}DZ^{-1}, \tag{6.1.6}$$

onde Z é bloco triangular superior e W é bloco triangular inferior, ambas com identidades na diagonal e D é bloco diagonal não singular, sendo os blocos de tamanho t. Então (6.1.6) pode ser considerada a decomposição bloco LDU de A com blocos de tamanho t. Como essa decomposição é única (Proposição 8), temos que:

$$W = L^{-1}, \quad Z = U^{-1}$$

e D é a mesma matriz.

Proposição 9. *Seja A uma matriz em blocos $N \times N$ esparsa e não singular. Seja a fatoração LDU de A em blocos, em que L e U são bloco triangulares inferior e superior, respectivamente, não singulares, com identidades nas diagonais e D é bloco diagonal não singular. Então, o Algoritmo 16 quando aplicado a A, sem os descartes, produz*

$$U_{IJ} = D_{II}^{-1} R_J^{(I-1)}, \tag{6.1.7}$$

$$L_{JI} = S_J^{(I-1)} D_{II}^{-1}, \tag{6.1.8}$$

onde $0 \leq I < J \leq N$.

Demonstração. Ver exercício 6.3. □

Proposição 10. *Seja A uma matriz em blocos $N \times N$ esparsa e não singular. O Algoritmo 16 quando aplicado a A, sem os descartes, atende à seguinte propriedade*

$$R_J^{(I-1)} = A_{IJ} - \sum_{K=1}^{I-1} S_I^{(K-1)} U_{KJ},$$

$$S_J^{(I-1)} = A_{JI} - \sum_{K=1}^{I-1} L_{JK} R_I^{(K-1)},$$

com $0 \leq I \leq J \leq N$.

Demonstração. A partir da fatoração LDU em blocos de A, então, para $0 \leq I \leq J \leq N$,

$$A_{JI} = \sum_{K=1}^{I} L_{JK} D_{KK} U_{KI} = L_{JI} D_{II} U_{II} + \sum_{K=1}^{I-1} L_{JK} D_{KK} U_{KI}.$$

Diante de (6.1.7), (6.1.8) e de que $U_{II} = I$,

$$\begin{array}{rcl} A_{JI} & = & S_J^{(I-1)} D_{II}^{-1} D_{II} + \sum_{K=1}^{I-1} S_J^{(K-1)} D_{KK}^{-1} D_{KK} D_{KK}^{-1} R_I^{(K-1)} \\ & = & S_J^{(I-1)} + \sum_{K=1}^{I-1} S_J^{(K-1)} D_{KK}^{-1} R_I^{(K-1)} \\ & \Rightarrow & S_J^{(I-1)} = A_{JI} - \sum_{K=1}^{I-1} L_{JK} R_I^{(K-1)} \end{array}.$$

Analogamente, a partir da fatoração LDU em blocos de A, com $0 \leq I \leq J \leq N$,

$$A_{IJ} = \sum_{K=1}^{I} L_{IK} D_{KK} U_{KJ} = L_{II} D_{II} U_{IJ} + \sum_{K=1}^{I-1} L_{IK} D_{KK} U_{KJ}.$$

De acordo com (6.1.7), (6.1.8) e de que $L_{II} = I$,

$$\begin{array}{rcl} A_{JI} & = & D_{II}^{-1} D_{II} R_J^{(I-1)} + \sum_{K=1}^{I-1} S_I^{(K-1)} D_{KK}^{-1} D_{KK} D_{KK}^{-1} R_J^{(K-1)} \\ & = & R_J^{(I-1)} + \sum_{K=1}^{I-1} S_I^{(K-1)} D_{KK}^{-1} R_J^{(K-1)} \\ & \Rightarrow & R_J^{(I-1)} = A_{IJ} - \sum_{K=1}^{I-1} S_I^{(K-1)} U_{KJ} \end{array}.$$

□

Mesmo que A seja esparsa, as matrizes Z e W produzidas pelos Algoritmos 19 e 20 podem ser densas e, para contornar este fato, são efetuados descartes de blocos ou entradas de Z e W. Ao se efetuar os descartes em Z e W é possível que as matrizes $A_{I*}Z_I$ ou $W_{I*}A_I$ sejam

singulares, mesmo quando $A_{1:J,1:J}$ é não singular para $J = 1, 2, \ldots, N$. Se A for positiva definida, uma alternativa livre de falha seria utilizar as expressões

$$(Z_I)^T A Z_I \tag{6.1.9}$$

ou

$$W_{I*} A (W_{I*})^T \tag{6.1.10}$$

para o cálculo de D_{II}. Isso se dá pois se A for positiva definida, então $(Z_I)^T A Z_I$ e $W_{I*} A (W_{I*})^T$ também serão positivas definidas e, portanto, não singulares. Esta é uma saída eficaz para se evitar a falha em matrizes positivas definidas.

6.2 Trabalhos desenvolvidos

Nesta seção, apresentamos os dois principais trabalhos encontrados na literatura sobre o AINV em blocos.

6.2.1 Inversa Aproximada em Blocos Não Simétrica (BAINV-NS)

Em [22], de 2000, Bridson e Tang apresentaram uma versão do SAINV-NS pela esquerda para quaisquer matrizes não singulares em blocos, chamada de Inversa Aproximada em Blocos Não Simétrica (*Nonsymetric Block Approximate Inverse* ou simplesmente BAINV-NS). Eles justificaram seu uso afirmando que esse tipo de abordagem pode reduzir os custos de operações em relação à versão escalar. Além disso, eles afirmaram que particionar A em pequenos blocos densos pode ser vantajoso em métodos diretos, reduzindo operações redundantes, como pode ser visto em [32].

O SAINV-NS (caso escalar) possui o SAINV como base, modificando o cálculo dos multiplicadores e do pivô. É possível adaptar essas modificações, levando em consideração as relações (6.1.4), (6.1.5) e $D_{JJ} = W_{J*} A Z_J$. Podemos ver esse processo no Algoritmo 21.

Determinar quando aplicar o descarte de blocos nas matrizes Z e W foi uma das principais questões abordadas, principalmente quando uma mesma tolerância é usada em blocos de diferentes tamanhos. Uma estratégia seria comparar a norma de Frobenius do bloco dividida pelo número de entradas dos blocos com uma tolerância τ. Caso o valor seja menor que τ, o descarte é aplicado. Eles utilizaram, então, esta estratégia de descarte com $\tau = 0, 1$.

Foram comparados o BAINV-NS e o SAINV-NS para os métodos CG para os casos SPD e BICGSTAB. Os precondicionadores gerados foram utilizados para os métodos CG, no caso SPD, e BIGSTAB, para

Algoritmo 21: BAINV-NS

Dados: matriz A em blocos $N \times N$ não singular.
Resultado: D, Z e W não singulares com $A^{-1} \approx ZD^{-1}W$.

$Z_1 \leftarrow E_1$;
$W_{1*} \leftarrow E_1^T$;
$D_{11} \leftarrow A_{11}$;
para $J \leftarrow 2$ **até** N **faça**
  $Z_J^{(0)} \leftarrow E_J$;
  $W_{J*}^{(0)} \leftarrow E_J^T$;
  para $I \leftarrow 1$ **até** $J-1$ **faça**
    $R_J^{(I-1)} \leftarrow W_{I*}A_J$;
    $S_J^{(I-1)} \leftarrow A_{J*}Z_I$;
    $Z_J^{(I)} \leftarrow descarte_I(Z_J^{(I-1)} - Z_I D_{II}^{-1} R_J^{(I-1)})$;
    $W_{J*}^{(I)} \leftarrow descarte_I(W_{J*}^{(I-1)} - S_J^{(I-1)} D_{II}^{-1} W_{I*})$;
  $Z_J \leftarrow Z_J^{(J-1)}$; $W_{J*} \leftarrow W_{J*}^{(J-1)}$
  $D_{JJ} \leftarrow W_{J*}AZ_J$;
$D \leftarrow \text{diag}(D_{11}, \ldots, D_{NN})$, $Z \leftarrow \left[\; Z_1, \quad \cdots \quad , Z_N \; \right]$ e

$$W \leftarrow \begin{bmatrix} W_{1*} \\ \vdots \\ W_{N*} \end{bmatrix};$$

os demais tipos de matrizes. O lado direito foi calculado usando o vetor solução com todas as entradas iguais a 1 e o critério de parada utilizado foi quando a norma dois do resíduo ficou menor que 10^{-6}. Em relação aos resultados, os autores afirmaram que esperavam que o algoritmo em blocos oferecesse ganhos em relação à eficiência de cache, mas a esses ganhos foram compensados pela sobrecarga da implementação. Eles também afirmaram que a versão em blocos melhorou a convergência de alguns problemas com estrutura em blocos mal condicionados, graças ao seu tratamento do pivô. Porém, para outros problemas, não foram obtidos bons resultados, já que a estratégia de descarte utilizada eliminava blocos inteiros, podendo impactar na qualidade do precondicionador. Na Tabela 6.1 exibimos um resumo das principais características deste trabalho.

Quadro Resumitivo	
Autores e Ano:	Bridson e Tang - 2000
Entrada:	Matriz esparsa A não simétrica e não singular e particionada em Blocos.
Saídas:	Z, W e D tais que $ZD^{-1}W^{\mathsf{T}} \approx A^{-1}$.
Robustez:	–
Na Literatura	
Método iterativo:	BICGSTAB
Descartes:	Tolerância (normalmente 0,1)
Escalamento:	Block Jacobi
Reordenamento:	MIP, MMD, GND, MC e QMD

Tabela 6.1: NS-SBAINV

6.2.2 Inversa Aproximada em Blocos Estabilizada (SBAINV)

Em 2001, Benzi, Kouhia e Tůma apresentaram o BAINV em [11], que é uma versão em blocos do AINV pela direita, para matrizes SPD. A partir desse, eles apresentaram a Inversa Aproximada em Blocos Estabilizada (*Stabilized Block Approximate Inverse* ou simplesmente SBAINV) como uma versão estável para matrizes SPD, baseando-se no SAINV, pois utiliza a relação $D_{II} = Z_I^T A Z_I$ (equação (6.1.9)) no cálculo dos pivôs. Os autores salientaram que a estrutura em blocos pode surgir tanto naturalmente, da discretização do problema real, quanto artificialmente, caso melhore o desempenho. O método proposto é exibido no Algoritmo 18, no Capítulo 5.

No BAINV, as inversas dos pivôs foram calculadas aplicando fatoração triangular. Por causa do descarte, os pivôs poderiam não ser SPD, e, para isso, era feita primeiramente uma fatoração LU do pivô

e, posteriormente, ele era calculado como LL^T, para deixá-lo simétrico. Se necessário, algumas modificações no pivô foram feitas para que ele se tornasse SPD. Com o SBAINV, eles não tiveram essa preocupação por ser garantidamente livre de falha para matrizes SPD.

Como mencionado, a estrutura em blocos de A pode vir naturalmente, mas nem sempre isso ocorre, podendo ser necessário reordenamentos de linhas e colunas em A para que tenha uma divisão em blocos "favorável". Os autores sugeriram, então, alguns tipos de reordenamentos e modificações na matriz A, dentre eles o uso do reescalamento Bloco Jacobi. O descarte utilizado foi de tolerância fixa ou tolerância relativa, podendo variar de acordo com algum aspecto do problema. Eles também utilizam padrão de zeros pré-definido "copiando"a esparsidade de A no precondicionador. Outro descarte feito foi a pós-filtragem, eliminando entradas de Z menores que um determinado valor após a execução do algoritmo.

Os autores compararam os precondicionadores BAINV, SBAINV, AINV, SAINV, Ichol e Ichol de Ajiz-Jennings em blocos [1] para o CG. O critério de parada utilizado foi quando a norma euclidiana do resíduo ficou menor que 10^{-5} ou 10^{-10}, dependendo do lado direito utilizado que variou podendo ser realístico ou artificial. Quanto aos resultados, os autores concluíram que o desempenho da versão em blocos foi melhor para problemas de casca fina, onde os blocos possuem geralmente tamanho igual a seis, do que em relação à versão escalar da inversa aproximada. Já em relação aos problemas de mecânica dos sólidos, cujo tamanho dos blocos foram iguais a três, o SBAINV, apesar de confiável, obteve resultados inferiores ao SAINV-NS. Na Tabela 6.2 exibimos um resumo das principais características deste trabalho.

Quadro Resumitivo	
Autores e Ano:	Benzi, Kouhia e Tüma - 2001
Entrada:	Matriz esparsa A SPD e particionada em Blocos.
Saídas:	Z e D tais que $ZD^{-1}Z^{\mathsf{T}} \approx A^{-1}$.
Robustez:	Matrizes SPD
Na Literatura	
Método iterativo:	CG
Descartes:	• AINV(0) - copia em Z a esparsidade de A. • Tolerância • Descarte Relativo (cada entrada de Z é descartada caso seja menor que uma tolerância multiplicada pela norma infinito da atual linha de A). • Forma pré-definida de zeros. • Filtragem posterior (depois que Z é calculado, usa-se outra tolerância, para descartar mais entradas de Z).
Escalamento:	Block Jacobi
Reordenamento:	MMD

Tabela 6.2: BAINV

6.3 Variações propostas

Nesta seção, apresentamos algumas adaptações para a inversa aproximada em blocos, algumas delas baseadas nas versões escalares do AINV exibidas no Capítulo 3.

6.3.1 Inversa Aproximada em Blocos Estabilizada Não Simétrica (SBAINV-NS)

Propomos uma versão não simétrica para o BAINV proposto por Benzi e Tůma em [11]. Temos como base o método de A-biconjugação em blocos, no Algoritmo 19, e incluímos a possibilidade de calcular o pivô como nas equações (6.1.9) e (6.1.10), se A for positiva definida. Por isso chamamos o método de Inversa Aproximada em Blocos Estabilizada

Não Simétrica ou simplesmente SBAINV-NS, exibido nos Algoritmos 22 e 23, nas versões pela direita e pela esquerda, respectivamente.

Algoritmo 22: SBAINV-NS pela direita

Dados: matriz A em blocos $N \times N$ não singular.
Resultado: D, Z e W não singulares com $A^{-1} \approx ZD^{-1}W$.

$Z_I^{(0)} \leftarrow E_I, \qquad I \in \{1, ..., N\}$;
$W_{I*}^{(0)} \leftarrow E_I^T, \qquad I \in \{1, ..., N\}$;
para $I \leftarrow 1$ **até** N **faça**
  $Z_I \leftarrow Z_I^{(I-1)}$;
  $W_{I*} \leftarrow W_{I*}^{(I-1)}$;
  $D_{II} \leftarrow A_{I*}Z_I$ ou $W_{I*}A_I$;
  $D_{II} \leftarrow (Z_I)^T AZ_I$ ou $W_{I*}A(W_{I*})^T$ se A for positiva definida;
  para $J \leftarrow I+1$ **até** N **faça**
    $M_J^{(I-1)} \leftarrow A_{I*}Z_J^{(I-1)}$;
    $S_J^{(I-1)} \leftarrow W_{J*}^{(I-1)}A_I$;
    $Z_J^{(I)} \leftarrow descarte_I(Z_J^{(I-1)} - Z_I D_{II}^{-1} M_J^{(I-1)})$;
    $W_{J*}^{(I)} \leftarrow descarte_I(W_{J*}^{(I-1)} - S_J^{(I-1)} D_{II}^{-1} W_{I*})$;
$D \leftarrow \text{diag}(D_{11}, \ldots, D_{NN})$, $Z \leftarrow \left[\; Z_1, \quad \cdots \quad , Z_N \; \right]$ e
$W \leftarrow \begin{bmatrix} W_{1*} \\ \vdots \\ W_{N*} \end{bmatrix}$;

Lema 6.3. *Caso o Algoritmo 22 não falhe, ele gera uma matriz bloco triangular inferior Z e uma matriz bloco triangular superior W. Além disso, os blocos diagonais de Z e W são formados pela identidade.*

Demonstração. A prova é análoga a do Lema 6.1. As únicas observações adicionais são de que $E_I^T descarte_I(V) = E_I^T V$ e que

$$E_L^T V = 0 \;\Rightarrow\; E_L^T descarte_I(V) = 0,$$

assim como $descarte_I(V)E_I = VE_I$ e que

$$VE_I = 0 \;\Rightarrow\; descarte_I(V)E_I = 0).$$

□

Desta forma, sendo A uma matriz em blocos não singular, se o Algoritmo 22 não falhar, então D é bloco diagonal não singular e Z e W são bloco triangulares inferior e superior, respectivamente, com blocos diagonais formados pela identidade e, portanto, também não singulares.

Algoritmo 23: SBAINV-NS pela esquerda

Dados: matriz A em blocos $N \times N$ não singular.
Resultado: D, Z e W não singulares com $A^{-1} \approx ZD^{-1}W$.

$Z_1 \leftarrow E_1$;
$W_{1*} \leftarrow E_1^T$;
$D_{11} \leftarrow A_{11}$;
para $J \leftarrow 2$ **até** N **faça**
  $Z_J^{(0)} \leftarrow E_J$; $W_{J*}^{(0)} \leftarrow E_J^T$;
  para $I \leftarrow 1$ **até** $J-1$ **faça**
    $M_J^{(I-1)} \leftarrow A_{I*} Z_J^{(I-1)}$;
    $S_J^{(I-1)} \leftarrow W_{J*}^{(I-1)} A_I$;
    $Z_J^{(I)} \leftarrow descarte_I(Z_J^{(I-1)} - Z_I D_{II}^{-1} M_J^{(I-1)})$;
    $W_{J*}^{(I)} \leftarrow descarte_I(W_{J*}^{(I-1)} - S_J^{(I-1)} D_{II}^{-1} W_{I*})$;
  $Z_J \leftarrow Z_J^{(J-1)}$; $W_{J*} \leftarrow W_{J*}^{(J-1)}$;
  $D_{JJ} \leftarrow A_{J*} Z_J$ ou $W_{J*} A_J$;
  $D_{JJ} \leftarrow (Z_J)^T A Z_J$ ou $W_{J*} A (W_{J*})^T$ se A for positiva definida;
$D \leftarrow diag(D_{11}, \ldots, D_{NN})$, $Z \leftarrow \left[\; Z_1, \quad \cdots \quad , Z_N \; \right]$ e

$$W \leftarrow \begin{bmatrix} W_{1*} \\ \vdots \\ W_{N*} \end{bmatrix};$$

6.3.2 Inversa Aproximada Fatorada pela Frente em Blocos (BFFAPINV)

Nesta seção, apresentamos a versão em blocos do algoritmo FFAPINV [69]. Utilizando as relações (6.1.4) e (6.1.5) propomos a Inversa Aproximada Fatorada pela Frente em Blocos, ou simplesmente BFFAPINV, que, de forma análoga ao caso escalar, utiliza os elementos de W para calcular $M_J^{(I-1)}$ e os elementos de Z para calcular $S_J^{(I-1)}$. Este processo pode ser visto no Algoritmo 24. Vejamos que, como no caso escalar, ele é exibido baseando-se na versão pela esquerda do algoritmo de A-biconjugação.

Algoritmo 24: BFFAPINV

Dados: matriz A em blocos $N \times N$ não singular.
Resultado: D, Z e W não singulares com $A^{-1} \approx ZD^{-1}W$.

$Z_1 \leftarrow E_1$;
$W_{1*} \leftarrow E_1^T$;
$D_{11} \leftarrow A_{11}$;
para $J \leftarrow 2$ **até** N **faça**
  $Z_J^{(0)} \leftarrow E_J$;
  $W_{J*}^{(0)} \leftarrow E_J^T$;
  para $I \leftarrow 1$ **até** $J-1$ **faça**
    $M_J^{(I-1)} \leftarrow W_{I*}A_J$;
    $S_J^{(I-1)} \leftarrow A_{J*}Z_I$;
    $Z_J^{(I)} \leftarrow descarte_I(Z_J^{(I-1)} - Z_I D_{II}^{-1} M_J^{(I-1)})$;
    $W_{J*}^{(I)} \leftarrow descarte_I(W_{J*}^{(I-1)} - S_J^{(I-1)} D_{II}^{-1} W_{I*})$;
  $Z_J \leftarrow Z_J^{(J-1)}$; $W_{J*} \leftarrow W_{J*}^{(J-1)}$;
  $D_{JJ} \leftarrow W_{J*}AZ_J$;
$D \leftarrow diag(D_{11}, \ldots, D_{NN})$, $Z \leftarrow \left[\; Z_1, \quad \cdots \quad , Z_N \; \right]$ e

$$W \leftarrow \begin{bmatrix} W_{1*} \\ \vdots \\ W_{N*} \end{bmatrix};$$

6.3.3 Inversa Aproximada Estabilizada Variada em Blocos (SBAINV-VAR)

Nesta seção, apresentamos a versão em blocos do algoritmo SAINV-VAR [65], a algoritmo da Inversa Aproximada Estabilizada Variada em Blocos, ou simplesmente SBAINV-VAR, descrito no Algoritmo 25. Baseado no SAINV-VAR, são produzidas aproximações dos fatores Z, L e D de A, sendo possível chegar na aproximação de L por meio

da equação (6.1.8). Por sua vez, para se calcular $S_J^{(I-1)}$ é utilizada a Proposição 10 e a equação (6.1.8), ou seja, seu cálculo é executado sem a utilização do fator W. Para o cálculo de D_{II}, é dada a opção de se utilizar a expressão $Z_I^T A Z_I$ que evita a falha em matrizes positivas definidas.

Algoritmo 25: SBAINV-VAR

Dados: matriz A em blocos $N \times N$ não singular.
Resultado: matriz bloco diagonal D não singular e matrizes Z e L não singulares.

$Z_I^{(0)} \leftarrow E_I, \qquad I \in \{1, ..., N\}$;
para $I \leftarrow 1$ **até** N **faça**
  $Z_I \leftarrow Z_I^{(I-1)}$;
  $D_{II} \leftarrow A_{I*} Z_I$ **ou** $Z_I^T A Z_I$ se A for positiva definida ;
  para $J \leftarrow I+1$ **até** N **faça**
    $M_J^{(I-1)} \leftarrow A_{I*} Z_J^{(I-1)}$;
    $S_J^{(I-1)} \leftarrow A_{JI} - \sum_{K=1}^{I-1} L_{JK} M_I^{(K-1)}$;
    $L_{JI} \leftarrow \text{descarte}(S_J^{(I-1)} D_{II}^{-1})$;
    $Z_J^{(I)} \leftarrow \text{descarte}_I(Z_J^{(I-1)} - Z_I D_{II}^{-1} M_J^{(I-1)})$;
$D \leftarrow diag(D_{11}, \ldots, D_{NN})$, $Z \leftarrow \left[\begin{array}{ccc} Z_1, & \cdots & , Z_N \end{array} \right]$ e $L \leftarrow [L_{JI}]$;

No SBAINV-VAR, além da utilização do $descarte_I$, também utilizamos o descarte para promover a esparsidade dos blocos L_{JI}, definido a seguir.

Definição 6.1. *Seja a matriz M de ordem t, a notação* descarte *indica a matriz resultante do procedimento de descartar determinadas entradas de M, i.e.,* $(\text{descarte}(M))_{ij} = m_{ij}$ *ou* $(\text{descarte}(M))_{ij} = 0$.

Para obter a aproximação de W, fazemos a aproximação da inversa de L ($W = L^{-1}$) por meio do trucamento da série de Neumann de grau l (veja [55]):

$$W_l = I + F + F^2 + F^3 + ... + F^l, \tag{6.3.11}$$

onde $F = I - L$ e I é a identidade. Por meio de (6.3.11) e do método de Horner [43], obtemos a inversa aproximada $A^{-1} \approx Z D^{-1} W_l$. A ideia é utilizar a fórmula de Horner [43] para multiplicar W_l pelo resíduo do método iterativo, fazendo uso da multiplicação matriz-vetor. Ou seja, a aplicação do precondicionador gerado pelo SBAINV-VAR se dá por meio do método de Horner e da multiplicação das matrizes Z e D^{-1} pelo resíduo em cada iteração do método iterativo. Este processo pode ser visto no Algoritmo 26.

Algoritmo 26: Aplicação do precondicionador gerado pelo SBAINV-VAR

Dados: matrizes D, Z e L não singulares, vetor resíduo r e inteiro l (grau do polinômio Neumann).

Resultado: vetor v.

$r_0 \leftarrow r$;
para $i \leftarrow 1$ **até** l **faça**
  $r_l \leftarrow r_{l-1} - Lr_{l-1}$;
$y \leftarrow \sum_{i=0}^{l} r_l$;
$v \leftarrow ZD^{-1}y$;

Comentário 5. *Notemos que o SAINV-VAR produz as matrizes W, D e U e calcula Z através da aproximação de U^{-1}, a partir do somatório de Neumann. Já o SBAINV-VAR produz as matrizes Z, D e L e calcula W através da aproximação de L^{-1}, a partir somatório de Neumann. Vejamos que, apesar de produzir matrizes diferentes, o SBAINV-VAR tem a mesma lógica de raciocínio que o SAINV-VAR. Escolhemos produzir fatores diferentes na versão blocos por questões de implementação.*

6.3.4 Fatoração Incompleta Robusta Não Simétrica em Blocos (BRIF-NS)

Nesta seção apresentamos a versão em blocos do algoritmo RIF-NS [64], chamada de Fatoração Incompleta Robusta Não Simétrica em Blocos, ou simplesmente BRIF-NS. A ideia principal do BRIF-NS é encontrar a fatoração bloco LDU aproximada de A através do processo de biconjugação em blocos de A e utilizando a equação 6.1.6, caso A for positiva definida. As entradas das aproximações de L e U são obtidas através das equações (6.1.8) e (6.1.7) e as matrizes R_J's calculadas com base na Proposição 10. No Algoritmo 27, vemos a implementação do BRIF-NS.

Desse modo, propomos uma versão em blocos do Algoritmo RIF-NS, o qual havia sido introduzido em [64]. Mais amplamente, ao longo deste capítulo, realizamos as versões bloco para três algoritmos, a saber, o FFAPINV, o SAINV-NS e RIF, todos estruturados para matrizes não simétricas. Além disso, apresentamos uma revisão sistemática dos trabalhos desenvolvidos no âmbito dos algoritmos de inversa aproximada em blocos.

Algoritmo 27: BRIF-NS pela direita

Dados: matriz A em blocos $N \times N$ não singular.
Resultado: a nonsingular block-diagonal matrix D, and nonsingular block-matrices L and U such that $A \approx LDU$.

1 $W_{I*}^{(0)} \leftarrow E_I^T, \qquad I \in \{1, ..., N\}$
2 **para** $I \leftarrow 1$ **até** N **faça**
3 $\quad W_{I*} \leftarrow W_{I*}^{(I-1)}$;
4 $\quad D_{II} \leftarrow W_{I*}^{(I-1)} A_I$;
5 $\quad D_{II} \leftarrow W_{I*}^{(I-1)} A (W_{I*}^{(I-1)})^T$(se for positiva definida);
6 $\quad$ **para** $J \leftarrow I + 1$ **até** N **faça**
7 $\quad\quad S_J^{(I-1)} \leftarrow W_{J*}^{(I-1)} A_I$;
8 $\quad\quad L_{JI} \leftarrow descarte(S_J^{(I-1)} D_{II}^{-1})$;
9 $\quad\quad M_J^{(I-1)} \leftarrow A_{IJ} + \sum_{K=1}^{I-1} S_I^{(K-1)} U_{KJ}$;
10 $\quad\quad U_{IJ} \leftarrow descarte(D_{II}^{-1} M_J^{(I-1)})$
11 $\quad\quad W_{J*}^{(I)} \leftarrow descarte_I(W_{J*}^{(I-1)} - S_J^{(I-1)} D_{II}^{-1} W_{I*})$;
12 $D \leftarrow diag(D_{11}, \ldots, D_{NN})$, $L \leftarrow [L_{JI}]$ e $U \leftarrow [U_{IJ}]$;

Exercício 6.1. *Considere a matriz A tal que:*

$$A = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & -2 & -1 & 1 \\ 2 & 0 & -1 & 0 \\ 0 & 1 & 0 & 2 \end{bmatrix}$$

Mostre que A é uma matriz não singular. Mostre que o Algoritmo de Biconjugação não pode ser aplicado a matriz A. Mostre que o Algoritmo de Biconjugação em Blocos (para o tamanho de bloco igual a 2) produz uma fatoração da inversa de A.

Exercício 6.2. *Demonstre a Proposição 8.*

Exercício 6.3. *Demonstre a Proposição 6.1.7.*

Bibliografia

[1] AJIZ, M. A.; JENNINGS, A. A robust incomplete cholesky conjugate gradient algorithm. *International Journal for Numerical Methods in Engineering*, v. 20, p. 949–966, 1984.

[2] ALMEIDA, M. C.; CRUZ, J. S.; GOLDFELD, P.; CARVALHO, L. M.; SOUZA, M. Supporting theory for a block approximate inverse preconditioner. *Linear Algebra and its Applications*, v. 614, p. 325–342, 2020.

[3] ALMEIDA, M. C. de. *Precondicionador de inversa aproximada por blocos*. Tese (Doutorado) — PPGEM, UERJ, Rio de Janeiro, 2019.

[4] AMESTOY, P. R.; DAVIS, T. A.; DUFF, I. S. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, v. 17, n. 4, p. 886–905, 1996.

[5] ARBENZ, P.; CHINELLATO, O.; SALA, M.; ARBENZ, P.; GUTKNECHT, M. H. *Software for numerical linear algebra*. ETH, 2006. Disponível em: <http://e-collection.ethbib.ethz.ch/view/eth:28724>.

[6] AXELSSON, O. *Iterative Solution Methods*. Cambridge: Cambridge University Press, 1994.

[7] AXELSSON, O.; KOLOTILINA, L. Y. Diagonally compensated reduction and related preconditioning methods. *Numerical Linear Algebra With Applications*, v. 1, p. 155–177, 1994.

[8] BARRETT, R.; BERRY, M.; CHAN, T.; DEMMEL, J.; DONATO, J.; DONGARRA, J.; EIJKHOUT, V.; POZO, R.; ROMINE, C.; VORST, V. D. H. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Philadelphia: Society for Insdustrial and Applied Mathematics, 1994.

[9] BENZI, M. Preconditioning Techniques for Large Linear Systems: A Survey. *Journal of Computational Physics*, v. 182, n. 2, p. 418–477, November 2002.

[10] BENZI, M.; CULLUM, J.; TUMA, M. Robust approximate inverse preconditioning for the conjugate gradient method. *SIAM Journal on Scientific Computing*, v. 22, n. 4, p. 1318–1332, 2000.

[11] BENZI, M.; KOUHIA, R.; TUMA, M. Stabilized and block approximate inverse preconditioners for problems in solid and structural mechanics. *Computer Methods in Applied Mechanics and Engineering*, v. 190, p. 6533–6554, 2001.

[12] BENZI, M.; MEYER, C. D.; TUMA, M. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM Journal on Scientific Computing*, v. 17, n. 55, p. 1135–1149, 1996.

[13] BENZI, M.; TUMA, M. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, v. 19, p. 968–994, 1998.

[14] BENZI, M.; TUMA, M. A comparative study of sparse approximate inverse preconditioners. *Applied Numerical Mathematics*, v. 30, p. 305–340, 1999.

[15] BENZI, M.; TUMA, M. A robust preconditioner with low memory requirements for large sparse least squares problems. *SIAM Journal on Scientific Computing*, v. 25, n. 2, p. 499–512, 2003.

[16] BERMAN, A.; KIM, K.; PLEMMONS, R. J. *Nonnegative Matrices in the Mathematical Sciences*. New York: Academic Press, 1979.

[17] BERMAN, A.; PLEMMONS, R. J. *Nonnegative Matrices in the Mathematical Sciences*. Philadelphia: Society for Industrial and Applied Mathematics, 1994.

[18] BOLLHÖFER, M.; SAAD, Y. On the relations between ilus and factored approximate inverses. *SIAM Journal on Matrix Analysis and Applications*, v. 24, n. 1, p. 219–237, 2002.

[19] BOLLHÖFFER, M.; SAAD, Y. A factored approximate inverse preconditioner with pivoting. *SIAM Journal on Matrix Analysis and Applications*, v. 253, p. 692–705, 2002.

[20] BREZINA, M.; CLEARY, A. J.; FALGOUT, R. D.; HENSON, V. E.; JONES, J. E.; MANTEUFFEL, T. A.; MCCORMICK, S. F.; RUGE, J. W. Algebraic multigrid based on element interpolation (AMGe). *SIAM Journal on Scientific Computing*, SIAM, v. 22, n. 5, p. 1570–1592, 2001. Disponível em: <http://link.aip.org/link/?SCE/22/1570/1>.

[21] BRIDSON, R.; TANG, W. Ordering, anisotropy, and factored sparse approximate inverses. *SIAM Journal on Scientific Computing*, v. 21, p. 867–882, 1999.

[22] BRIDSON, R.; TANG, W. Refining an approximate inverse. *Journal of Computational and Applied Mathematics*, v. 123, p. 293–306, 2000.

[23] BRU, R.; CORRAL, C.; I, G.; MAS, J. Classes of general h-matrices. *Linear Algebra and Its Applications*, v. 429, p. 2358–2366, 2008.

[24] CAO, Z.; LIU, Z. Symmetric multisplitting of a symmetric positive definite matrix. *Linear Algebra and Its Applications*, v. 37, n. 1, p. 309–319, 1998.

[25] CARVALHO, L. M. *Preconditioned Schur complement methods in distributed memory environments*. Tese (Doutorado) — INPT/CERFACS, France, out. 1997. TH/PA/97/41, CERFACS.

[26] CARVALHO, L. M.; GIRAUD, L.; MEURANT, G. Local preconditioners for two-level nonoverlapping domain decomposition methods. *Numerical Linear Algebra with Applications*, v. 8, n. 4, p. 207 – 227, 2001.

[27] CARVALHO, L. M.; GRATTON, S. *Avanços em Métodos de Krylov para Solução de Sistemas Lineares de Grande Porte*. São Carlos: SBMAC, 2012. (Notas em Matemática Aplicada, v. 42). E-book.

[28] CHAN, T. F.; VORST, H. A. Approximate and incomplete factorizations. In: _____. *Parallel numerical algorithms*. [S.l.]: Springer, 1997. p. 167–202.

[29] CHEN, K. *Matrix Preconditioning Techniques and Applications*. [S.l.]: Cambridge University Press, 2005.

[30] CHOW, E.; SAAD, Y. Approximate inverse techniques for block-partitioned matrices. *SIAM Journal on Scientific Computing*, SIAM, v. 18, n. 6, p. 1657–1675, 1997. Disponível em: <http://link.aip.org/link/?SCE/18/1657/1>.

[31] CHOW, E.; SAAD, Y. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM Journal on Scientific Computing*, v. 19, n. 3, p. 995–1023, maio 1998.

[32] ESMOND, J. W. H. L. G. N.; PEYTON, P. W. On finding supernodes for sparse matrix computations. *SIAM Journal on Matrix Analysis and Applications*, v. 14, n. 1, p. 242–252, 1993.

[33] FORSYTH, J. P. A.; SAMMONP, H. Practical considerations for adaptive implicit methods in reservoir simulation. *Journal of Computational Physics*, v. 62, p. 265–281, 1986.

[34] FREUND, R. W. A transpose-free quasi-minimal residual algorithm for non-hermitian linear systems. *SIAM Journal on Scientific Computing*, v. 14, n. 2, p. 470–482, 1993.

[35] FROMMER, A.; MAASS, P. Fast CG-based methods for Tikhonov–Phillips regularization. *SIAM Journal on Scientific Computing*, v. 20, p. 1831–1850, 1999.

[36] FUJINO, S.; IKEDA, Y. An improvement of SAINV and RIF precondtionings of CG method by double dropping strategy. *IPSJ*, v. 45, n. 1, p. 10–17, 2004.

[37] GEORGE, A.; LIU, J. *Computer solution of large sparse positive definite systems*. Englewood Cliffs: Prentice-Hall, 1981. (Computational Mathematics).

[38] GOLUB, G. H.; LOAN, C. F. V. *Matrix Computations*. 4rd. ed. [S.l.]: Johns Hopkins University Press, 2013.

[39] GOULD, N.; SCOTT, J. *On approximate-inverse preconditioners*. Chilton, 1995. Disponível em: <citeseer.ist.psu.edu/gould95approximateinverse.html>.

[40] GROTE, M. J.; HUCKLE, T. Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, SIAM, v. 18, n. 3, p. 838–853, 1997.

[41] HACKBUSCH, W. *Iterative Solution of Large Sparse Linear Systems of Equations*. [S.l.]: Springer, 1994.

[42] HESTENES, M.; STIEFEL, E. Methods of conjugate gradients for solving linear systems. *National Bureau of Standards*, v. 49, p. 409–436, 1952.

[43] HIGHAM, J. N. *Accuracy and stability of numerical algorithms*. Manchester: Society for Insdustrial and Applied Mathematics, 2002.

[44] HORN, R. A.; JOHNSON, C. R. *Matrix Analysis*. 2. ed. New York: Cambridge University Press, 2013.

[45] KARYPIS, G.; KUMAR, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, v. 20, p. 359–392, 1998.

[46] KARYPIS, G.; KUMAR, V. *METIS, a Software Package for Partitioning Unstructured Graphs and Computing Fill-Reduced Orderings of Sparse Matrices*. Minnesota: University of Minnesota, 1998.

[47] KOLOTILINA, L. Y.; YEREMIN, A. Y. Factorized sparse approximate inverse preconditionings I. theory. *SIAM Journal on Matrix Analysis and Applications*, v. 14, n. 1, p. 45–58, 1993.

[48] KOLOTILINA, Y. Two-sided bounds for the inverse of an h-matrix. *Linear Algebra and Its Applications*, v. 225, p. 114–123, 1995.

[49] KRYLOV, A. N. On the numerical solution of the equation by which in technical questions frequencies of small oscillations of material systems are determined. *News of Academy of Sciences of the USSR*, VII, n. 4, p. 491–539, 1931. (in Russian).

[50] LEE, E. J.; ZHANG, J. Factored approximate inverse preonditioners with dynamic sparsity patterns. *Computers & Mathematics with Applications*, v. 62, p. 235–242, 2011.

[51] LIU, J. W. H. Modification of the minimum-degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software*, v. 11, n. 1, p. 141–153, 1985.

[52] MANTEUFFEL, T. A. An incomplete factorization technique for positive definite linear system. *Mathematics of Computation*, v. 34, p. 473–4979, 1980.

[53] MATHEW, T. P. A. *Domain Decomposition Methods for the Numerical Solution of Partial Differential Equations*. [S.l.]: Springer, 2008.

[54] MATRIX Market. `https://math.nist.gov/MatrixMarket/index.html`. Accessed: 2023-02-14.

[55] MEURANT, G. *Computer solution of large linear systems*. North Holland: Elsevier, 1999.

[56] MEURANT, G.; TEBBENS, J. D. *Krylov Methods for Nonsymmetric Linear Systems*. [S.l.]: Springer, 2020.

[57] MEYER, C. D. *Matrix analysis and applied liner algebra*. [S.l.]: SIAM, 2000.

[58] MáLEK, J.; STRAKOS, Z. *Preconditioning and the Conjugate Gradient Method in the Context of Solving PDEs*. [S.l.]: SIAM, 2015.

[59] OSTROWSKI, A. Uber die determinanten mit uberwiegender hauptdiagonal. *Commentarii Mathematici Helvetici*, v. 10, p. 69–96, 1937.

[60] PAPADOULOS, A. T.; DUFF, I. S.; WATHEN, A. J. Incomplete orthogonal factorization methods using givens rotations ii: Implementation and results. *Oxford University Computing Laboratory*, Report, p. 175–188, 2002.

[61] PLEMMONS, R. J. M-matrix characterizations. I—nonsingular M-matrices. *Linear Algebra and Its Applications*, v. 18, p. 175–188, 1977.

[62] RAFIEI, A. A complete pivoting strategy for the right-looking Robust Incomplete Factorization preconditioner. *Computers and Mathematics with Applications*, v. 64, p. 2682–2694, 2012.

[63] RAFIEI, A. Left-looking version of AINV preconditioner with complete pivoting strategy. *Linear Algebra and its Applications*, v. 445, p. 103–126, 2014.

[64] RAFIEI, A.; BOLLHÖFER, M. Robust incomplete factorization for nonsymmetric matrices. *Numerische Mathematik*, v. 118, p. 247–269, 2011.

[65] RAFIEI, A.; TOUTOUNIAN, F. New breakdown-free variant of AINV method for nonsymmetric positive definite matrices. *Journal of Computational and Applied Mathematics*, v. 219, p. 72–80, 2008.

[66] RUGE, J. W.; STÜBEN, K. Algebraic multigrid (AMG). In: MCCORMICK, S. F. (Ed.). *Multigrid Methods*. Philadelphia, PA: SIAM, 1987, (Frontiers in Applied Mathematics, v. 3). p. 73–130.

[67] SAAD, Y. *Iterative Methods for Sparse Linear Systems*. 2. ed. Minnesota: Society for Insdustrial and Applied Mathematics, 2003.

[68] SAAD, Y.; ZHANG, J. Block versions of multielimination and multilevel ILU preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, v. 20, n. 6, p. 2103–2121, 1999.

[69] SALKUYEH, D. K. A sparse approximate inverse preconditioner for nonsymmetric positive definite matrices. *J. Appl. Math. & Informatics*, v. 28, n. 5-6, p. 1131–1141, 2010.

[70] SALKUYEH, D. K.; ROOHANI, H. On the relation between the AINV and the FAPINV algorithms. *International Journal of Mathematics and Mathematical Sciences*, v. 2009, p. 1131–1141, 2009.

[71] SCHNABEL, R. B.; ESKOW, E. A new modified cholesky factorization. *National Bureau of Standards*, v. 11, p. 1136–1158, 1981.

[72] SCHWARZ, H. A. *Ueber einen Grenzübergang durch alternirendes Verfahren*. [S.l.]: Zürcher u. Furrer, 1870.

[73] SHEWCHUK, J. R. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain Edition* $1\frac{1}{4}$. [S.l.], August 1994.

[74] SMITH, B.; BJØRSTAD, P.; GROPP, W. *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*. 1st. ed. New York: Cambridge University Press, 1996.

[75] THOMAS, G. W.; THURNAU, D. H. Reservoir simulation using an adaptive implicit method. *SPE Journal*, v. 23, p. 760–768, 1983.

[76] TOSELLI, A.; WIDLUND, O. *Domain Decomposition methods - Algorithms and Theory*. [S.l.]: Springer, 2004. (Computational Mathematics, v. 34).

[77] TREFETHEN, L.; III, D. B. *Numerical linear algebra*. Philadelphia: SIAM press, 1997.

[78] TROTTENBERG, U.; OOSTERLEE, C.; SCHULLER, A. *Multigrid*. [S.l.]: Academic Press, 2001.

[79] WESSELING, P. *An Introduction to Multigrid Methods*. New York: Wiley, 1992.

[80] WESSELING, P.; OOSTERLEE, C. Geometric multigrid with applications to computational fluid dynamics. *Journal of Computational and Applied Mathematics*, v. 128, n. 1-2, p. 311–334, 2001.

[81] ZHANG, J. Sparse approximate inverse and multilevel block ILU preconditioning techniques for general sparse matrices. *Applied Numerical Mathematics*, v. 35, n. 1, p. 67–86, set. 2000. Disponível em: <http://www.sciencedirect.com/science/article/B6TYD-413KWYV-D/2/74cafe7b9754d45b373531e9c8ad1b24>.

[82] ZOU, Q. GMRES algorithms over 35 years. *arXiv preprint arXiv:2110.04017*, 2021.

Índice