Universidade do Estado do Rio de Janeiro

Centro de Tecnologia e Ciências

Faculdade de Engenharia

Julia Sekiguchi da Cruz

Variações da Inversa Aproximada e suas versões em Blocos

Rio de Janeiro

2021

Julia Sekiguchi da Cruz

Variações da Inversa Aproximada e suas versões em Blocos

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

Orientador: Prof. Dr. Luiz Mariano Carvalho

Prof. Dr. Michael Ferreira de Souza

Rio de Janeiro

2021

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

C957 Cruz, Julia Sekiguchi da.
Variações da inversa aproximada e suas versões em blocos / Julia Sekiguchi da Cruz. – 2021.
126f.

Orientadores: Luiz Mariano Paes de Carvalho Filho, Michael Ferreira de Souza.
Tese (Doutorado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.

1. Engenharia mecânica - Teses. 2. Sistemas lineares - Teses. 3. Matriz inversa - Teses. 4. Algorítmos - Teses. I. Carvalho Filho, Luiz Mariano Paes de. II. Souza, Michael Ferreira de. III. Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia. IV. Título.

CDU 512.643

Bibliotecária: Júlia Vieira – CRB7/6022

Assinatura

Data

Julia Sekiguchi da Cruz

Variações da Inversa Aproximada e suas versões em Blocos

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

Aprovada em: 28 de Setembro de 2021

Banca Examinadora:

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

Prof. Dr. Michael Ferreira de Souza (Orientador)
Departamento de Estatística e Matemática Aplicada da UFC

Prof. Dr. Moises Ceni de Almeida
Departamento de Matemática do Instituto Federal do Rio de Janeiro

---

Prof. Dr. Norberto Mangiavachi
Faculdade de Engenharia da UERJ

Prof. Dr. Nelson Maculan
COPPE - UFRJ

Rio de Janeiro

2021

AGRADECIMENTO

Agradeço a Deus, por tudo que me é possibilitado.

À minha família, em especial aos meus pais Thereza e Marcos, que estão sempre me apoiando e acreditando em mim.

Ao meu irmão Vitor, pela ajuda com programação.

Aos meus amigos, por me darem força e me animarem.

Aos meus orientadores e amigos professores Dr. Luiz Mariano Carvalho e Dr. Michael Souza, pelos ensinamentos e companheirismo.

Ao meu colega de trabalho e amigo professor Dr. Moisés Ceni de Almeida, por ter acreditado em mim e me apresentado este ramo de pesquisa.

À UERJ, por todo o suporte e acolhimento de sempre.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

RESUMO

CRUZ, Julia Sekiguchi da. *Variações da Inversa Aproximada e suas versões em Blocos.* 95 f. Tese (Doutorado em Engenharia Mecânica) - Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro (UERJ), Rio de Janeiro, 2021.

Neste trabalho, analisamos as principais características do precondicionador Inversa Aproximada (AINV), proposto por Benzi, Meyer e Tůma em 1996, e de suas variações. Nós também demonstramos alguns resultados referentes à Inversa Aproximada em Blocos (BAINV), proposto por Benzi e Tůma em 2001, e também referentes a algumas variações do BAINV, que propomos neste trabalho. Por fim, nós apresentamos os resultados numéricos obtidos usando duas destas variações.

Palavras-chave: Sistema Linear; Precondicionador; Inversa Aproximada; Matrizes em bloco.

ABSTRACT

CRUZ, Julia Sekiguchi da. *Variations of the Approximate Inverse and its Block versions.* 95 f. Thesis (PhD in Mechanical Engineering) - Engineering Faculty, Rio de Janeiro State University (UERJ), Rio de Janeiro, 2021.

In this work, we study the main characteristics of the Approximate Inverse Preconditioning (AINV), proposed by Benzi, Meyer and Tůma in 1996, and its variations. We also prove some results regarding the Approximate Inverse in Blocks (BAINV), proposed by Benzi and Truma in 2001, and also referring to some variations of BAINV, which we propose in this work. Finally, we present the numerical results obtained using two of these variations.

Keywords: Linear System; Preconditioner; Approximate Inverse; Block Matrices.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

INTRODUÇÃO

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 larga escala, o uso de métodos diretos para a sua resolução, como a eliminação gaussiana, é impraticável, pois a complexidade é $\mathcal{O}(n^3)$, fazendo-se necessário o uso de métodos iterativos. Dentre eles, temos os métodos de projeção em subespaços de Krylov [37,64]. Para matrizes simétricas positivas definidas (SPD) a escolha usual é o Método de Gradientes Conjugados (CG) [39]. Porém, métodos de Krylov podem convergir lentamente sem um precondicionador adequado, [37].

A ideia do precondicionador é construir um operador M tal que $MAx = Mb$ ou $AMy = b$ ($x = My$) 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. Exitem diversos tipos de precondicionadores e um exemplo bem conhecido é o precondicionador ILU [54], que é uma aproximação de A, baseado na fatoração LU incompleta de A. Neste caso, quando o método de Krylov é aplicado ao sistema precondicionado, são resolvidos dois sistemas triangulares. Ele possui dezenas de variações [63] e, por ser uma aproximação de A, é chamado de precondicionador explícito. Outro exemplo é o precondicionador que aproxima uma fatoração da inversa de A, onde temos o AINV [13], FSAI [47, 71], ISM-based [23], SPAI [38] e BIF [25, 26] como alguns exemplos encontrados na literatura. A vantagem de precondicionadores implícitos é que a aplicação do precondicionador da inversa é feita através da multiplicação matriz-vetor, uma operação paralelizável, sendo, assim, adequado para a utilização em computadores paralelos híbridos atuais, que contam com CPUs e GPUs [12,13,30,38,47,71]. Idealmente, o precondicionador deveria ser igual a A^{-1}, porém computar a inversa de forma exata tem um custo computacional tão elevado quanto resolver o sistema por método direto. Por isso é computada a inversa aproximada de A.

O objeto de estudo deste trabalho é o precondicionador AINV, proposto originalmente proposto por Benzi, Meyer e Tůma [12], em 1996, e encontra a fatoração da

inversa aproximada de matrizes simétricas positivas definidas. Em 1998, Benzi e Tůma apresentaram uma versão generalizada do AINV em [13], onde A pode ser qualquer matriz esparsa não singular. O AINV é baseado no algoritmo de A-biortogonalização, ou A-biconjugação, de Gram-Schmidt, descartando-se alguns elementos durante o processo, para garantir sua viabilidade e produzindo, assim, os fatores da inversa aproximada de A. Posteriormente, diversas variações do AINV foram publicadas, modificando-se alguns aspectos do algoritmo, como, por exemplo, as estratégias de descarte ou procedimentos que evitassem sua quebra [15, 22, 36, 66]. Algumas variações fizeram uso do AINV como método para se computar a fatoração LDU de A, a partir de suas relações com os fatores da inversa aproximada [16, 59, 60]. Outras variações resultaram da aplicação de pivoteamento completo no AINV a fim de se evitar a quebra do algoritmo ou acelerar a convergência de problemas difíceis [20, 61, 62].

Poucas variações do AINV foram desenvolvidas para matrizes em blocos, sendo as principais apresentadas em [11, 22]. Existem diversas vantagens em utilizar matrizes em blocos, seja com blocos artificiais ou oriundos de aspectos físicos do problema. Primeiramente, a blocagem é uma técnica padrão para melhorar o desempenho dos cálculos matriciais especialmente com matrizes esparsas, onde o endereçamento indireto de esquemas de armazenamento esparsos reduz a localização durante o produto de matriz esparsa por vetor denso [2, 27, 31]. Em muitas aplicações, como na simulação do reservatório de petróleo, cada ponto de malha pode estar relacionado a propriedades distintas (pressão e saturações de diversos hidrocarbonetos, por exemplo), originando uma estrutura de matriz com blocos densos e quadrados. Portanto, esses problemas podem se beneficiar da combinação de paralelismo proveniente da aplicação de um precondicionador do tipo inversa aproximada e operações de álgebra linear densa devido à ocorrência de blocos densos. Além disso, explícitos ou não, os precondicionadores são aproximações que devem ser razoavelmente significativas, especialmente quando se trata de equações lineares de sistemas físicos. Consequentemente, se na matriz original os coeficientes são blocados por orientações físicas, então é preferível um precondicionador blocado da mesma maneira. Existem na literatura alternativas em blocos para alguns dos métodos de inversa aproximada. Entre eles podemos encontrar, BFSAI [32, 42–44], BSPAI [8, 69], Block ISM-based [29] e BAINV [11].

Este trabalho tem o objetivo estudar os principais aspectos do AINV e de suas

variações a fim de desenvolver novas variações do AINV para matrizes em blocos. Primeiramente, são analisadas algumas características do método de biconjugação e do AINV. Depois é feita uma revisão das suas principais variações, analisando a maneira em que foram abordadas nos trabalhos que as propuseram, além de classificá-las de acordo com alguns aspectos inerentes a cada uma delas. Também analisamos as complexidades destes algoritmos. Posteriormente, estudamos a inversa aproximada em blocos, apresentando os resultados que obtivemos em [4]. Mais especificamente, nós demonstramos a consistência matemática do BAINV, algoritmo da inversa aproximada em blocos proposta por Benzi e Tůma em [11] para matrizes simétricas, e provamos que esse algoritmo não quebra para matrizes M e H. Por fim, nós propomos novos algoritmos para matrizes em blocos baseados no AINV e algumas de suas variações, para qualquer matriz não singular (não necessariamente simétrica). Nós demonstramos a consistência desses métodos e apresentamos os resultados de testes numéricos comparando dois deles.

Este trabalho está dividido da seguinte maneira: o Capítulo 1, trata das principais caracteríticas do AINV e do método de biconjugação. No Capítulo 2, é feita uma análise dos artigos em que as principais variações do AINV foram tratadas, além de propor uma classificação destas variações. No Capítulo 3, fazemos uma análise da complexidade das variações do AINV. No Capítulo 4, nós exibimos a demonstração da consistência matemática da inversa aproximada em blocos para matrizes simétricas (BAINV) e que ele, sob determinadas condições, é livre de quebra para matrizes M e H. No Capítulo 5 demonstramos a consistência do algoritmo de biconjugação em blocos para matrizes não simétricas e, utilizando alguns resultados, propomos algumas variações da inversa aproximada em blocos. Neste capítulo, também analisamos dois trabalhos sobre o AINV em blocos encontrados na literatura e exibimos os resultados dos testes numéricos executados utilizando dois dos algoritmos em blocos propostos. Por fim, apresentamos as Conclusões do trabalho.

1 AINV E SUAS CARACTERÍSTICAS

Seja $A \in \mathbb{R}^{n\times n}$ uma matriz esparsa e não singular. O precondicionador AINV ("Approximate Inverse"), proposto por Benzi e Tůma para matrizes simétricas positivas definidas (SPD) em [12] e para qualquer matriz não singular em [13], tem o objetivo de computar uma fatoração da inversa aproximada de A. O AINV tem como base o método de biconjugação completa que computa as matrizes Z, W e D, tais que $W^{\mathsf{T}}AZ = D$, onde 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 1. 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 1, 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 chamados de multiplicadores. Observemos que, caso A seja simétrica, então $Z = W$ e a fatoração fica como $A^{-1} = ZD^{-1}Z^{\mathsf{T}}$ ou $Z^{\mathsf{T}}AZ = D$.

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

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 1 em A. Seja Δ_i o i-ésimo menor principal líder de A, com $\Delta_0 = 1$, então*

Algoritmo 1: Biconjugação right-looking

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 $W \leftarrow (w_1, w_2, ..., w_n)$

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

Demonstração. A demonstração é direta por indução em (1). Ver [3].

□

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

Comentário 1. *Se o Algoritmo 1 não quebrar, 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*

$$d_{ii} = a_i^{\mathsf{T}} z_i = w_i^{\mathsf{T}} A z_i = c_i^{\mathsf{T}} w_i. \tag{2}$$

Também temos que

$$d_{ii} = a_i^{\mathsf{T}} z_i = z_i^{\mathsf{T}} A z_i \tag{3}$$

e

$$d_{ii} = c_i^{\mathsf{T}} w_i = w_i^{\mathsf{T}} A^{\mathsf{T}} w_i. \tag{4}$$

As igualdades (3) *e* (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*

Algoritmo 2: Biconjugação left-looking

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

1 $z_1, w_1 \leftarrow e_1$;
2 $d_{ii} \leftarrow a_{11}$;
3 **para** $j \leftarrow 2$ **até** n **faça**
4 $z_j^{(0)}, w_j^{(0)} \leftarrow e_j$;
5 **para** $i \leftarrow 1$ **até** $j-1$ **faça**
6 $r_j^{(i-1)} \leftarrow {a_i}^{\mathsf{T}} z_j^{(i-1)}$;
7 $s_j^{(i-1)} \leftarrow {c_i}^{\mathsf{T}} w_j^{(i-1)}$;
8 $z_j^{(i)} \leftarrow z_j^{(i-1)} - z_i \dfrac{r_j^{(i-1)}}{d_{ii}}$;
9 $w_j^{(i)} \leftarrow w_j^{(i-1)} - w_i \dfrac{s_j^{(i-1)}}{d_{ii}}$;
10 $z_j \leftarrow z_j^{(j-1)}$; $w_j \leftarrow w_j^{(j-1)}$;
11 $d_{jj} \leftarrow {a_j}^{\mathsf{T}} z_j$ ou ${c_j}^{\mathsf{T}} w_j$;
12 $D \leftarrow diag(d_{11}, d_{22}, ..., d_{nn})$, $Z \leftarrow (z_1, z_2, ..., z_n)$ e $W \leftarrow (w_1, w_2, ..., w_n)$

para evitar a quebra do AINV em matrizes positivas definidas, como será explicado mais detalhadamente na Seção 1.6.

As matrizes Z e W, em geral, são densas mesmo que A seja esparsa. Portanto, algumas entradas de Z e W devem ser descartas durante a execução do algoritmo, a fim de promover 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.

1.1 Versões right-looking e left-looking

Existem, pelo menos, duas maneiras de se executar o processo de biconjugação, conhecidas como right-looking, Algoritmo 1, e left-looking, Algoritmo 2. A diferença entre as versões right-looking e left-looking é como as matrizes Z e W são preenchidas. Na versão right-looking, a matriz 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 $j > i$ ainda serão atualizadas nas iterações seguintes. A Figura 1 mostra como ocorre o preenchimento da matriz Z por iteração. As regiões atualizadas da matriz estão destacadas por retângulos.

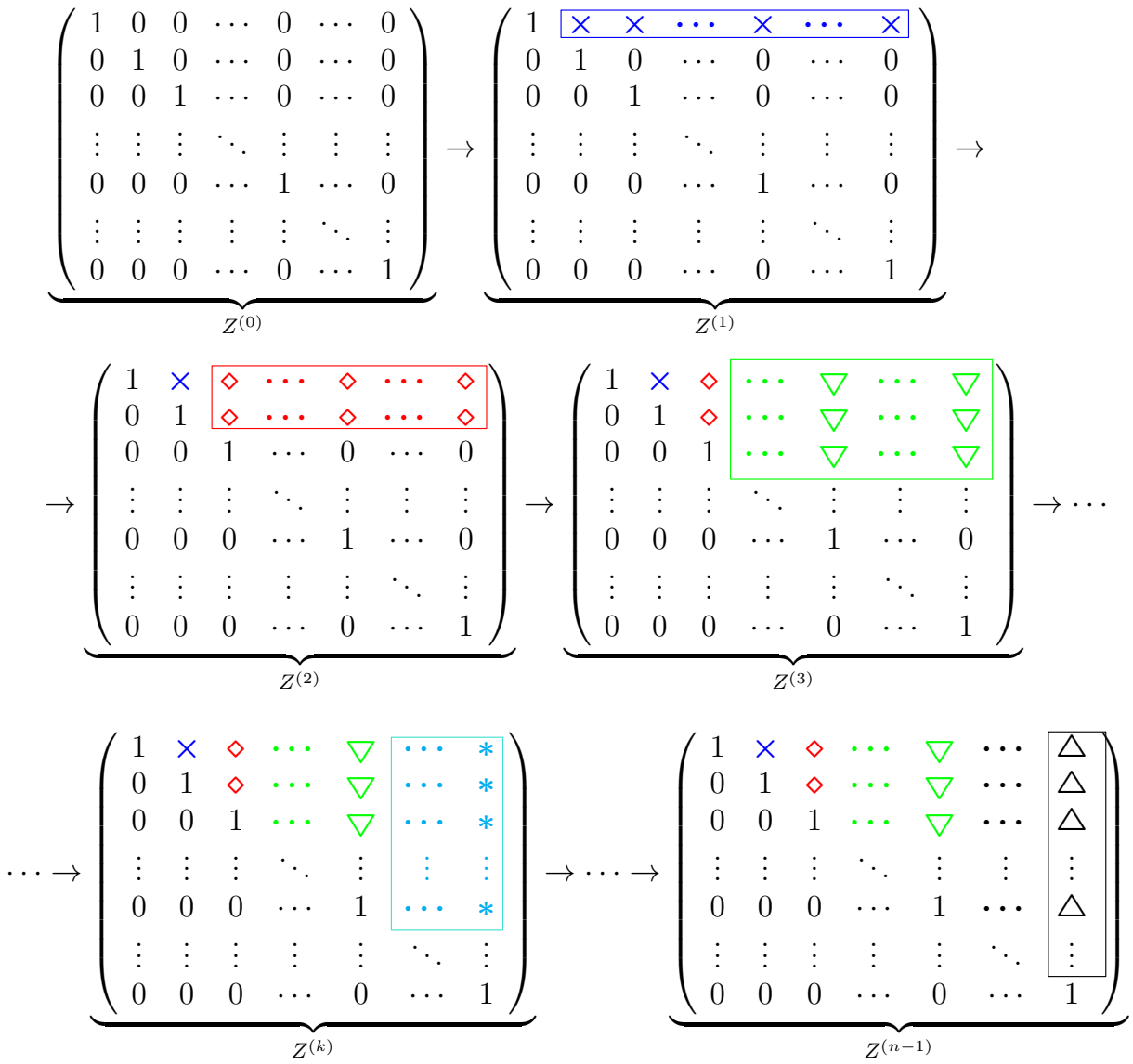

Figura 1 Esquema da biconjugação right-looking

Na versão left-looking, 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 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.

Proposição 2. *Os Algoritmos 1 e 2, em aritmética exata, produzem os mesmos resultados finais e intermediários.*

Demonstração. Usemos $\left(\square\right)_{\text{RL}}$ para representar o vetor ou escalar gerado pela versão right-looking (Algoritmo 1) e $\left(\square\right)_{\text{LL}}$ para a versão left-looking (Algoritmo 2).

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 a seguir é verdadeira para cada $j \in \{1, \ldots, n\}$.

$$H_j : \begin{cases} \left(z_j^{(i-1)}\right)_{\text{RL}} = \left(z_j^{(i-1)}\right)_{\text{LL}}, & \text{para } 1 \leqslant i \leqslant j \leqslant n; \quad (5a) \\ \left(w_j^{(i-1)}\right)_{\text{RL}} = \left(w_j^{(i-1)}\right)_{\text{LL}}, & \text{para } 1 \leqslant i \leqslant j \leqslant n. \quad (5b) \end{cases}$$

Primeiramente, provemos (5a): para $j = 1$, temos

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

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

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

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

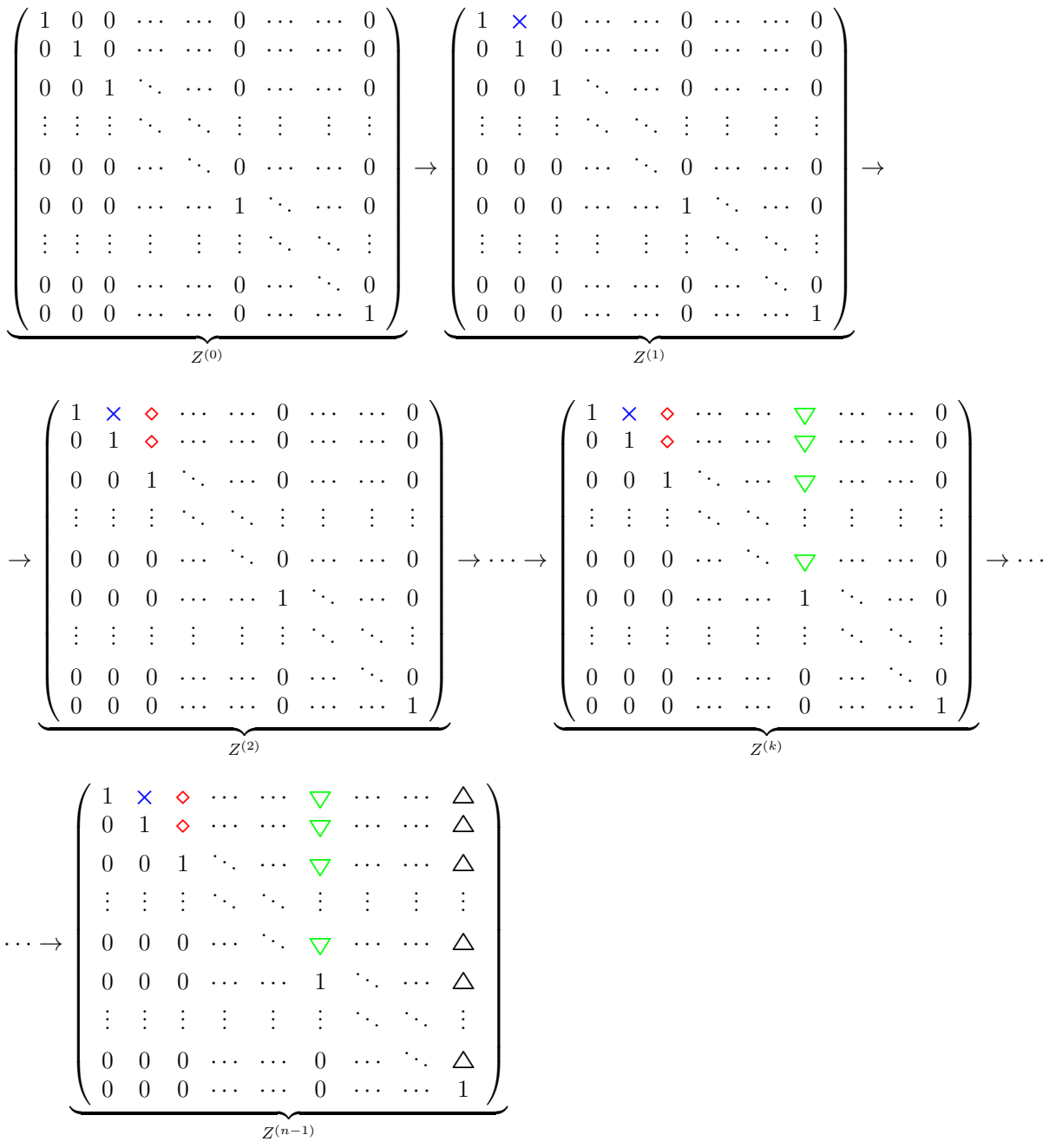

Figura 2 Esquema da biconjugação left-looking

obtendo assim,

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

Com isso, provamos (5a). A prova de (5b) é feita de forma análoga. □

Como as versões right-looking e left-looking se equivalem, produzindo os mesmos resultados finais e intermediários, trataremos os próximos resultados apenas utilizando a versão right-looking, Algoritmo 1, ao menos que se diga algo contrário. Os mesmos resultados podem ser obtidos utilizando a versão left-looking, Algoritmo 2.

1.2 Relações com os fatores L e U

Consideremos a fatoração

$$A = W^T D Z^{-1}, \tag{6}$$

obtida pelo Algoritmo 1, onde 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 (6) corresponde a esta fatoração. Ou seja,

$$W = L^T, \quad Z = U^{-1} \tag{7}$$

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

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

$$z_j = z_j^{(i)} - \sum_{k=i+1}^{j-1} z_k \frac{r_j^{(k-1)}}{d_{kk}} \tag{8}$$

e

$$w_j = w_j^{(i)} - \sum_{k=i+1}^{j-1} w_k \frac{s_j^{(k-1)}}{d_{kk}}, \tag{9}$$

para $0 \leqslant i < j$.

Demonstração. As igualdades (8) ((9)) seguem diretamente das linhas 3, 4, 6 e 8 (3, 4, 7 e 9) do Algoritmo 1. □

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 $A = LDU$, onde 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 $s_j^{(i-1)}$, onde $1 \leqslant i < j \leqslant n$, os escalares produzidos pelo Algoritmo 1. Então,*

$$u_{ij} = \frac{r_j^{(i-1)}}{d_{ii}} \tag{10}$$

e

$$l_{ji} = \frac{s_j^{(i-1)}}{d_{ii}}. \tag{11}$$

Demonstração. Seja e_j o j-ésimo vetor canônico, pelo Lema 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), $w_i^{\mathsf{T}} A z_i = d_{ii}$; assim,

$$0 = w_i^{\mathsf{T}} A e_j - \frac{r_j^{(i-1)}}{d_{ii}} w_i^{\mathsf{T}} A z_i \Rightarrow r_j^{(i-1)} = w_i^{\mathsf{T}} A e_j. \tag{12}$$

De acordo com (6) e (7),

$$Z^{-1} = U = D^{-1} W^{\mathsf{T}} A \Rightarrow u_{ij} = \frac{w_i^{\mathsf{T}} A e_j}{d_{ii}}. \tag{13}$$

Através de (12) e (13),

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

Analogamente, pelo Lema 1,

$$s_j^{(i-1)} = z_i^{\mathsf{T}} A^{\mathsf{T}} e_j \tag{14}$$

e

$$W^T = L = AZD^{-1} \Rightarrow l_{ji} = \frac{z_i^\mathsf{T} A^\mathsf{T} e_j}{d_{ii}}, \text{ com } 1 \leq i < j \leq n; \tag{15}$$

logo,

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

□

Logo, pela Proposição 3, além das entradas de L^{-1} e U^{-1} (W e Z, respectivamente), o método de biconjugação também computa implicitamente as entradas dos fatores L e U de A. É possível que a computação dessas entradas seja feita de forma explícita, adaptando o laço interno do Algoritmo 1, como pode ser visto no Algoritmo 3. Analogamente, para a versão left-looking, pode se adaptar o laço interno do Algoritmo 2, como é visto no Algoritmo 4.

Algoritmo 3: Computação das entradas dos fatores L e U no método de biconjugação right-looking

```
1 para j ← i + 1 até n faça
2     r_j^(i-1) ← a_i^T z_j^(i-1);
3     s_j^(i-1) ← c_i^T w_j^(i-1);
4     l_ji = s_j^(i-1) / d_ii;
5     u_ij = r_j^(i-1) / d_ii;
6     z_j^(i) ← z_j^(i-1) − z_i u_ij;
7     w_j^(i) ← w_j^(i-1) − w_i l_ji;
```

Algoritmo 4: Computação das entradas dos fatores L e U no método de biconjugação left-looking.

```
1 para i ← 1 até j − 1 faça
2     r_j^(i-1) ← a_i^T z_j^(i-1);
3     s_j^(i-1) ← c_i^T w_j^(i-1);
4     l_ji = s_j^(i-1) / d_ii;
5     u_ij = r_j^(i-1) / d_ii;
6     z_j^(i) ← z_j^(i-1) − z_i u_ij;
7     w_j^(i) ← 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 1; então,*

$$d_{ii} = a_{ii} - \sum_{k=1}^{i-1} \frac{r_i^{(k-1)} s_i^{(k-1)}}{d_{kk}}, \tag{16}$$

$$r_j^{(i-1)} = a_{ij} - \sum_{k=1}^{i-1} s_i^{(k-1)} u_{kj} \ e \tag{17}$$

$$s_j^{(i-1)} = a_{ji} - \sum_{k=1}^{i-1} r_i^{(k-1)} l_{jk}, \tag{18}$$

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

$$a_{ii} = \sum_{k=1}^{i} l_{ik} d_{kk} u_{ki} = l_{ii} d_{ii} u_{ii} + \sum_{k=1}^{i-1} l_{ik} d_{kk} u_{ki}, \text{ considerando (10) e (11),}$$

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

provando (16). Para $1 \leqslant i \leq j \leqslant n$,

$$a_{ij} = \sum_{k=1}^{i} l_{ik} d_{kk} u_{kj} = l_{ii} d_{ii} u_{ij} + \sum_{k=1}^{i-1} l_{ik} d_{kk} u_{kj}, \text{ considerando (10) e (11),}$$

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

provando (17). Para $1 \leqslant i \leq j \leqslant n$,

$$a_{ji} = \sum_{k=1}^{j} l_{jk} d_{kk} u_{ki} = l_{ji} d_{ii} u_{ii} + \sum_{k=1}^{j-1} l_{jk} d_{kk} u_{ki}, \text{ considerando (10) e (11),}$$

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

provando (18). □

Utilizando a Proposição 4, podemos computar 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 computar as entradas de U sem precisar utilizar a matriz Z. Exibimos, então, nos Algoritmos 5 e 6 as modificações nos laços internos dos Algoritmos 1 e 2, respectivamente, a fim de computar as entradas de L e U, sem utilizar Z, de forma explícita.

Algoritmo 5: Computação das entradas dos fatores L e U no método de biconjugação right-looking, 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 6: Computação das entradas dos fatores L e U no método de biconjugação left-looking, 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 1 e 2, gerando os Algoritmos 7 e 8, a fim de computar as entradas de L e U, sem utilizar W de forma explícita.

Algoritmo 7: Computação das entradas dos fatores L e U no método de biconjugação right-looking, sem utilizar W

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 $\quad s_j^{(i-1)} = a_{ji} - \sum_{k=1}^{i-1} r_i^{(k-1)} l_{jk}$;
4 $\quad l_{ji} = \frac{s_j^{(i-1)}}{d_{ii}}$;
5 $\quad u_{ij} = \frac{r_j^{(i-1)}}{d_{ii}}$;
6 $\quad z_j^{(i)} \leftarrow z_j^{(i-1)} - z_i u_{ij}$;

Algoritmo 8: Computação das entradas dos fatores L e U no método de biconjugação left-looking, sem utilizar W

1 **for** $i \leftarrow 1$ **to** $j-1$ **do**
2 $\quad r_j^{(i-1)} \leftarrow {a_i}^{\mathsf{T}} z_j^{(i-1)}$;
3 $\quad s_j^{(i-1)} = a_{ji} - \sum_{k=1}^{i-1} r_i^{(k-1)} l_{jk}$;
4 $\quad l_{ji} = \frac{s_j^{(i-1)}}{d_{jj}}$;
5 $\quad u_{ij} = \frac{r_j^{(i-1)}}{d_{jj}}$;
6 $\quad z_j^{(i)} \leftarrow z_j^{(i-1)} - z_j^{(j-1)} u_{ij}$;

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 [19] e são dadas por:

$$(S^{(i-1)})_{ji} = e_j A z_i^{(i-1)},\ (S^{(i-1)})_{ij} = (w_i^{(i-1)})^T A e_j, \qquad j \geq i, \tag{19}$$

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.

1.3 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 execução, 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 1 (versão right-looking), respectivamente. Abaixo listamos as principais estratégias de descartes utilizadas nos trabalhos sobre o AINV.

1.3.1 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, $\mid z_{kj}^{(i)} \mid < \tau$ ($\mid w_{kj}^{(i)} \mid < \tau$), onde 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 left-looking no Algoritmo 2.

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, [10, 12–15, 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). Com o progresso dos trabalhos, outras estratégias competitivas foram aparecendo, como veremos adiante.

1.3.2 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 $\mid w_{kj}^{(i)} \mid < \tau \parallel a_i \parallel_2$, onde $\parallel a_i \parallel_2$ 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, [60].

1.3.3 Padrão de zeros pré definido 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.

1.3.4 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 1.

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 [60], 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.

1.3.5 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.

1.3.6 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:

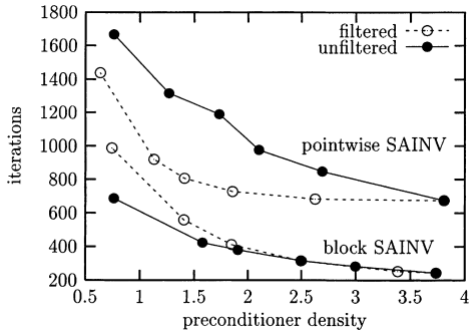

Figura 3 Efeito da pós filtragem. Fonte: [11]

No caso da figura Figura 3, podemos observar os resultados nos testes aplicados na matriz S3RMT3M3, do Matrix Market. Os valores dos parâmetros utilizados para o SBAINV são:

- Tolerâncias de descarte sem a pós filtragem: 0,1; 0,04; 0,03; 0,025 e 0,02;
- Tolerâncias utilizadas na pós filtragem: 0,3; 0,2; 0,15; 0,1 e 0,05.

Para o SAINV, os valores são:

- Tolerâncias de descarte sem a pós filtragem: 0,06; 0,035; 0,025; 0,02 e 0,015;
- Tolerâncias utilizadas na pós filtragem: 0,4; 0,2; 0,15; 0,1 e 0,06.

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

1.4 Apresentação do Algoritmo AINV

Nos Algoritmos 9 e 10 apresentamos as versões left-looking e right-looking do AINV, respectivamente, com os possíveis descartes.

Algoritmo 9: AINV right-looking

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 $W \leftarrow (w_1, w_2, ..., w_n)$

1.5 Reordenamentos e escalamentos na Literatura

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 sequência apresentá-los com mais detalhes:

Reescalamentos:

- 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 $\frac{1}{\alpha}A$, aonde A é a matriz de coeficientes de ordem $n \times n$.

Algoritmo 10: AINV left-looking

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);
$D \leftarrow diag(d_{11}, d_{22}, ..., d_{nn})$, $Z \leftarrow (z_1, z_2, ..., z_n)$ and $W \leftarrow (w_1, w_2, ..., w_n)$

- 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 $\frac{1}{\sqrt{a_{ii}}}$.

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

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

onde 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

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

Esse ordenamento foi fortemente recomendado no caso bloco do AINV no texto de [11] que concluíram que o uso de escalamento de Bloco Jacobi é superior ao demais, como mostra o resultado na Figura 4.

Nesse exemplo, a matriz usada foi a S3DKT3M2 do Matrix Market, ψ é tolerância

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 4 Comparação entre os diferentes escalamentos. Fonte: [11].

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 [5].
- MultiLevel Nested Dissection reordering (MNLD) - tirado de [45] e [46].
- Minimum Degree (MIP) - tirado de [21].
- DCR - tirado de [50].

1.6 Quebra no AINV

Vimos que o método de biconjugação pode quebrar por conta da ocorrência de pivôs nulos ou muito pequenos e que era garantido que o método não quebrava 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 quebra para matrizes SPD, pelo critério de Sylvester, mas com os descartes, ele pode quebrar. Entretanto, para algumas famílias de matrizes, o AINV não quebra, como veremos a seguir. Primeiramente, vejamos algumas definições.

Definição 1. *Uma matriz $A \in \mathbb{R}^{n\times n}$, $A = (a_{ij})$, é estritamente diagonal dominante se*

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

Definição 2. *Uma matriz quadrada A é uma Z-matriz se $a_{ij} \leq 0, \ \forall\, i \neq j$.*

Definição 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* $s \geqslant \rho(B)$*, onde* $\rho(B)$ *denota o raio espectral de B. Nós denotamos por* $\mathcal{M}$ *o conjunto de todas as M-matrizes.*

Definição 4 (Matriz comparação, ver [18])**.** *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 5 (H-matriz)**.** *Uma matriz é uma H-matriz se sua matriz comparação for uma M-matriz. Ver [18]. Neste caso, a sua matriz comparação é a sua M-matriz associada.*

A seguir temos dois resultados que asseguram que o AINV não quebra 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 computados 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 quebre (a não ser pela ocorrência de pivôs de magnitude muito baixa). De acordo com [53], qualquer matriz diagonal dominante é uma matriz H. Assim sendo, se o processo de biconjugação quebrar, uma

alternativa, sugerida em [15], 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 quebra, 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 quebra 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 apenas quando a necessidade surgir, modificando os pivôs se sua magnitude for menor que um limite especificado.

Uma forma de evitar a quebra para matrizes SPD é utilizando 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 [15], 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 [17]). Em particular, $\Delta - R$ é semipositiva definida, 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 [17]). A ideia, então, é aplicar o AINV na matriz $\hat{A}$ (sendo livre de quebra, 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, devido ao fato de A estar 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 [15], uma modificação no cálculo dos pivôs a fim de assegurar que ele sejam positivos, garantindo que o processo não quebre. 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 (3) e (4). Porém, ao utilizarmos os descartes nas entradas dos vetores z_i e w_i, as igualdades (3) e (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 quebre, 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 9 (right-looking) e 10 (left-looking) pelas expressões

$$d_{ii} \leftarrow (z_i^{(i-1)})^\mathsf{T} A z_i^{(i-1)} \text{ ou } (w_i^{(i-1)})^\mathsf{T} A w_i^{(i-1)}, \tag{20}$$

$$d_{jj} \leftarrow (z_j^{(j-1)})^\mathsf{T} A z_j^{(j-1)} \text{ ou } (w_j^{(j-1)})^\mathsf{T} A w_j^{(j-1)}, \tag{21}$$

respectivamente.

Estas fórmulas foram bastante utilizadas em trabalhos posteriores sobre variações do AINV como principal ferramenta para evitar a quebra de matrizes positivas definidas.

2 PRINCIPAIS VARIAÇÕES DO AINV

Neste capítulo, fazemos uma breve descrição de cada uma das principais variações do AINV encontradas na literatura desde que ele foi proposto e como foram abordadas pelos autores nos trabalhos.

2.1 AINV

O AINV foi originalmente proposto por Benzi e Tůma em [12], em 1996, para matrizes 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 right-looking, como no Algoritmo 1 (sem fazer o cálculo de W). No trabalho, eles demonstram as Proposições 5 e 6, garantindo a que ele não quebra para matrizes M e H.

Nos testes numéricos apresentados, eles utilizam o AINV para gerar precondicionadores de diferentes matrizes para o método Gradientes Conjugados (CG) [39]. 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 quebra, além de restringir a ocorrência de elementos muito grandes em Z. Se algum pivô p_i fosse menor que $\sqrt{\varepsilon_M}$, onde ε_M é a precisão de máquina, então ele poderia ser substituído por

$$p_i \longleftarrow \max\{\sqrt{\varepsilon_M}, \mu\sigma\theta\}, \tag{22}$$

onde

$$\sigma = \max_{i \leq k \leq n-1} \{r_k^{(i-1)}\}, \theta = \|z_i^{(i-1)}\|_\infty \neq 0 \tag{23}$$

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 1 AINV

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 (IC) [68] 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 dois do resíduo não precondicionado ficou 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 1 exibimos um resumo das principais características deste trabalho.

Comentário 2. *Aqui nos referimos como AINV ao algoritmo proposto por Benzi 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 1, ao menos que especifiquemos que seja para o caso simétrico.*

2.2 AINV-NS

O AINV-NS ("AINV-Nonsymmetric") 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 1 deste trabalho, correspondendo ao Algoritmo 1. A versão utilizada foi a right-looking, mas eles também citaram a versão left-looking como opção (Algoritmo 2). No artigo, eles reafirmaram que o algoritmo não quebra 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 da fatoração ILU(0) [65] e servidos como precondicionadores para os métodos BIGSTAB, QMR e GMRES (ver métodos em [9]). 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 2 do resíduo não precondicionado ficou 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 comparadas em média, com as taxas 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 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 2 AINV-NS

2.3 SAINV

O SAINV ("Stabilized AINV") foi desenvolvido por Benzi e Tůma em [15] para matrizes simétricas, em 2000, e seu principal diferencial é o fato dele não quebrar para qualquer matriz SPD. Isso é possível pois o método do SAINV é semelhante ao do 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 (20), garantindo ser livre de quebra para matrizes SPD.

Como o SAINV calcula o pivô utilizando produto matriz-vetor, é natural que haja questionamento sobre seu custo, já que ele tem um custo maior que o AINV (para matrizes simétricas de [12]) original, que 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 computar 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 1.6) afim de se evitar a quebra. 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 dois do resíduo inicial foi menor que 10^{-8} ou quando o método utilizou mais de 10000 iterações. O lado direito foi construído como $b = Ax$, aonde x é um vetor com entradas randômicas uniformemente distribuídas em (0,1). Eles também utilizaram escalamento Jacobi e reordenamento MMD. Diante 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 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 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 SAINV

2.4 SAINV-NS

Em 2000, Bridson e Tang desenvolveram o SAINV-NS ("Stabilized AINV-Nonsymetric"), 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 no método 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 (12) e (14)). Ou seja, dado o Algoritmo 2, as linhas 6 e 7 são substituídas pelas equações (12) e (14), respectivamente. Já o pivô é dado por $d_{jj} = w_j^T A z_j$, substituindo a linha 11 do Algoritmo 2 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 $\frac{z_i r_j^{(i-1)}}{p_i}$ e $w_i \frac{s_j^{(i-1)}}{q_i}$ das linhas 8 e 9 do Algoritmo 2, 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 dois do resíduo ficou menor que 10^{-6}. Eles compararam as versões left-looking e right-looking do SAINV-NS, concluindo que os precondicionadores melhoraram os desempenhos dos métodos iterativos, porém a versão right-looking 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 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 4 NS-SAINV

2.5 AINVP

Em 2002, Bollhofer e Saad propuseram o AINVP ("AINV with Pivoting") para quaisquer matrizes, em [20]. A ideia sugerida é aplicar o processo de pivoteamento completo na versão right-looking do AINV, baseado no processo de pivoteamento completo da eliminação gaussiana com ordem IJK. A principal motivação do uso de pivoteamento é evitar o aparecimento de pivôs nulos ou de baixa magnitude durante o algoritmo, ou seja, evitar a quebra.

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, diante de determinada condição, com intuito de evitar a quebra. 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 1). 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 AINVP é executado como no Algoritmo 11.

Algoritmo 11: AINVP

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

1 $z_i^{(0)}, w_i^{(0)} \leftarrow e_i$, $i = 1, \ldots, n$;
2 $\Pi = \Sigma = I_n$;
3 Tolerância $\alpha \in (0, 1]$;
4 **para** $i \leftarrow 1$ **até** n **faça**
5 satisfied_s =Falso, satisfied_r =Falso;
6 **enquanto** *não satisfied_s* **faça**
7 $s_j^{(i-1)} = (w_j^{(i-1)})^T (\Pi A\Sigma) z_i^{(i-1)}$ $j \geq i$;
8 **se** $|s_i^{(i-1)}| < \alpha max_{m \geq i+1} |s_m^{(i-1)}|$ **então**
9 satisfied_r =Falso;
10 Escolher k tal que $|s_k^{(i-1)}| = max_{m \geq i+1} |s_m^{(i-1)}|$;
11 Permutar as colunas i e k de $W - I$ e linhas i e k de Π e os elementos $s_i^{(i-1)}$ e $s_k^{(i-1)}$;
12 satisfied_s =Verdadeiro;
13 $r_j^{(i-1)} = (w_i^{(i-1)})^T (\Pi A\Sigma) z_j^{(i-1)}$ $j \geq i$;
14 **se** $|r_i^{(i-1)}| < \alpha max_{m \geq i+1} |r_m^{(i-1)}|$ **então**
15 satisfied_s =Falso;
16 Escolher k tal que $|r_k^{(i-1)}| = max_{m \geq i+1} |r_m^{(i-1)}|$;
17 Permutar as colunas i e k de $Z - I$ e Σ e os elementos $r_i^{(i-1)}$ e $r_k^{(i-1)}$;
18 satisfied_r =Verdadeiro;
19 $d_{ii} = r_i^{(i-1)}$ ou $s_i^{(i-1)}$;
20 $z_i = z_i^{(i-1)}$; $w_i = w_i^{(i-1)}$;
21 **para** $j \leftarrow i + 1$ **até** n **faça**
22 $z_j^{(i)} \leftarrow z_j^{(i-1)} - z_i \frac{r_j^{(i-1)}}{d_{ii}}$;
23 $w_j^{(i)} \leftarrow w_j^{(i-1)} - w_i \frac{s_j^{(i-1)}}{d_{ii}}$;
24 Aplicar estratégia de descarte em $z_j^{(i)}$ e $w_j^{(i)}$.
25 $D \leftarrow diag(d_{11}, d_{22}, ..., d_{nn})$, $Z \leftarrow (z_1, z_2, ..., z_n)$ e $W \leftarrow (w_1, w_2, ..., w_n)$

Abaixo explicamos, de forma geral, as etapas do Algoritmo 11 a cada iteração i:

- São obtidos valores de $s_i^{(i-1)}, ..., s_n^{(i-1)}$ pela equação $s_j^{(i-1)} = w_j^T (\Pi A\Sigma) z_i$, $j \geq i$ (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 os valores de $r_i^{(i-1)}, ..., r_n^{(i-1)}$ pela equação $r_j^{(i-1)} = w_i^T(\Pi A \Sigma)z_j$, $j \geq i$ (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 ate 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 quebra, 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 com AINVP com o ILU e ILUP para os métodos GMRES(30) e QMR. O critério de parada utilizado foi quando o método atingiu um máximo de 500 iterações, ou se a norma relativa do resíduo ficou menor 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 com bastante robustez, principalmente em problemas difíceis. Na Tabela 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 5 AINVP

2.6 RIF

Em 2003, Benzi e Tůma desenvolveram o RIF ("Robust Incomplete Factorization"), em [16]. Primeiramente, os autores apresentaram o problema dos mínimos quadrados que busca resolver

$$\|b - Ax\|_2 = min \tag{24}$$

onde A é esparsa, de ordem $m \times n$ e possui posto completo. Naturalmente, o problema vale para o caso de $m = n$, mas o maior interesse é quando $m > n$. Para resolver (24) podemos usar métodos diretos ou métodos iterativos aplicados implicitamente às equações normais do problema, dadas por

$$Cx = f, \tag{25}$$

onde

$$C = A^T A, f = A^T b. \tag{26}$$

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 $C = A^tA$ precise ser explicitamente calculada;

2 A fatoração incompleta não quebre;

3 O armazenamento intermediário seja 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 root-free de Cholesky), onde L é diagonal inferior unitária. Então $C^{-1} = L^{-T}D^{-1}L^{-1}$ e, portanto, $Z = L^{-T}$ (como visto na Seção 1.2). Sabemos que, para $L = [l_{ji}]$, então $l_{ji} = \frac{r_j^{(i-1)}}{d_{ii}}$ (ver equação (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 3, considerando apenas o cálculo das entradas de L e Z. A fatoração LDL aproximada seria, então, utilizada como precondicionador para (25).

Como o processo é baseado no SAINV, então o algoritmo é livre de quebra pois $C = A^TA$ 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 $z_i^TCz_j = z_i^TA^TAz_j = (Az_i)^T(Az_j)$, $1 \leq i \leq j \leq n$. Portanto, A^TA não precisa ser explicitamente calculada. O segundo item é garantido pois o RIF não quebra 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, onde 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 [16], IMGS [1] e IGR [57] para serem aplicados no CGLS. 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 dois do resíduo relativo ficou 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 quebra, dependendo do valor de descarte utilizado. Na Tabela 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 6 RIF

2.7 ISAINV

Em 2004, Seiji Fujino e Yusuki Ikeda apresentaram o ISAINV ("Improved Stabilized AINV"), em [36] para matrizes SPD. Eles se basearam na versão right-looking 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 9, 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 método 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 critério de parada utilizado foi quando a norma dois do resíduo relativo ficou 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 máquina para a para 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 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 7 ISAINV

2.8 SAINV-VAR

O SAINV-VAR ("SAINV Variety") foi elaborado por Rafiei e Toutounian em [59], em 2008, para ser aplicado em quaisquer matrizes positivas definidas. Eles se basearam

na versão right-looking 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 (10)), ou seja, utilizando os multiplicadores gerados o 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 (17)). Dessa forma, não seria necessária a utilização de Z para calculá-los. O algoritmo é feito tomando como base o Algoritmo 1, mas reescrevendo o laço "para"do j pelo laço mostrado no Algoritmo 5, 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:

$$Z_l = I + F + F^2 + F^3 + ... + F^l \tag{27}$$

onde $F = I - U$ e I é a identidade. Por meio de (27) e do método de Horner [40], 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 critério de parada utilizado foi quando a norma dois do resíduo ficou 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 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 8 SAINV-VAR

2.9 FFAPINV

Em 2010, Salkuyeh propôs um algoritmo em [66], baseado no de Lee e Zhang [49]. denominado FFAPINV ("Forward Factored Approximate Inverse"). O FFAPINV pode ser utilizado em qualquer matriz não singular e tem como base a versão left-looking do AINV-NS. 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 (12) e (14)). Ou seja, dado o Algoritmo 2, as linhas 6 e 7 são substituídas pelas equações (12) e (14), 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 (21)), garantindo que o algoritmo não quebre para qualquer matriz positiva definida. Salkuyeh também mostrou em um trabalho anterior [67] que, sem considerar os descartes, o AINV e o FFAPINV são matematicamente equivalentes.

A estratégia de descarte foi aplicada em duas partes. Caso, $\mid \frac{r_j^{(i-1)}}{d_{ii}} \mid < \tau$ e $\mid \frac{s_j^{(i-1)}}{d_{ii}} \mid < \tau$, 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 FFAPINV, 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 dois 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 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 9 FFAPINV

2.10 RIF-NS

Em 2011, Rafiei e Bollhöfer propuseram o RIF-NS ("Nonsymetryc RIF"), em [60], 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 quebrar se a matriz for positiva definida. Nesse caso, o pivô seria calculado como em (20).

Primeiramente, eles tratam da sua versão right-looking que é baseada no AINV

right-looking. 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 (11) e (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 1). Porém, os multiplicadores $r_j^{(i-1)}$ são calculados como (ver equação (17)):

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

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 5.

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

$$|w_{lj}| < \varepsilon_{LW}, \tag{28}$$

$$|l_{ji}| \|e_i^T L^{-1}\|_\infty < \varepsilon_{LW}, \tag{29}$$

$$|u_{ij}| \|e_i^T U^{-1}\|_\infty < \varepsilon_U, \tag{30}$$

onde w_{lj} é a entrada l do vetor w_j na iteração i, com $1 \leq l < i < j \leq n$. As relações (29) e (30) 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 (29) para

$$|l_{ji}| \|w_i\|_\infty < \varepsilon_{LW}, \tag{31}$$

já que $\|w_i\|_\infty \approx \|L^{-T} e_i\|_\infty = \|e_i^T L^{-1}\|_\infty$. Desta forma, a condição (31) 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 [60]) 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 left-looking, a ideia é semelhante, porém com alguns ajustes em relação aos índices.

Nos testes foram comparados RIF-NS right-looking, RIF-NS left-looking, AINV left-looking e ILUT para os métodos BICGSTAB, GMRES(30) e TFQMR [34]. Foram feitos escalamento NSSM (ver [60]) e reordenamento MNLD. O valor inicial nos testes foi o vetor nulo e lado direito foi b, onde $b = Ae$, sendo e um vetor com estradas iguais a 1. O critério de parada utilizado foi quando a norma dois do resíduo relativo ficou menor que 10^{-10}. Os autores concluíram que a versão left-looking 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 menores número de iterações do que o RIF-NS. Portanto, eles concluíram que a versão left-looking do RIF-NS obteve melhores resultados que o AINV, porém resultados um pouco inferiores que o ILUT. Na Tabela 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 10 RIF-NS

2.11 RIFP

Em 2012, Rafiei propôs o RIFP ("RIF with pivoting") em [61], que é uma adaptação do processo de pivoteamento do AINVP para o RIF. Ele é proposto para quaisquer matriz não singular. O pivoteamento é feito de maneira análoga à descrita na Seção 2.5, 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 (11) e (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 12.

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 right-looking 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 atingiu um máximo de 5000 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 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 11 RIFP

2.12 LLAINVP

Em 2014, Rafiei propôs o LLAINVP ("Left Looking Version of AINV with Pivoting") para quaisquer matrizes não singulares, em [62]. Baseado no trabalho de Bollhofer e Saad [20], que aplica estratégia de pivoteamento completo na versão right-looking do AINV, ele sugere também o pivoteamento completo, só que na versão left-looking. Assim como no AINVP, a estratégia é baseada no pivoteamento completo da versão eliminação gaussiana com ordem IJK [63].

Aqui comentamos, de forma generalizada, o processo de pivoteamento completo para o AINV left-looking. O processo é análogo ao AINVP, (descrito na Seção 2.5), 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 determinada condição. 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 permutação Π e Σ, que no início são dadas como a identidade I. Mas apesar de o processo ser análogo, existem algumas diferenças, inerentes ao fato de se trabalhar com a versão left-looking. Uma delas é o cálculo dos multiplicadores $s_j^{(j-1)}, ..., s_j^{(n)}$. Na versão right-looking, esse cálculo se dá através de $\Pi A \Sigma$ e dos vetores de W e Z da iteração anterior. Porém, na

versão left-looking, 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 (14)) para achar tais valores. Para isso, os vetores $z_j^{(1)}, ..., z_j^{(i-1)}$ são previamente atualizados, como no Algoritmo 2. 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 2 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 13.

Abaixo explicamos, de forma geral, as etapas do Algoritmo 13 a cada iteração j:

- São atualizados os vetores $z_j^{(1)}, ..., z_j^{(j-1)}$ como no Algoritmo 2 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 2 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 os vetores $z_j^{(1)}, ..., z_j^{(j-1)}$ e os valores $s_j^{(j-1)}, ..., s_n^{(j-1)}$ 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 $(\Pi A \Sigma)^{-1} \approx Z D^{-1} W^T$. É enfatizado que a principal motivação do uso de pivoteamento é evitar o aparecimento de pivôs nulos ou de baixa magnitude durante o 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 quebram na fase de construção do precondicionador ou que não convergem.

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 também variou entre os valores $0,1$, $0,5$, $0,8$ e 1. Os resultados foram comparados com o AINV left-looking 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 5000 iterações. Foi utilizado ordenamento MNLD. Os autores concluíram que quando a tolerância foi de $0,1$ os testes com o LLAINVP obteve 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 a 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 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 12 LLAINVP

2.13 Classificação do AINV

Após as análises e reflexões sobre o Algoritmo de Inversa Aproximada, nós agrupamos as suas diversas variações encontradas na literatura em quatro classes. Essa separação dos Algoritmos em classes foi percebida, principalmente, durante a implementação dos mesmos na linguagem C++ e leva em consideração elementos como: armazenamentos do Algoritmo, uso de memória compartilhada e a forma como alguns cálculos são efetuados. Esperamos com a presente seção clarificar as proximidades e diferenças entre as versões do Algoritmo de Inversa Aproximada e estabelecer estruturas gerais sobre as quais os Algoritmos serão considerados casos especiais.

As quatro classes gerais de 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.

2.13.1 Classe AINV

Essa classe é formada pelas variações: AINV, AINV-NS, SAINV e ISAINV. Elas têm como base o Algoritmo 1. 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 serve para matrizes não simétricas, operando exatamente como no Algoritmo 1. 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 1. Já o SAINV e ISAINV calculam o pivô como $d_{ii} = z_i^T A z_i$, garantindo ser livre de quebra 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 $z_j^{(i)}$.

2.13.2 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.

Estruturado 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 quebre 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 em 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 em $z_j^{(i)}$ e $w_j^{(i)}$ e, posteriormente, são feitos descartes em $z_j^{(i)}$ e $w_j^{(i)}$.

2.13.3 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 como meio para o cálculo. Esse cálculo é possível, pelas relações (10) e (11) exibidas e demonstradas na Seção 1.2. 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 quebram para matrizes positivas definidas por utilizarem a relação (20). 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 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 caso.

Já o SAINV-VAR computa 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 que são armazenados até o final, descartando-se os vetores de W.

2.13.4 Classe Pivoteamento

A classe Pivoteamento é formada pelas variações AINVP, RIFP e LLAINVP, sendo a principal característica dessa classe o uso de pivoteamento completo durante a execução do AINV. A implementação de um pivoteamento completo no contexto do 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 quebra, 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:

- Right ou left-looking - O AINVP e RIFP são apresentados na versão right-looking do AINV, já o LLAINVP é mostrado na versão left-looking. Essa é a principal diferença entre o AINVP e o LLAINVP, já que a estrutura do algoritmo com pivoteamento na versão left-looking é bastante diferente devido algumas peculiaridades explicadas com mais detalhes na Seção 2.12.

- Matrizes produzidas - O AINVP e 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 computando L e U através das fórmulas (10) e (11). Além disso, também são efetuadas permutações em L e U para manter a consistência do algoritmo (mais detalhes na Seção 2.11).

Algoritmo 12: 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**
        $s_j^{(i-1)} = (w_j^{(i-1)})^T (\Pi A \Sigma) z_i^{(i-1)}$ $j \geq i$;
        if $|s_i^{(i-1)}| < \alpha max_{m \geq i+1} |s_m^{(i-1)}|$ **then**
            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 13: LLAINVP

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

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

$\Pi = \Sigma = I_n$;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 $|r_j^{(j-1)}| < \alpha \max\limits_{m \geq j+1} |r_m^{(j-1)}|$ **então**

        $n_j = n_j + 1$; $\sigma_{n_j}^{(j-1)} = I_n$; satisfied_$s$ = Falso;

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

      Permutar as linhas j e k de $\sigma_{n_j}^{(j-1)}$ e os elementos $r_j^{(j-1)}$ and $r_k^{(j-1)}$;

      $\Sigma = \Sigma \sigma_{n_j}^{(j-1)}$;

      satisfied_r =Verdadeiro;

  $d_{jj} = s_j^{(j-1)}$; $z_j = z_j^{(j-1)}$; $w_j = w_j^{(j-1)}$;

$D \leftarrow diag(d_{11}, d_{22}, ..., d_{nn})$, $Z \leftarrow (z_1, z_2, ..., z_n)$, and $W \leftarrow (w_1, w_2, ..., w_n)$

3 COMPLEXIDADE DO AINV

Neste capítulo vamos estudar a complexidade do algoritmo de Inversa Aproximada. Iremos com esse estudo verificar que, sem a existência de descarte, o algoritmo é inviável. Nas seções seguintes verificaremos como modificar a ordem da complexidade com algumas escolhas computacionais. Começaremos fazendo essa análise para 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 right-looking (a versão left-looking 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:

$$\sum_{i=1}^{n} -4i^2 + 4ni = -4\sum_{i=1}^{n} i^2 + (4n)\sum_{i=1}^{n} i. \tag{32}$$

Simplificando (32) 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 $d_{ii} = a_i^{\intercal} z_i$, 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 quebra 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:

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

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 $1 \leqslant i \leqslant n$, onde $nnz(z_{:,i})$ 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 right-looking (a versão left-looking 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 $d_{ii} = a_i^{\intercal} z_i$, 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 quebra 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 $C(ISAINV) = (5N + 1)Mn + (2N^2 + N - 1)n$.

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

As versões não simétricas AINV-NS e SAINV-NS necessitam do cálculo de Z e W e seu custo será, precisamente, o dobro do cálculo de Z. Logo, $C(Z) = 2 \cdot \sum_{i=1}^{n} 5NM = 2 \cdot 5NMn = 10NMn$. No AINV-NS, o cálculo do pivô é o mesmo que o cálculo do AINV, então teremos $C(AINV - NS) = 10NMn + (2N - 1)n$. Já, 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, $C(SAINV - NS) = 10NMn + (2N^2 + N - 1)n$.

No caso do FFAPPINV, é oportuno observar que o cálculo de atualização das matrizes Z e W possuem a mesma quantidade de operações: 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-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, $C(RIF) = C(SAINV - VAR)$.

Já o AINVP, dentre todos os algoritmos estudados aqui é mais complicado e caro. Isso se dá pelo pivoteamento. 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 multplicadores 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 arma-

zenado ao final. Portanto, $C(RIFP) = C(AINVP)$.

Quanto ao LLAINVP, sua principal diferença em relação ao AINVP é o fato dele ser left-looking, 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

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

Algoritmo	Custo
A-conjugação	$\frac{2n^3}{3} - \frac{2n}{3}$
A-conjugação livre de quebra para 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 13 Custo dos algoritmos

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 $C(AINVP) = (7NM + M)Pn$.

Diante do estudo aqui apresentado, sintetizamos na Tabela 13 os custos de cada um 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.

4 INVERSA APROXIMADA EM BLOCOS PARA MATRIZES SIMÉTRICAS

Neste capítulo, apresentamos os resultados do trabalho que desenvolvemos em [4], a respeito do suporte teórico para a inversa aproximada em blocos (BAINV) de matrizes simétricas, precondicionador proposto por Benzi, Meyer e Tůma, em 2001. Nós provamos sua consistência e que para as classes de matrizes do tipo M e H o algoritmo não quebra, independente da estratégia de descarte adotada.

4.1 Notação

4.1.1 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:

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

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

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

Na sequência, (exceto na Subseção 4.1.2), nós assumiremos que os blocos têm tamanho t fixado.

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, com $I = 1, 2, \ldots, N$, 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 (4.1.1), 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 $I < J$.

4.1.2 Estrutura em blocos heterogêneos (tamanho de blocos variável)

Em algumas aplicações (por exemplo, o método AIM de simulação de reservatório de petróleo, ver [33, 70], 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. Isto está relacionado com o número variado de incógnitas por célula de grade.

Neste caso, nós descartamos 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 (4.1.1), porém, neste caso, cada A_{IJ} é (possivelmente) um bloco retangular $n_I \times n_J$. Esta estrutura é apresentada mais detalhadamente em [11].

4.1.3 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, $|A|_{ij} = |a_{ij}|$.

4.2 A-conjugação em blocos

Seja A uma matriz simétrica em blocos.

Definição 6 (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, onde $0 \leq I \leq K$ 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 $Z^{\mathsf{T}}AZ = D$, onde D é bloco-diagonal. Note que, neste caso, se A e Z são não singulares, então

$$A^{-1} = ZD^{-1}Z^{\mathsf{T}}. \tag{33}$$

A decomposição (33) pode ser obtida através do algoritmo da versão right-looking da A-conjugação em blocos, Algoritmo 14 (que é essencialmente o mesmo algoritmo em [11]). Nós também apresentamos a versão left-looking, Algoritmo 15. É fácil verificar que ambos Algoritmos 14 e 15 produzem os mesmos resultados finais e parciais, podendo ser provado de forma análoga à versão escalar, no Capítulo 1.

Nós demonstraremos, no Teorema 1, que sob determinadas condições, o Algoritmo 14 realmente produz a decomposição (33). Este teorema também estabelece as

Algoritmo 14: A-conjugação em blocos right-looking

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

Algoritmo 15: A-conjugação em blocos left-looking

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**
        $R_J^{(I-1)} \leftarrow A_I^\intercal Z_J^{(I-1)}$;
        $Z_J^{(I)} \leftarrow Z_J^{(I-1)} - Z_I D_{II}^{-1} R_J^{(I-1)}$;
    $Z_J \leftarrow Z_J^{(J-1)}$;
    $D_{JJ} \leftarrow A_J^\intercal Z_J$;
$D \leftarrow diag(D_{11}, ..., D_{NN})$ e $Z \leftarrow \left[\; Z_1, \;\; \cdots \;\; , Z_N \; \right]$;

bases para nossos principais resultados, Teoremas 2 e 3. Vejamos, primeiramente, dois resultados auxilares, Lemas 2 e 3.

Lema 2. *Caso o Algoritmo 14 não quebre (i.e., todos as matrizes D_{II}'s produzidas são não-singulares), ele produz uma matriz bloco-triangular superior Z. Além disso, sua diagonal em blocos (ou bloco-diagonal) é formada por matrizes identidade.*

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

$$E_L^T Z_J^{(I)} = \delta_{LJ}, \quad \forall\ J, L \ \text{ tal que } \ I < J \leq L \leq N, \tag{34}$$

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 (34) é claramente verdadeira para $I = 0$, já que $Z_J^{(0)} = E_J$. Agora, assumamos que (34) é verdadeira para algum $0 \leq I < N-1$. A partir dessa hipótese, somado a linha 7 do Algoritmo 14,

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

completando a indução. □

Lema 3. *Caso o Algoritmo 14 não quebre, temos que, para algum $1 \leq J \leq N$ fixado,*

$$Z_J = Z_J^{(K)} - \sum_{I=K+1}^{J-1} Z_I D_{II}^{-1} R_J^{(I-1)}, \tag{35}$$

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

Demonstração. Segue diretamente das linhas 3 e 7 do Algoritmo 14. □

Teorema 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 14 não quebra 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 $J \in \{1, \ldots, N\}$.

$$H_J : \begin{cases} A_I^\mathsf{T} Z_J = 0, & I \in \{1, 2, \ldots, J-1\}; \quad (36a) \\ Z_I^\mathsf{T} A Z_J = Z_J^\mathsf{T} A Z_I = 0, & I \in \{1, 2, \ldots, J-1\}; \quad (36b) \\ Z_J^\mathsf{T} A Z_J = D_{JJ}; & (36c) \\ D_{JJ} \text{ é não singular}; & (36d) \end{cases}$$

As hipóteses (36b), (36c), e (36d) demonstram o resultado pretendido, enquanto a hipótese (36a) serve como suporte técnico para a demonstração.

Para $J = 1$, as condições (36a) e (36b) são verdadeiras por vacuidade e as condições (36c) e (36d) 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 (36a), multiplicamos pela esquerda a linha 7 do Algoritmo 14 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 14) e, dado que $A_I^\mathsf{T} Z_I = D_{II}$ (linha 4 do Algoritmo 14), concluímos que

$$A_I^\mathsf{T} Z_J^{(I)} = 0 \quad \forall\, I < J. \tag{37}$$

Agora, de acordo com o Lema 3, 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 (37) 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 (36a).

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 (36b).

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 (36a), e que o segundo somatório vale zero, de acordo com o Lema 2. Então (36c) segue do Lema 2 e da linha 4 do Algoritmo 14.

Notemos que as hipóteses (36b) e (36c) 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 2 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 (36d) 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 2 e 3.

4.3 BAINV para M-matrizes não singulares

Mesmo que A seja uma matriz esparsa, a matriz Z produzida pelos Algoritmos 14 e 15 pode ser densa. Como em [11], introduzimos no Algoritmo 14 o descarte de blocos (ou entradas) de Z, ver Definição 7. O método resultante, apresentado no Algoritmo 16, é chamado de BAINV (block approximate inverse).

Nesta seção, provaremos que o BAINV não quebra quando aplicado a M-matrizes. Para isso, inicialmente, precisaremos de algumas definições e propriedades desta classe de matrizes, estudada sistematicamente por Ostrowski [56].

Em [58], Plemmons apresentou 40 caracterizações equivalentes de M-matrizes não singulares. Nós usaremos três delas:

Lema 4. *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 [41], 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. *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 6. *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 7. *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 4 e do Lema 5. □

Lema 8. *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 4, $A^{-1} \geqslant 0$ e $B^{-1} \geqslant 0$. Usando o Lema 5, 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 9. *Se A for SPD, a matriz diagonal em blocos D produzida pelo Algoritmo 14 é SPD.*

Demonstração. O Teorema 1 garante que D é igual a $Z^{\mathsf{T}}AZ$ e que Z é não singular. □

Lema 10. *Seja A uma M-matriz SPD e D e Z as matrizes resultantes da aplicação do Algoritmo 14 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 14 e Lema 2, 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 9). Usando a Condição 1 do Lema 4, nós concluímos que D_{JJ} é uma M-matriz não singular. □

Agora apresentaremos o Algoritmo 16, que implementa o BAINV. A única diferença respeito do Algoritmo 14 é a linha 7, aonde são executados descartes para promover esparsidade. A seguir nós definimos tal procedimento.

Definição 7. *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.,*

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

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. Ainda sim, diversas 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, etc. Nossos resultados teóricos valem para qualquer estratégia de descarte empregada.

Algoritmo 16: BAINV right-looking 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 $\widetilde{Z} \leftarrow \left[\; \widetilde{Z}_1, \;\; \cdots \;\; , \widetilde{Z}_N \; \right]$;

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 11. *Se o Algoritmo 16 não quebrar, ele produz uma matriz bloco-triangular superior $\widetilde{Z}$. Além disso, sua bloco diagonal é formada por matrizes identidade.*

Demonstração. A prova é análoga à do Lema 2. 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 2. *Seja $A \in \mathcal{M}$ uma matriz em blocos $N \times N$, simétrica e não singular. Então o Algoritmo 16 não quebra, 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 $J \in \{1, \ldots, N\}$.

$$H_J : \begin{cases} 0 \leq \widetilde{Z}_I^{(J-1)} \leq Z_I^{(J-1)}, & I \in \{J, \ldots, N\}; \quad (38a) \\ R_I^{(J-1)} \leq \widetilde{R}_I^{(J-1)} \leq 0, & I \in \{J+1, \ldots, N\}; \quad (38b) \\ \widetilde{D}_{JJ} \geq D_{JJ}; & (38c) \\ \widetilde{D}_{JJ} \text{ é uma } M\text{-matriz não singular}; & (38d) \end{cases}$$

Enquanto a hipótese (38d) é, de fato, o resultado pretendido, as hipóteses (38a), (38b), e (38c) servem como suporte técnico para a demonstração.

Primeiro verifiquemos se H_1 é verdadeira. A linha 1 dos Algoritmos 14 e 16 mostram que $Z_I^{(0)} = \widetilde{Z}_I^{(0)} = E_I$ para $I = 1, \ldots, N$, o que garante (38a). Para provar (38b), obervemos que pelas linhas 1 e 6 do Algoritmo 16, $\widetilde{R}_I^{(0)} = A_{1I}$, $I = 2, \ldots, N$, enquanto pelas linhas 1 e 6 do Algoritmo 14, $R_I^{(0)} = A_{1I}$, $I = 2, \ldots, N$. A Equação (38b) 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 10), nós obtemos (38c) e (38d).

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

Para provarmos (38a), primeiro vejamos que a linha 7 dos Algoritmos 14 e 16 implica que

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

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

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 8 e pela condição 2 do Lema 4, $D^{-1}_{(J-1)(J-1)} \geq \widetilde{D}^{-1}_{(J-1)(J-1)} \geq 0$. Agora, pelo Lema 5 e pela definição de descarte, temos

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

provando (38a).

Para provar (38b), vejamos que pela linha 6 dos Algoritmos 14 e 16, junto com os Lemas 2 e 11, 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 (38a) , $Z_{LI}^{(J-1)} \geq \widetilde{Z}_{LI}^{(J-1)} \geq 0$. Com estes resultados e o Lema 5 provamos (38b).

Para provar (38c), vejamos que as linhas 4 dos Algoritmos 14 e 16 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, 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 (38d), vejamos primeiramente que o Lema 10 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 (38c) e pelo Lema 7, concluímos que $\widetilde{D}_{JJ}$ é uma M-matriz não singular. □

4.4 BAINV para H-matrizes não singulares

Nesta seção, provaremos que o Algoritmo 16 também produz matrizes não singulares quando aplicado a H-matrizes com matriz comparação não singular.

Lema 12. *Se A é uma H-matriz e $\mathcal{C}(A)$ é não singular então A é não singular. Ver [24].*

Na sequência, seguindo Plemmons em [58], consideraremos apenas H-matrizes que possuem M-matrizes não singulares como suas matrizes comparação.

Lema 13 (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 14. *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 [28].*

A seguir demonstraremos um dos principais resultados deste capítulo.

Teorema 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 16 não quebra, 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 14 aplicado a A e $\widetilde{D}_{JJ}$, $\widetilde{Z}_J^{(K)}$ e $\widetilde{R}_J^{(K)}$ para aquelas geradas pelo Algoritmo 16 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 16 sem descartes[1] aplicado à matriz comparação $\mathcal{C}(A)$. Finalmente, $\mathcal{C}(A)_{IJ}$ denota $(\mathcal{C}(A))_{IJ}$, não $\mathcal{C}(A_{IJ})$.

[1]Vejamos que não executar nenhum descarte, isto é, $descarte_I(V) = V$, é uma estratégia de descarte válida, de acordo com a equação (7). Vejamos, então, que o Algoritmo 16 com esta estratégia de descarte (sem executar descartes) equivale ao Algoritmo 14. Porém, preferimos mostrá-los separadamente, já que usaremos alguns resultados da Seção 4.3.

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

$$H_J : \begin{cases} |\widetilde{Z}_I^{(J-1)}| \leq \widehat{Z}_I^{(J-1)}, & I \in \{J, \ldots, N\}; \quad (39a) \\ |\widetilde{R}_I^{(J-1)}| \leq -\widehat{M}_I^{(J-1)}, & I \in \{J+1, \ldots, N\}; \quad (39b) \\ \widehat{D}_{JJ} \leq \mathcal{C}(\widetilde{D}_{JJ}); & (39c) \\ \widetilde{D}_{JJ} \text{ é uma } H\text{-matriz não singular.} & (39d) \end{cases}$$

A hipótese (39d) demonstra o resultado pretendido, enquanto as outras hipóteses servem como suporte técnico para a demonstração.

Primeiro verifiquemos se H_1 é verdadeira. A hipótese (39a) é verdadeira, tornando-se uma igualdade, para $J = 1$, pois $\widetilde{Z}_I^{(0)} = \widehat{Z}_I^{(0)} = E_I$, $1 \leq I \leq N$. Também a hipótese (39b) é verdadeira, tornando-se uma igualdade, para $J = 1$, pois para $I \geq 2$ we temos que $\mathcal{C}(A)_{1I} = -|A_{1I}|$ e, a partir das linhas 1 e 6 do Algoritmo 16, $\widetilde{R}_I^{(0)} = A_{1I}$ e $\widehat{M}_I^{(0)} = \mathcal{C}(A)_{1I}$. Da linha 4 do Algoritmo 16, $\widetilde{D}_{11} = A_{11}$ e $\widehat{D}_{11} = \mathcal{C}(A)_{11}$. Para provar que (39c) é verdadeira, como uma igualdade, note que $\widehat{D}_{11} = \big(\mathcal{C}(A)\big)_{11} = \mathcal{C}(A_{11}) = \mathcal{C}\left(\widetilde{D}_{11}\right)$. Provando a hipótese (39d): primeiramente, $\widetilde{D}_{11} = A_{11}$ é uma matriz não singular e, pelo Lema 14, é uma submatriz principal de uma matriz SPD. $\mathcal{C}(A)$ é M-matriz não 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 2). Pela hipótese de indução (39c), $\widehat{D}_{(J-1)(J-1)} \leq \mathcal{C}(\widetilde{D}_{(J-1)(J-1)})$, então, pelo Lema 7, $\mathcal{C}(\widetilde{D}_{(J-1)(J-1)})$ é M-matriz não singular. Pelo Lema 8, $\big(\mathcal{C}(\widetilde{D}_{(J-1)(J-1)})\big)^{-1} \leqslant \widehat{D}^{-1}_{(J-1)(J-1)}$, e pelo Lema 13, $\big|\widetilde{D}^{-1}_{(J-1)(J-1)}\big| \leq \big(\mathcal{C}(\widetilde{D}_{(J-1)(J-1)})\big)^{-1}$. Então

$$\big|\widetilde{D}^{-1}_{(J-1)(J-1)}\big| \leqslant \widehat{D}^{-1}_{(J-1)(J-1)}. \quad (40)$$

Como $|descarte_I(V)| \leq |V|$ para qualquer índice I e qualquer vetor-bloco V e considerando as hipóteses de indução (39a) e (39b) assim como a equação (40), podemos

provar (39a) ao examinarmos a linha 7 do Algoritmo 16:

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

Para provar (39b), 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 (39a) 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 (39c), devemos comparar os blocos

$$\widehat{D}_{JJ} = \mathcal{C}(A)_{JJ} + \sum_{I=1}^{J-1} \mathcal{C}(A)_{JI}\, \widehat{Z}_{IJ} \quad \text{e} \tag{41}$$

$$\mathcal{C}\left(\widetilde{D}_{JJ}\right) = \mathcal{C}\left(A_{JJ} + \sum_{I=1}^{J-1} A_{JI}\widetilde{Z}_{IJ}\right). \tag{42}$$

Primeiramente percebamos que, usando $|A_{JI}| = -\mathcal{C}(A)_{JI}$ e a hipótese de indução (39a), o Lema 6 garante que

$$\mathcal{C}(A)_{JI}\, \widehat{Z}_{IJ} \leq A_{JI}\widetilde{Z}_{IJ} \leq -\mathcal{C}(A)_{JI}\, \widehat{Z}_{IJ}. \tag{43}$$

Agora comparemos as entradas da diagonal principal. Como $\mathcal{C}(A)$ é uma M-matriz não singular, (38d) 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 4. Com isso temos:

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

Para as entradas fora da diagonal, primeiro vejamos que, pela definição de matriz comparação, $\left(\mathcal{C}\left(A\right)_{JJ}\right)_{ij} \leq \left(A_{JJ}\right)_{ij} \leq -\left(\mathcal{C}\left(A\right)_{JJ}\right)_{ij}$ para $i \neq j$. Junto com as desigualdades em (43) e as fórmulas (41) e (42), 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 (39c).

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 (38d) $\widehat{D}_{JJ}$ M-matriz não singular, o Lema 7 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 12 garante que $\widetilde{D}_{JJ}$ é não singular e, então, (39d) é provado.

□

5 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 quebra para matrizes positivas definidas. Além dessa, também propomos versões em blocos para o FFAPINV, SAINV-VAR e RIF-NS. No final do capítulo, exibimos alguns resultados obtidos comparando as adaptações em blocos do SAINV e SAINV-VAR, em testes numéricos executados.

5.1 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 $A^{-1} = ZD^{-1}W$, onde 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, $W_I * AZ_J = 0$ 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 17 e 18 que são as versões right-looking e left-looking, 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 de forma análoga à versão escalar, no Capítulo 1. Para demonstrar os resultados deste capítulo, nos basearemos apenas na versão right-looking, já que para a versão left-looking pode ser feita de forma análoga. Vejamos alguns resultados a seguir.

Lema 15. *Caso o Algoritmo 14 não quebre (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 para provar que, para $I = 0, 1, \ldots, N-1$,

$$E_L^T Z_J^{(I)} = W_{J*}^{(I)} E_L = \delta_{LJ} \quad \forall\, J, L \ \text{ tal que } \ I < J \leq L \leq N, \tag{44}$$

Algoritmo 17: A-biconjugação em blocos right-looking

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**
    $R_J^{(I-1)} \leftarrow A_{I*}Z_J^{(I-1)}$;
    $S_J^{(I-1)} \leftarrow W_{J*}^{(I-1)}A_I$;
    $Z_J^{(I)} \leftarrow Z_J^{(I-1)} - Z_I D_{II}^{-1} R_J^{(I-1)}$;
    $W_{J*}^{(I)} \leftarrow W_{J*}^{(I-1)} - S_J^{(I-1)} D_{II}^{-1} W_{I*}$;

$D \leftarrow \text{diag}(D_{11}, \ldots, D_{NN})$, $Z \leftarrow \begin{bmatrix} Z_1, & \cdots & , Z_N \end{bmatrix}$ e $W \leftarrow \begin{bmatrix} W_{1*} \\ \vdots \\ W_{N*} \end{bmatrix}$;

Algoritmo 18: A-biconjugação em blocos left-looking

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**
    $R_J^{(I-1)} \leftarrow A_{I*}Z_J^{(I-1)}$;
    $S_J^{(I-1)} \leftarrow W_{J*}^{(I-1)}A_I$;
    $Z_J^{(I)} \leftarrow Z_J^{(I-1)} - Z_I D_{II}^{-1} R_J^{(I-1)}$;
    $W_{J*}^{(I)} \leftarrow W_{J*}^{(I-1)} - S_J^{(I-1)} D_{II}^{-1} W_{I*}$;
  $Z_J \leftarrow Z_J^{(J-1)}$; $W_{J*} \leftarrow W_{J*}^{(J-1)}$
  $D_{JJ} \leftarrow A_{J*}Z_J$;

$D \leftarrow \text{diag}(D_{11}, \ldots, D_{NN})$, $Z \leftarrow \begin{bmatrix} Z_1, & \cdots & , Z_N \end{bmatrix}$ and $W \leftarrow \begin{bmatrix} W_{1*} \\ \vdots \\ W_{N*} \end{bmatrix}$;

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 (34) é notavelmente verdadeira para $I = 0$, já que $Z_J^{(0)} = E_J$ e $W_{J*}^{(0)} = E_J^T$. Agora, assumamos que (44) é verdadeira para $0 \leq I < N - 1$. A partir disso e da linha 9 do Algoritmo 17,

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

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 16. *Se o Algoritmo 17 não quebrar, temos que, para qualquer* $1 \leq J \leq N$ *fixo,*

$$Z_J = Z_J^{(K)} - \sum_{I=K+1}^{J-1} Z_I D_{II}^{-1} R_J^{(I-1)}, \qquad W_{J*} = W_{J*}^{(K)} - \sum_{I=K+1}^{J-1} S_J^{(I-1)} D_{II}^{-1} W_{I*} \tag{45}$$

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

Demonstração. Segue diretamente das linhas 4, 9 e 10 do Algoritmo 17. □

A seguir demonstramos, no Teorema 4, que sob determinadas condições, o Algoritmo 17 realmente produz a decomposição $A^{-1} = ZD^{-1}W$.

Teorema 4. *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 17 não quebra 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 $J \in \{1, \ldots, N\}$.

$$H_J : \begin{cases} A_{I*}Z_J = 0, & I \in \{1, 2, \ldots, J-1\}; \quad (46a) \\ W_{J*}A_I = 0, & I \in \{1, 2, \ldots, J-1\}; \quad (46b) \\ W_{I*}AZ_J = W_{J*}AZ_I = 0, & I \in \{1, 2, \ldots, J-1\}; \quad (46c) \\ W_{J*}AZ_J = D_{JJ}; & (46d) \\ W_{J*}A_J = D_{JJ}; & (46e) \\ D_{JJ} \text{ é não singular}; & (46f) \end{cases}$$

As hipóteses (46c), (46d), e (46f) são suficientes para chegar no resultado desejado. Já as hipóteses (46a), (46b), e (46e) servem como suporte técnico para a demonstração.

Para $J = 1$, as condições (46a), (46b) e (46c) são alcançadas por vacuidade e as condições (46d), (46e) e (46f) 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 (46a), nós primeiramente multiplicamos a linha 9 do Algorimto 17 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 17) e, sendo $A_{I*}Z_I = D_{II}$ (linha 5 do Algoritmo 17), nós concluímos que

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

Agora, a partir do Lema 16, 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 (46a).

Analogamente, para provar (46b) primeiramente multiplicamos do lado direito a

linha 10 do Algoritmo 17 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 17) e, sendo $W_{I*}A_I = D_{II}$ (pelas hipóteses de indução (46e)), concluímos que

$$W_{J*}^{(I)} A_I = 0 \quad \forall\, I < J. \tag{47}$$

Agora, a partir do Lema 16, 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 (47) 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 (46b).

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 (46c).

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

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

Notemos que o primeiro somatório do último termo é igual a zero por conta da hipótese de indução (46a) e que o segundo somatório é zero de acordo com o Lema 15. Então chegamos ao (46d) pelo Lema 15 e linha 5 do Algoritmo 17.

A partir de (46d), Lema 15 e linha 1 do Algoritmo 17 nós temos

$$D_{JJ} = W_{J*}AZ_J = W_{J*}AE_J - \sum_{I=1}^{J-1} W_{J*}AZ_I D_{II}^{-1} R_J^{(I-1)}. \tag{48}$$

O somatório no último termo é zero por conta de (46c). Portanto, temos $D_{JJ} = W_{J*}A_J$, provando (46e).

A hipóteses (46c) e (46d) resultam que

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

uma matriz bloco-diagonal com D_{II}, $1 \leqslant I \leqslant J$, na sua diagonal em blocos. Além disso, o Lema 15 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 $A_{1:J:,1:J}$, por hipótese, é também não singular, chegamos a (46f).

□

No Corolário 1 temos mais uma opção para o cálculo dos blocos D_{II} do Teorema 4.

Corolário 1. *Considerando os elementos do Algoritmo 17, temos que* $D_{II} = W_{I*}A_I$.

O resultado a seguir verifica que as relações (12) e (14) 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 17 aplicado a A temos*

$$R_J^{(K-1)} = W_{K*}A_J, \tag{49}$$

$$S_J^{(K-1)} = A_{J*}Z_K. \tag{50}$$

Demonstração. Pelas linhas 1, 2, 9 e 10 do Algoritmo 17, 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 $K = 1, ..., J-1$, temos

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

Analogamente, temos:

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

obtendo (50). □

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, onde 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. Consideremos, por absurdo, que a fatoração bloco LDU de A não seja única e, portanto, existam duas fatorações distintas LDU e $L'D'U'$. Então,

$$LDU = L'D'U' \Rightarrow (L')^{-1}LD = D'U'U^{-1} = B$$

Como $(L')^{-1}$ e L são bloco triangulares inferior com identidades nas diagonais e D é bloco

diagonal, então B é bloco triangular inferior tal que $R_{II} = D_{II}$, $1 \leq I \leq N$. Também temos que, como $U^{'}$ e U^{-1} são bloco triangulares superior com identidades nas diagonais e $D^{'}$ é bloco diagonal, então B é bloco superior tal que $R_{II} = D^{'}_{II}$, $1 \leq I \leq N$. Portanto, $B = D = D^{'}$. Com isso, $(L^{'})^{-1}LD = D$ e $D^{'}U^{'}U^{-1} = D^{'}$ e, assim, $L = L^{'}$ e $U = U^{'}$, contradizendo a hipótese da fatoração não ser única. □

Consideremos a decomposição

$$A = W^{-1}DZ^{-1}, \tag{51}$$

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 (51) pode ser considerada a decomposição bloco LDU de A de tamanho t. Como essa decomposição é única (Proposição 8), temos que:

$$W = L^{-1}, \quad Z = U^{-1} \tag{52}$$

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, onde 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 14 quando aplicado a A, sem os descartes, produz*

$$U_{IJ} = D_{II}^{-1}R_J^{(I-1)}, \tag{53}$$

$$L_{JI} = S_J^{(I-1)}D_{II}^{-1}, \tag{54}$$

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

Demonstração. De acordo com (51), para $0 \leq I < J \leq N$,

$$U = Z^{-1} = D^{-1}WA \Rightarrow U_{IJ} = D_{II}^{-1}W_{I*}A_J,$$

e

$$W^{-1} = L = AZD^{-1} \Rightarrow L_{JI} = A_{J*}Z_ID_{II}^{-1}.$$

Considerando (49) e (50) , temos que $U_{IJ} = D_{II}^{-1}R_J^{(I-1)}$ e $L_{JI} = S_J^{(I-1)}D_{II}^{-1}$. □

Proposição 10. *Seja A uma matriz em blocos $N \times N$ esparsa e não singular. O Algoritmo 14 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. Usando as mesmas hipóteses da Proposição 9, a partir da fatoração LDU em blocos de A, temos, para $0 \leq I \leq J \leq N$,

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

A partir de (53), (54) e de que $U_{II} = I$,

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

Analogamente, usando as mesmas hipóteses da Proposição 9, a partir da fatoração LDU em blocos de A, temos, para $0 \leq I \leq J \leq N$,

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

A partir de (53), (54) e de que $L_{II} = I$,

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

□

Mesmo que A seja esparsa, as matrizes Z e W produzidas pelos Algoritmos 17 e

18 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 quebra seria utilizar as expressões

$$(Z_I)^T A Z_I \tag{55}$$

ou

$$W_{I*}A(W_{I*})^T \tag{56}$$

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 quebra em matrizes positivas definidas.

5.2 Trabalhos desenvolvidos

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

5.2.1 BAINV-NS

Em [22], de 2000, Bridson e Tang apresentaram uma versão do SAINV-NS left-looking para quaisquer matrizes não singulares em blocos, chamada de BAINV-NS ("Nonsymmetric Block AINV"). 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 [51].

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 (49), (50) e $D_{JJ} = W_{J*}AZ_J$. Podemos ver esse processo no Algoritmo 19.

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

Algoritmo 19: 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**
    $R_J^{(I-1)} \leftarrow W_{I*}A_J$;
    $S_J^{(I-1)} \leftarrow A_{J*}Z_I$;
    $Z_J^{(I)} \leftarrow descarte_I(Z_J^{(I-1)} - Z_I D_{II}^{-1} R_J^{(I-1)})$;
    $W_{J*}^{(I)} \leftarrow descarte_I(W_{J*}^{(I-1)} - S_J^{(I-1)} D_{II}^{-1} W_{I*})$;
  $Z_J \leftarrow Z_J^{(J-1)}$; $W_{J*} \leftarrow W_{J*}^{(J-1)}$
  $D_{JJ} \leftarrow W_{J*}AZ_J$;

$$D \leftarrow \mathrm{diag}(D_{11}, \ldots, D_{NN}) \text{ , } Z \leftarrow \left[\begin{array}{ccc} Z_1, & \cdots & , Z_N \end{array} \right] \text{ and } W \leftarrow \left[\begin{array}{c} W_{1*} \\ \vdots \\ W_{N*} \end{array} \right];$$

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 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, ele não obteve bons desempenhos, diante da estratégia de descarte que descartava blocos inteiros, sendo mais provável de se obter escolhas ruins.

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 14 NS-SBAINV

5.2.2 SBAINV

Em 2001, Benzi, Kouhia e Tůma apresentaram o BAINV ("Block AINV"), em [11], que é uma versão em bloco do AINV right-looking, para matrizes SPD. A partir desse, eles apresentaram o SBAINV ("Stabilized Block AINV") como um versão estável para matrizes SPD, baseando-se no SAINV, pois utiliza relação $D_{II} = Z_I^T A Z_I$ (equação (55)) no cálculo dos pivôs. Os autores salientaram que, a estrutura em blocos, pode surgir, tanto naturalmente, da discretização do problema real, como por exemplo, quando usamos o método dos elementos finitos para discretizar uma equação diferencial parcial, quanto artificialmente, caso melhore o desempenho. O método proposto é exibido no Algoritmo 16 no Capítulo 4.

Para o BAINV, as inversas dos pivôs no BAINV foram computadas aplicando fatoração triangular. Por causa do descarte, os pivôs podem não ser SPD, e, para isso, é feita primeiramente uma fatoração LU do pivô e, posteriormente ele é computado como LL^T, para deixá-lo simétrico. Se necessário, algumas modificações no pivô foram ser feitas para que ele se tornasse SPD. Com o SBAINV eles não tiveram essa preocupação por ser garantidamente livre de quebra 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 de Bloco Jacobi. O descarte utilizado foi de tolerância fixa ou tolerância relativa, podendo variar de acordo com algum aspecto do problema, por exemplo, a norma da linha i de A vezes algum valor. 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 comparam 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 que o SAINV-NS. Na Tabela 14 exibimos um resumo das principais características deste trabalho. Na Tabela 15 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 15 BAINV

5.3 Variações propostas

Nesta seção, apresentamos algumas adaptações para a inversa aproximada em blocos, algumas delas baseadas nas versões escalares do AINV exibidas no Capítulo 2.

5.3.1 SBAINV-NS

Aqui 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 17, e incluímos a possibilidade de calcular o pivô como nas equações (55) e (56), se A for positiva definida. Por isso chamamos o método de SBAINV-NS ("Stabilized Block AINV-Nonsymmetric"), exibido nos Algoritmos 20 e 21, nas versões right-looking e left-looking, respectivamente.

Algoritmo 20: SBAINV-NS right-looking

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**
    $M_J^{(I-1)} \leftarrow A_{I*}Z_J^{(I-1)}$;
    $S_J^{(I-1)} \leftarrow W_{J*}^{(I-1)}A_I$;
    $Z_J^{(I)} \leftarrow descarte_I(Z_J^{(I-1)} - Z_I D_{II}^{-1} M_J^{(I-1)})$;
    $W_{J*}^{(I)} \leftarrow descarte_I(W_{J*}^{(I-1)} - S_J^{(I-1)} D_{II}^{-1} W_{I*})$;

$D \leftarrow \mathrm{diag}(D_{11}, \ldots, D_{NN})$, $Z \leftarrow \begin{bmatrix} Z_1, & \cdots & , Z_N \end{bmatrix}$ e $W \leftarrow \begin{bmatrix} W_{1*} \\ \vdots \\ W_{N*} \end{bmatrix}$;

Lema 17. *Caso o Algoritmo 20 não quebre, 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 15. As únicas observações adicionais são

Algoritmo 21: SBAINV-NS left-looking

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;

$D \leftarrow diag(D_{11}, \ldots, D_{NN})$, $Z \leftarrow \left[\; Z_1, \;\; \cdots \;\; , Z_N \; \right]$ and $W \leftarrow \begin{bmatrix} W_{1*} \\ \vdots \\ W_{N*} \end{bmatrix}$;

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 20 não quebrar 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.

5.3.2 BFFAPINV

Nesta seção apresentamos a versão em blocos do algoritmo FFAPINV [66]. Utilizando as relações (49) e (50) propomos o 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 22. Vejamos que, como no caso escalar, ele é exibido baseando-se na versão left-looking do algoritmo de A-biconjugação.

Algoritmo 22: 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**
    $M_J^{(I-1)} \leftarrow W_{I*}A_J$;
    $S_J^{(I-1)} \leftarrow A_{J*}Z_I$;
    $Z_J^{(I)} \leftarrow descarte_I(Z_J^{(I-1)} - Z_I D_{II}^{-1} M_J^{(I-1)})$;
    $W_{J*}^{(I)} \leftarrow descarte_I(W_{J*}^{(I-1)} - S_J^{(I-1)} D_{II}^{-1} W_{I*})$;
  $Z_J \leftarrow Z_J^{(J-1)}$; $W_{J*} \leftarrow W_{J*}^{(J-1)}$;
  $D_{JJ} \leftarrow W_{J*}AZ_J$;

$D \leftarrow diag(D_{11}, \ldots, D_{NN})$, $Z \leftarrow \begin{bmatrix} Z_1, & \cdots & , Z_N \end{bmatrix}$ e $W \leftarrow \begin{bmatrix} W_{1*} \\ \vdots \\ W_{N*} \end{bmatrix}$;

5.3.3 SBAINV-VAR

Nesta seção apresentamos a versão em blocos do algoritmo SAINV-VAR [59], o SBAINV-VAR (Block SAINV-VAR), descrito no Algoritmo 23. 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 (54). Por sua vez, para se calcular $S_J^{(I-1)}$ é utilizada a Proposição 10 e a equação (54), 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 quebra em matrizes positivas definidas.

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 8. *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]):

Algoritmo 23: 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**
    $M_J^{(I-1)} \leftarrow A_{I*}Z_J^{(I-1)}$;
    $S_J^{(I-1)} \leftarrow A_{JI} - \sum_{K=1}^{I-1} L_{JK} M_I^{(K-1)}$;
    $L_{JI} \leftarrow \text{descarte}(S_J^{(I-1)} D_{II}^{-1})$;
    $Z_J^{(I)} \leftarrow \text{descarte}_I(Z_J^{(I-1)} - Z_I D_{II}^{-1} M_J^{(I-1)})$;
$D \leftarrow diag(D_{11}, \ldots, D_{NN})$, $Z \leftarrow \left[\; Z_1, \;\; \cdots \;\; , Z_N \; \right]$ e $L \leftarrow [L_{JI}]$;

$$W_l = I + F + F^2 + F^3 + ... + F^l, \tag{57}$$

onde $F = I - L$ e I é a identidade. Por meio de (57) e do método de Horner [40], obtemos a inversa aproximada $A^{-1} \approx ZD^{-1}W_l$. A ideia é usar a fórmula de Horner [40] 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 24.

Algoritmo 24: 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.

5.3.4 BRIF-NS

Nesta seção apresentamos a versão em blocos do algoritmo RIF-NS [60], chamada de 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 51, caso A for positiva definida. As entradas das aproximações de L e U são obtidas através das equações (54) e (53) e as matrizes R_J's calculadas com base na Proposição 10. No Algoritmo 25 vemos a execução do BRIF-NS.

Algoritmo 25: BRIF-NS right-looking

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 $A \approx LDU$.

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

5.4 Experimentos Numéricos

Vamos analisar os comportamentos do SBAINV-NS, proposto na Seção 5.3.1, e do SBAINV-VAR, apresentado na Seção 5.3.3. Escolhemos apenas esse dois métodos, pois o consideramos o BFFAPINV muito semelhante ao BAINV-NS, já proposto na literatura. Também não optamos em testar o BRIF-NS, pois estamos interessados em comparar

apenas os fatores da inversa aproximada de A, diferente do BRIF-NS, que computa os fatores LDU da aproximação de A.

Os algoritmos foram implementados em Python 3.8, com o auxílio das bibliotecas NumPy e SciPy, e da biblioteca Matplotlib para criar os gráficos. Os experimentos foram simulados em um computador com 2 $\times$ CPU i5-7200U, de 2,5 GHz, e memória RAM de 8GB, com o sistema operacional Windows 64. As matrizes utilizadas pertencem aos repositórios Matrix Market[2] e Tim Davis' Collection[3]. Estas matrizes são provenientes de simulações de reservatórios de petróleo (SHERMAN1 e SHERMAN4), de um modelo proposto por H. Elman usando equações diferenciais parciais (PDE900 e PDE2961), de problemas de fluxo de rede (HOR131) e de problemas de eletroforese de DNA (CAGE8). Algumas destas matrizes possuem uma estrutura de blocos determinada pelo problema físico subjacente; no entanto, estabelecemos divisões arbitrárias para construir blocos homogêneos e avaliar o impacto da abordagem em blocos (b2 e b3) em relação à escalar (b1). A Tabela 16 mostra a ordem n de cada uma das matrizes, as respectivas quantidades de elementos não nulos nnz, os tamanhos dos blocos homogêneos utilizados nos experimentos com cada matriz e o número médio de elementos não nulos por linha (nnz/n) em cada uma delas.

Criamos um conjunto de dez vetores aleatórios b_i, $i = 1, ..., 10$, com entradas uniformemente distribuídas entre $[0, 1)$, para a solução do sistema $Ax = b_i$ por meio do método BICGSTAB (nativo da biblioteca SciPy) precondicionado por SBAINV-NS e SBAINV-VAR. Com isso, simulamos os testes com o método iterativo dez vezes para cada matriz. Assim, os números de iterações indicados nas tabelas são as médias das iterações obtidas nos dez testes realizados. O critério de parada foi 10^{-6} como cota superior para a norma do resíduo relativo ao lado direito, sendo o valor inicial da solução sempre zero e o número máximo de iterações igual a 1000. Nos experimentos do SBAINV-VAR, foram usados os primeiros quatro termos da série de Neumann para aproximar a inversa de W.

A norma de Frobenius foi utilizada para decidir quais blocos seriam descartados. No SBAINV-NS, um bloco de $Z_J^{(I)}$ ou de $W_{J*}^{(I)}$ (ver Algoritmo 14) foi descartado caso a sua norma tenha sido menor que um valor determinado. No SBAINV-VAR, o mesmo procedimento foi executado para $Z_J^{(I)}$. Para a matriz L, o bloco L_{JI} foi descartado caso sua norma fosse menor que um valor escolhido. Utilizamos a mesma tolerância para os

[2]https://math.nist.gov/MatrixMarket.

[3]https://www.cise.ufl.edu/research/sparse/matrices.

Tabela 16 Ordens e número de elementos não nulos das matrizes e ordem dos blocos homogêneos (b1, b2 e b3).

Matriz	n	nnz	nnz $/n$	b1	b2	b3
SHERMAN1	1000	3750	3,75	1	2	8
SHERMAN4	1104	3786	3,43	1	2	6
PDE900	900	4380	4,87	1	2	6
PDE2961	2961	14585	4,93	1	3	7
HOR131	434	4710	10,85	1	2	7
CAGE8	1015	11003	10,84	1	5	7

descartes em $Z_J^{(I)}$ e em L_{JI}. Os valores usados para os descartes nos testes foram de 0,1, 0,2, 0,3, 0,4 e 0,5.

As densidades dos precondicionadores SBAINV-NS e SBAINV-VAR foram medidas, respectivamente, pelas equações (58) e (59):

$$\delta_{\text{NS}} = \frac{\text{nnz}(Z) + \text{nnz}(W) + \text{nnz}(D)}{\text{nnz}(A)} \tag{58}$$

e

$$\delta_{\text{VAR}} = \frac{\text{nnz}(Z) + \text{nnz}(L) + \text{nnz}(D)}{\text{nnz}(A)}, \tag{59}$$

onde nnz(X) é o número de elementos não nulos da matriz X. Incluímos o número de não zeros da matriz D, pois, no caso bloco, quanto maior for o tamanho do bloco, maior será o preenchimento de D, uma vez que não há descarte nestes blocos.

O número médio, mediana e desvio padrão dos números de iterações obtidos pelo BICGSTAB sem precondicionamento com os dez lados direitos aleatórios associados a cada matriz são exibidos na Tabela 17. O símbolo † indica que o sistema não reduziu o resíduo, na tolerância requerida, dentro do número máximo de iterações. Diante da proximidade entre os valores da média e mediana para cada matriz e do baixo desvio padrão em relação a esses valores, nós utilizaremos as médias apresentadas como base de comparação com os outros testes.

Tabela 17 Dados dos números de iterações do BICGSTAB sem precondicionamento.

Matriz	Iterações		
	Média	Mediana	Desvio Padrão
SHERMAN1	350,7	355,5	18,97
SHERMAN4	80,2	80,5	2,52
PDE900	70,9	70,5	1,22
PDE2961	132,7	133	1,95
HOR131	†	†	†
CAGE8	10,9	11	0,54

5.4.1 Variando blocos e descartes

Analisaremos os algoritmos de acordo com o número de iterações e densidades dos precondicionadores, ao variarmos os parâmetros. A Tabela 18 mostra o impacto da variação dos descartes e do tamanho dos blocos no número médio de iterações do BICGSTAB, para ambos os precondicionadores. Na última linha desta tabela, temos o quociente dos valores do número médio de iterações do SBAINV-VAR para o parâmetro de descarte de $0,5$ e $0,1$, para cada tamanho de bloco. E na última coluna temos o quociente dos valores do número médio de iterações para os blocos de tamanho $b3$ e $b1$. A Tabela 19 apresenta dados para os mesmos parâmetros e suas influências nas densidades dos precondicionadores. Estes números são relativos à matriz SHERMAN1. Na última linha desta tabela, temos o quociente dos valores das densidades dos SBAINV-VAR para o parâmetro de descarte de $0,5$ e $0,1$, para cada tamanho de bloco. E na última coluna temos o quociente dos valores das densidades para os blocos de tamanho $b3$ e $b1$. As demais matrizes tiveram comportamentos semelhantes.

A Tabela 20 apresenta as razões entre as iterações dos precondicionadores (SBAINV-NS no numerador e SBAINV-VAR no denominador) na aplicação do BICGSTAB precondicionado, para diversos tamanhos de bloco, para várias tolerâncias de descarte, com a matriz SHERMAN1. Acima de 1 o BICGSTAB precondicionado pelo SBAINV-VAR faz menos iterações. Ou seja, é a razão entre as linhas da Tabela 18, relativas a cada tolerância de descarte. Da mesma forma, a Tabela 21 apresenta as razões entre as densidades dos precondicionadores (SBAINV-NS no numerador e SBAINV-VAR no denominador) na aplicação do BICGSTAB precondicionado, para diversos tamanhos de bloco, para várias

tolerâncias de descarte, com a matriz SHERMAN1. Acima de 1 o SBAINV-VAR é menos denso. Ou seja, é a razão entre as linhas da Tabela 19, relativas a cada tolerância de descarte.

A partir da Tabela 18 e da Tabela 20, temos que o SBAINV-VAR realiza menos iterações em todos os casos estudados, com uma média de 33,3% menos iterações. Combinado a este fato, a partir das Tabela 19 e da Tabela 21, vemos que o SBAINV-NS tende a ficar mais denso do que o SBAINV-VAR com o aumento do tamanho dos blocos, basta ver a linha Média na Tabela 21. Considerando as médias da coluna b3/b1 da Tabela 19, para ambos os precondicionadores, temos que o SBAINV-NS tem um crescimento médio na sua densidade de 6,89 quando aumentamos o bloco, enquanto o SBAINV-VAR tem de 5,39. Ou seja, o SBAINV-VAR é melhor nos dois parâmetros estabelecidos para a análise, já que fornece um menor número de iterações e menor densidade de precondicionador, em relação ao SBAINV-NS. O mesmo comportamento foi observado para todas as demais matrizes testadas, como se confirmará na Subseção 5.4.2.

Uma segunda observação em relação ao SBAINV-VAR, a partir da coluna b3/b1 da Tabela 18, é que o aumento do tamanho do bloco diminui o número médio de iterações entre 31% e 56%. Além disso, a coluna b3/b1 da Tabela 19, mostra um aumento na densidade de quase 8 vezes para o descarte 0,1 e de quase 4 vezes para o descarte 0,5, quando aumentamos a ordem do bloco homogêneo. Para matrizes esparsas, o aumento de densidade para o descarte 0,1 em relação ao 0,5, significando maior utilização de memória, pode ser compensado pela diminuição em média de 35% do número de iterações do método de Krylov. Isso pode indicar maior vantagem ao se utilizar o valor de $0,1$ como tolerância de descarte. Esta hipótese terá que ser confirmada em testes com linguagens compiladas, como o C++, e em ambientes paralelos, onde o AINV tem maiores chances de ter um bom desempenho [12].

Uma outra indicação para o uso do descarte 0,1, aparece na análise das linhas 0,5/0,1 das Tabela 18 e Tabela 19. Nelas, vemos que há um aumento na razão entre o número de iterações do SBAINV-VAR com o aumento da ordem do bloco, assim como uma diminuição na razão entre as densidades respectivamente. Porém, blocos densos em matrizes esparsas tendem a induzir uma melhor utilização das memórias rápidas nas arquiteturas usuais de CPU's e GPU's, como pode ser visto em [2]. A partir destas observações, focaremos nosso estudo para a tolerância de descarte de 0,1.

Tabela 18 Número médio de iterações obtidos na aplicação do BICGSTAB com os precondicionadores SBAINV-NS e SBAINV-VAR na matriz SHERMAN1, com variação dos descartes.

Tolerância	Precondicionador	b1	b2	b3	b3/b1
0,1	SBAINV-NS	30,1	18,7	13,1	0,44
	SBAINV-VAR	25,8	15,8	11,3	0,44
0,2	SBAINV-NS	34,2	27,7	19,5	0,57
	SBAINV-VAR	28,6	23,8	15,7	0,55
0,3	SBAINV-NS	38,9	30,3	24,8	0,64
	SBAINV-VAR	30,6	24,2	19,2	0,63
0,4	SBAINV-NS	46,3	44,9	28,4	0,61
	SBAINV-VAR	35,9	29,2	22,3	0,62
0,5	SBAINV-NS	63,2	47,9	40,5	0,64
	SBAINV-VAR	41,0	30,6	28,3	0,69
0,5/0,1	SBAINV-VAR	1,59	1,94	2,5	

Tabela 19 Densidade dos precondicionadores (ver fórmulas (58) e (59)) SBAINV-NS e SBAINV-VAR para a matriz SHERMAN1, com variação dos descartes.

Tolerância	Precondicionador	b1	b2	b3	b3/b1
0,1	SBAINV-NS	2,05	3,70	21,75	10,61
	SBAINV-VAR	1,82	2,97	14,52	7.98
0,2	SBAINV-NS	1,50	2,12	11,17	7,45
	SBAINV-VAR	1,52	2,05	8,62	5,67
0,3	SBAINV-NS	1,33	1,69	8,10	6,09
	SBAINV-VAR	1,43	1,84	6,98	4,88
0,4	SBAINV-NS	1,18	1,48	6,50	5,51
	SBAINV-VAR	1,35	1,73	6,09	4.51
0,5	SBAINV-NS	1,04	1,28	5,00	4.81
	SBAINV-VAR	1,28	1,63	5,02	3.92
0,5/0,1	SBAINV-VAR	0,70	0,55	0,35	

Tabela 20 Razões entre as iterações dos precondicionadores (SBAINV-NS/SBAINV-VAR) na aplicação do BICGSTAB precondicionado, com a matriz SHERMAN1, com variação de descartes.

Matriz	b1	b2	b3
0,1	1,17	1,18	1,16
0,2	1,20	1,16	1,24
0,3	1,27	1,25	1,29
0,4	1,29	1,54	1,27
0,5	1,54	1,57	1,43
Média	1,29	1,34	1,28

Tabela 21 Razões entre as densidades dos precondicionadores (SBAINV-NS/SBAINV-VAR) na aplicação do BICGSTAB precondicionado, com a matriz SHERMAN1, com variação de descartes.

Matriz	b1	b2	b3
0,1	1,13	1,25	1,50
0,2	0,99	1,03	1,30
0,3	0,93	0,92	1,16
0,4	0,87	0,86	1,07
0,5	0,81	0,79	1,00
Média	0,95	0,97	1,20

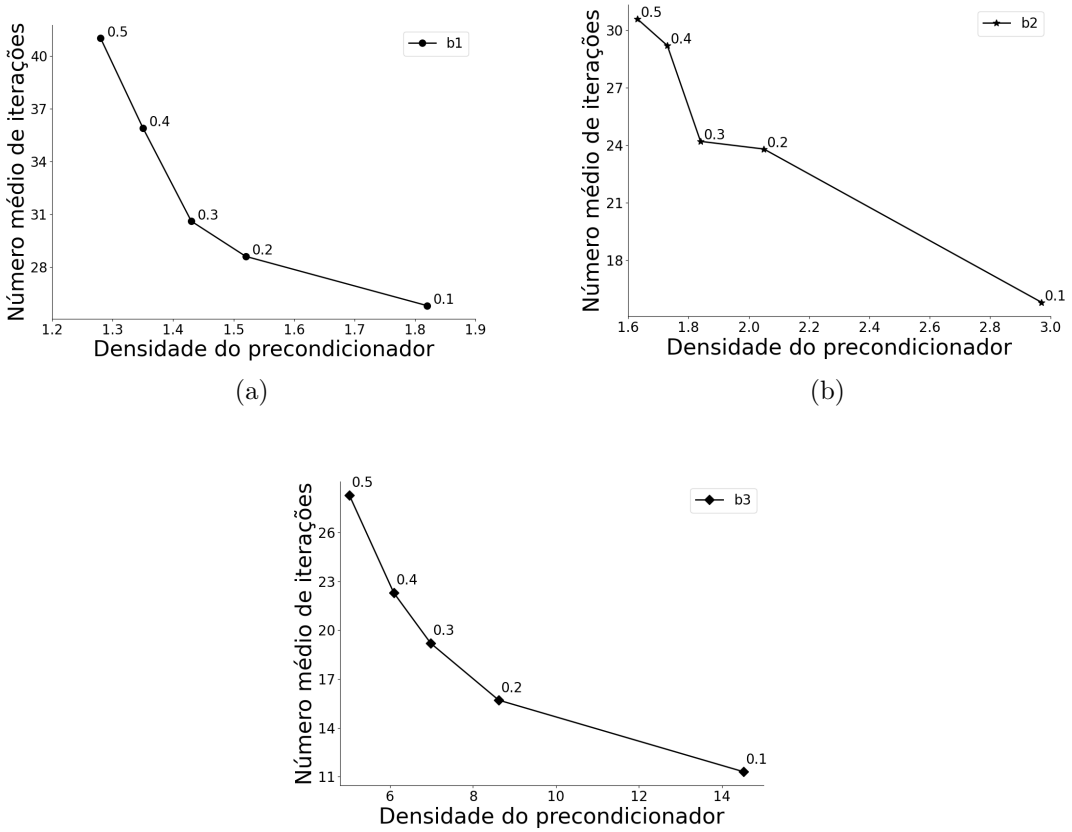

Figura 5 Relação entre a densidade do SBAINV-VAR e o número médio de iterações do BICGSTAB precondicionado, com variação dos descartes, para a matriz SHERMAN1.

A Figura Figura 5 mostra a relação entre densidade e número médio de iterações quando aumentamos a tolerância nos diversos tamanhos de bloco. Chama atenção que há um aumento importante da densidade para todos os blocos, mas de forma mais notável no b3. Esta constatação nos indica que outros tipos de descartes devem ser testados para se tentar controlar melhor este crescimento, sendo esta uma indicação para trabalhos futuros.

5.4.2 Variando blocos e mantendo descarte constante

Utilizando as matrizes descritas na Tabela 16, a Tabela 22 mostra o número médio de iterações do BICGSTAB para os dois precondicionadores quando há variação de tamanho dos blocos homogêneos. Na última coluna desta tabela temos o quociente dos valores do número médio de iterações para os blocos de tamanho $b3$ e $b1$. Na Tabela 23, apresentamos as densidades dos precondicionadores em relação aos tamanhos dos blocos.

Na última coluna desta tabela temos o quociente dos valores das densidades para os blocos de tamanho $b3$ e $b1$. Nestes testes, a tolerância para o descarte foi fixada em 0,1, seguindo as observações realizadas na Subeção 5.4.1.

Tabela 22 Número médio de iterações na aplicação do BICGSTAB precondicionado por SBAINV-NS e SBAINV-VAR, para diversos tamanhos de bloco, com tolerância de descarte de 0,1.

Matriz	Precondicionador	b1	b2	b3	b3/b1
SHERMAN1	SBAINV-NS	30,1	18,7	13,1	0,44
	SBAINV-VAR	25,8	15,8	11,3	0,44
SHERMAN4	SBAINV-NS	30,8	24,2	18,3	0,59
	SBAINV-VAR	24,6	20,7	15,1	0,61
PDE900	SBAINV-NS	19,4	12,7	7,0	0,36
	SBAINV-VAR	15,2	9	5,2	0,34
PDE2961	SBAINV-NS	41,3	17,8	12,2	0,30
	SBAINV-VAR	31,1	15,6	11,8	0,38
HOR131	SBAINV-NS	26,7	32	9,6	0,36
	SBAINV-VAR	24,7	20,2	10,1	0,41
CAGE8	SBAINV-NS	6,1	4,6	4,0	0,66
	SBAINV-VAR	5,5	4,0	4,0	0,73

Na Tabela 24, utilizando os dados da Tabela 22, apresentamos a razão entre os precondicionadores (SBAINV-NS no numerador e SBAINV-VAR no denominador) em relação aos números médios de iterações, para diversos tamanhos de bloco, com tolerância para o descarte de 0,1. Acima de 1 o BICGSTAB precondicionado pelo SBAINV-VAR faz menos iterações. Na Tabela 25, apresentamos as razões entre as densidades descritas na Tabela 23 (SBAINV-NS no numerador e SBAINV-VAR no denominador), para diversos tamanhos de bloco, com tolerância para o descarte de 0,1 (baseado nos dados da Tabela 23). Acima de 1 o SBAINV-VAR é menos denso.

Ao observarmos a Tabela 22, observamos que em média, os blocos de tamanho b2 precisaram de aproximadamente 68,2% (com desvio padrão de 13,5%) do número médio de iterações necessárias para as versões escalares (b1) enquanto que os blocos b3 precisaram de aproximadamente 48,5% (com desvio padrão de 15,1%) em comparação às versões escalares. Também podemos concluir que o SBAINV-VAR teve um melhor desempenho no número médio de iterações. Esse fato ocorre em todas as matrizes testadas

Tabela 23 Densidade dos precondicionadores para diversos tamanhos de bloco, com tolerancia 0,1.

Matriz	Precondicionador	b1	b2	b3	b3/b1
SHERMAN1	SBAINV-NS	2,05	3,70	21,75	10,61
	SBAINV-VAR	1,82	2,97	14,52	7,98
SHERMAN4	SBAINV-NS	1,62	2,58	8,93	5,51
	SBAINV-VAR	1,60	2,46	7,33	4,58
PDE900	SBAINV-NS	2,57	5,42	21,77	8,47
	SBAINV-VAR	2,41	5,02	16,96	7,04
PDE2961	SBAINV-NS	2,54	11,41	37,45	14,74
	SBAINV-VAR	2,28	7,78	22,78	9,99
HOR131	SBAINV-NS	1,81	4,61	13,90	7,68
	SBAINV-VAR	1,71	4,02	11,82	6,91
CAGE8	SBAINV-NS	0,34	2,14	3,50	10,29
	SBAINV-VAR	0,58	2,46	3,94	6,79

e em quaisquer tamanhos de blocos utilizados, à exceção da matriz HOR131 para b3. Em média, o SBAINV-NS obteve 20% mais iterações do que o SBAINV-VAR na versão escalar, 27% mais iterações nos blocos de tamanho b2 e 12% mais iterações nos blocos de tamanho b3. Da Tabela 24, podemos extrair que o SBAINV-VAR é melhor em 16 testes, empata em um e perde para o SBAINV-NS em apenas um teste.

As Tabela 23 e Tabela 25 nos permitem observar que o SBAINV-VAR produz um precondicionador menos denso do que o SBAINV-NS para cinco matrizes e mais denso para a matriz CAGE8. Como comentado anteriormente, uma densidade maior não é necessariamente uma limitação relevante, já que a localidade dos dados armazenados pelos blocos pode ter um impacto positivo no desempenho das operações de básicas de álgebra linear (uma outra fonte para este fato é [72]). Os precondicionadores nos blocos de ordem b2 foram, em média, 2,53 vezes mais densos que os precondicionadores das versões escalares (note que pela Tabela 16, os valores de b2 são, em média 2,66 vezes maiores que b1) enquanto que os precondicionadores produzidos pelo SBAINV-VAR com os tamanhos de bloco b3 são, em média, 7,19 vezes mais densos que as versões escalares (note que pela Tabela 16, os valores de b3 são, em média 6,83 vezes maiores que b1). Assim, os testes indicaram que a densidade foi aproximadamente proporcional ao aumento na ordem dos

blocos.

O SBAINV-VAR novamente foi superior nos dois quesitos aqui avaliados. Adicionalmente, observamos na Tabela 24 que para três casos as razões entre as iterações se mantêm estáveis e que para outros três existe uma redução desta razão com o aumento da ordem dos blocos. Da Tabela 25, vemos que a densidade do SBAINV-NS tende a crescer mais rapidamente do que densidade do SBAINV-VAR.

Tabela 24 Razões entre as iterações médias do BICGSTAB precondicionado (SBAINV-NS/SBAINV-VAR), com tolerância para o descarte de 0,1.

Matriz	b1	b2	b3
SHERMAN1	1,17	1,18	1,16
SHERMAN4	1,25	1,17	1,21
PDE900	1,28	1,41	1,35
PDE2961	1,33	1,14	1,03
HOR131	1,08	1,58	0,95
CAGE8	1,11	1,15	1,00
Média	1,20	1,27	1,12

Tabela 25 Razões entre as densidades dos precondicionadores (SBAINV-NS/SBAINV-VAR), com tolerância para o descarte de 0,1.

Matriz	b1	b2	b3
SHERMAN1	1,13	1,25	1,50
SHERMAN4	1,01	1,05	1,22
PDE900	1,07	1,08	1,28
PDE2961	1,11	1,47	1,64
HOR131	1,06	1,15	1,18
CAGE8	0,59	0,87	0,89
Média	0,99	1,14	1,28

A Tabela 26 mostra a razão entre as iterações do BICGSTAB sem precondicionamento, apresentadas na Tabela 17, e as iterações com o SBAINV-VAR, com b3, conferir Tabela 22. Acima de 1 o BICGSTAB precondicionado pelo SBAINV-VAR faz menos iterações. Neste caso, o método precondicionado se comportou melhor para as cinco matrizes, fazendo em média 90% menos iterações, lembrando que para a matriz HOR131, sem precondicionamento, o método não convergiu dentro do número máximo

Tabela 26 Razões entre as iterações médias do BICGSTAB sem precondicionamento (no numerador) e precondicionado pelo SBAINV-VAR (no denominador), com tamanhos de bloco do tipo b3 e tolerância para o descarte de 0,1.

Matriz	Razão
SHERMAN1	31,04
SHERMAN4	5,31
PDE900	13,63
PDE2961	11,25
CAGE8	1,08

de iterações. Para uma das matrizes, CAGE8, o método precondicionado foi tão melhor quanto os demais e teve uma margem de apenas 7% em média.

CONCLUSÃO

O AINV é um dos métodos mais conhecidos na obtenção de precondicionadores para sistemas lineares de grande porte, muito comuns em problemas da indústria e da ciência. Neste trabalho, buscamos, primeiramente, revisar as características mais relevantes do algoritmo de biconjugação e do AINV, como as relações com os fatores LDU, as estratégias de descarte utilizadas, o cálculo dos pivôs, dentre outros fatores. Fizemos uma extensa análise das principais variações do AINV e como elas foram abordadas na literatura. Destacamos suas principais diferenças e semelhanças e, por isso, as classificamos em quatro classes: classe AINV, classe FFAPINV, classe AINV-LU e classe Pivoteamento. Também analisamos as complexidades de cada um dos algoritmos e verificamos que, em todos os métodos, o fato de A ser esparsa e a utilização dos descartes são os principais pontos que os tornam computacionalmente viáveis.

Vimos que poucas variações do AINV em blocos foram propostas, sendo elas o BAINV-NS [22] e o BAINV [11]. A abordagem em blocos pode trazer benefícios quanto à melhora de estabilidade, melhora do desempenho além de preservar as propriedades físicas relativas a blocagem da matriz. Nós, buscamos, então, analisar com mais detalhes o BAINV, que é a versão em blocos do AINV para matrizes simétricas. Nós demonstramos sua consistência e que ele não quebra, sob determinadas condições, para matrizes do tipo M e H. Além disso, adaptamos esse algoritmo para o caso não simétrico, demonstrando também sua consistência. Além disso, provamos alguns resultados relativos ao AINV não simétrico em blocos que possibilitaram desenvolver versões em blocos para algumas variações do AINV escalar, sendo elas: SBAINV-NS, BFFAPINF, BSAINV-VAR e BRIF.

Por fim, apresentamos os resultados dos testes numéricos comparando o SBAINV-NS e o BSAINV-VAR. O SBAINV-NS é uma versão não simétrica do BAINV e livre de quebra para matrizes positivas definidas. Já o SBAINV-VAR é a versão em blocos do SAINV-VAR. Os principais resultados dos experimentos numéricos indicam que o SBAINV-VAR possui melhor desempenho em comparação ao SBAINV-NS, tanto em número médio de iterações do BICGSTAB, quanto em densidade do precondicionador.

Além disso, nestes experimentos, o valor da tolerância do descarte de $0,1$ foi adequado e o uso de uma estrutura em blocos representou um ganho importante no número de iterações, ainda que tenha tornado os precondicionadores mais densos, com o aumento da ordem dos blocos.

Como trabalhos futuros, pretendemos desenvolver mais variações do AINV para matrizes em blocos, como, por exemplo, adaptações em blocos para as versões do AINV com pivoteamento (AINVP, RIFP, LLAINVP). Em relação à implementação dos testes, pretendemos utilizar outros critérios para descarte (que não sejam puramente por tolerância de magnitude dos blocos), além de realizar reordenamentos e escalamentos nas matrizes. Pretendemos ainda realizar uma comparação do tempo computacional de aplicação e produção do precondicionador. Por fim, iremos realizar implementações em C++, tanto com MPI quanto com OpenMP, para medir o desempenho destas alternativas em computadores paralelos híbridos e compará-las com outros precondicionadores conhecidos, como as fatorações incompletas e o multigrid, para matrizes reais oriundas de problemas de simulação de reservatórios de petróleo e de geomecânica.

REFERÊNCIAS

[1] AJIZ, M. A.; JENNINGS, A. A Robust Incomplete Cholesky Conjugate Gradient Algorithm. *Int. J. Numerical Methods Engrg.*, v. 20, p. 949-966, 1984.

[2] ABDELFATTAH, A.; ANZT, H.; DONGARRA, J.; GATES, M.; HAIDAR, A.; KURZAK, J.; LUSZCZEK, P.; TOMOV, S.; YAMAZAKI, I.; YARKHAN, A. Linear algebra software for large-scale accelerated multicore computing. *Acta Numerica*, v. 25, p. 1-160, 2016.

[3] ALMEIDA, M. C. Precondicionador de inversa aproximada por blocos. *PhD thesis*, Universidade do Estado do Rio de Janeiro, 2019.

[4] 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.

[5] AMESTOY, P. R.; DAVIS T. A.; DUFF I. S. An Approximate Minimum Degree Ordering Algorithm. *SIAM J. Matrix Analysis & Applic.*, v. 17, n. 4, p. 886-905, 1996.

25, 1951.

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

[7] AXELSSON, O.; KOLOTINA, L. Y. Diagonally Compensated Reduction and Related Preconditioning Methods, *Numer. Linear Algebra Appl.*, v. 1, p. 155-177, 1994.

[8] BARNARDY, S. T.; GROTEZ, M. J. A block version of the spai preconditioner. *Proceedings of the 9th SIAM conference on Parallel Processing for Scientific Computing,*, San Antonio, TX, 1999.

[9] 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. *SIAM*, Philadelphia, PA, 1994.

[10] BENZI M. Preconditioning techniques for large linear systems: a survey. *Journal of Computational Physics*, v. 182, n. 2, p.418-477, 2002.

[11] BENZI, M.; KOUHIA, R.; TŮMA, 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; TŮMA, 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.; TŮMA, M. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, v. 19, p. 968-994, 1998.

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

[15] BENZI, M.; CULLUM, J.; TŮMA M. Robust Approximate Inverse Preconditioning for the Conjugate Gradient Method. *SIAM J. Sci. Comput.*, v. 22, n. 4, v. 1318-1332. 2000.

[16] BENZI, M.; TŮMA, M. A Robust Preconditioner with Low Memory Requirements for Large Sparse Least Squares Problems. *Siam J. S. C. Vol.*, v. 25, n. 2, p. 499–512. 2003.

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

[18] BERMAN, A.; PLEMMONS, R. J. Nonnegative Matrices in the Mathematical Sciences. *SIAN*, 1994.

[19] BOLLHÖFER, M. e SAAD, Y. On the relations between ILUs and factored approximate inverses. *SIAM J. Matrix Anal. Appl.*, v. 24, n. 1, p. 219–237, 2002.

[20] BOLLHÖFER, M.; SAAD Y. A factored approximate inverse preconditioner with pivoting. *SIAM J.Matrix Anal. Appl.*, v. 23, p. 692–705. 2002.

[21] BRIDSON, R.; TANG, W. Ordering, Anisotropy, and Factored Sparse Approximate Inverses.*SIAM J. Sci. Comput.*, v. 21, p. 867-882, 1999.

[22] BRIDSON, R. e TANG, W. Refining an Approximate Inverse. *Appeared in J. of Comp. and Appl. Math*, v. 123, p. 293–306, 2000.

[23] BRU, R.; CERDÁN J.; MARÍN, J.; MAS, J. Preconditioning sparse nonsymmetric linear systems with the Sherman–Morrison formula. *SIAM Journal on Scientific Computing*, v. 25, n. 2, p.:701-715, 2003.

[24] BRU, R.; CORRAL, C.; GIMENEZ, I.; MAS J. Classes of general H-matrices . *Linear Algebra and its Applications, v. 429, p. 2358–2366, 2008.*

[25] BRU, R.; MARÍN, J.; MAS, J.; TŮMA, M. Balanced incomplete factorization. *SIAM Journal on Scientific Computing*, v. 30, n. 5, p. 2302-2318, 2008.

[26] BRU, R.; MARÍN, J.; MAS, J.; TŮMA, M. Improved balanced incomplete factorization. *SIAM Journal on Matrix Analysis and Applications*, v. 31, n. 5, p. 2431-2452, 2010.

[27] BUTTARI, A.; LANGOU, J.; KURZAK, J.; DONGARRA, J. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Computing*, v. 35, n. 1, p. 38-53, 2009.

[28] 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.

[29] CERDÁN J.; FARAJ T.; MALLA N.; MARÍN J.; MAS J.; Block approximate inverse preconditioners for sparse nonsymmetric linear systems. *Electronic Transactions on Numerical Analysis*, v. 37, p. 23-40, 2010.

[30] CHOW, E.; SAAD, Y. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM Journal on Scientific Computing*, v. 19, n. 3, p. 995-1023, 1998.

[31] DONGARRA, J.; DUFF, I.S.; SORENSEN, D. C.; VORST, V. D. H. A. *Numerical Linear Algebra for High-Performance Computers. SIAM*, Philadelphia, PA, USA, 1998.

[32] FERRONATO, M.; JANNA, C.; PINI, G. A generalized block FSAI preconditioner for nonsymmetric linear systems. *Journal of Computational and Applied Mathematics*, v. 256, n. 0, p. 230-241, 2014.

[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 J. Sci. Comput.*, v.14, n.2, p. 470-482, 1993.

[35] FROMMER, A.; MAASS, P. Fast CG-based methods for Tikhonov–Phillips regularization. *SIAM J. Sci. Comput*, v. 20, n.5, 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, p. 10-17. 2004.

[37] GOLUB, G. H. LOAN, C. F. Matrix computations. *Johns Hopkins University Press*, 1996.

[38] GROTE, M. J.; HUCKLE, T. Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, v. 18, n. 3, p. 838-853, 1997.

[39] HESTENES M.; STIEFEL E. Methods of conjugate gradients for solving linear systems. *J . Res. Natl. Bur. Stand* , v. 49, p. 409-436, 1952.

[40] HIGHAM, N. J. Accuracy and stability of numerical algorithms. *SIAM*, University of Manchester, England, 2002.

[41] HORN, R. A.; JOHNSON, C.R. Matrix Analysis. *Cambridge University Press*, 2ª Edição, 2013.

[42] JANNA, C.; FERRONATO, M.; GAMBOLATI, G. A block FSAI-ILU parallel preconditioner for symmetric positive definite linear systems. *SIAM Journal on Scientific Computing*, v. 32, n. 5, p. 2468-2484, 2010.

[43] JANNA, C.; FERRONATO, M.; GAMBOLATI, G. Enhanced block fsai preconditioning using domain decomposition techniques. *SIAM Journal on Scientific Computing*, v. 35, n.5, p. S229-S249, 2013.

[44] JANNA, C.; FERRONATO, M.; GAMBOLATI, G. The use of supernodes in factored sparse approximate inverse preconditioning. *SIAM Journal on Scientific Computing*. v. 37, n. 1, p. C72-C94, 2015.

[45] KARYPIS, G.; KUMAR, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 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. *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 L.Y. Two-Sided Bounds for the Inverse of an H-Matrix. *Linear Algebra and its Applications, v. 225, p. 114-123, 1995.*

[49] LEE, E.J.; ZHANG, J. Fatored approximate inverse preonditioners with dynamic sparsity patterns. *Computers & Mathematics with Applications*, v. 62, n. 1, p.235-242, 2011.

[50] LIU, J. W. H. Modification of the minimum-degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software*, v. 11, p. 141-153, 1985.

[51] LIU, J. W. H.; ESMOND, G. N.; PEYTON, P. W. On finding supernodes for sparse matrix computations. *SIAM J. Matrix Anal. Appl*, v. 14, n.1, p. 242-252, 1993.

[52] LIU, W.; LI A.; HOGG, J.; DUFF, I. S.; VINTER B. *A Synchronization-Free Algorithm for Parallel Sparse Triangular Solves Euro-Par 2016: Parallel Processing*, v. 9833, p. 617-630, 2016.

[53] MANTEUFFEL, T.A. An incomplete factorization technique for positive definite linear system. *textitMathematics of Computation*, v. 34, p. 473-497, 1980.

[54] MEIJERINK, J. A.; VORST, V. D. H. A. An iterative solution method for linear systems of which the coefficient matrix is a symmetric m-matrix. *Math. Comp.*, v. 31, p. 148-162, 1977.

[55] MEURANT, G. Computer solution of large linear systems. *Elsevier*, North Holland, 1999.

[56] OSTROWSKI, A., Uber die Determinanten mit uberwiegender Hauptdiagonal. *Comment. Math. Hek.*, v. 10, p. 69-96, 1937.

[57] PAPADOPOULOS, A. T.; DUFF, I. S.; WATHEN, A. J. Incomplete Orthogonal Factorization Methods Using Givens Rotations II: Implementation and Results. *Oxford University Computing Laboratory*, v. Report, no 02/07, n. 2, p. 175-188, 2002.

[58] PLEMMONS, R. J. M-matrix characterizations. I—nonsingular M-matrices. *Linear Algebra and Its Applications*, v. 18, p. 175-188, 1977.

[59] RAFIEI, A.; TOUTOUNIAN, F. New breakdown-free variant of AINV method for nonsymmetric positive definite matrices. *Journal of Computational and Applied Mathematics* v. 219. pp. 72-80. 2008.

[60] RAFIEI, A.; BOLLHÖFER, M. Robust incomplete factorization for nonsymmetric matrices. *Numerische Mathematik*, v.118. p. 247–269. 2011.

[61] 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.

[62] RAFIEI, A. Left-looking version of AINV preconditioner with complete pivoting strategy. *Linear Algebra and its Applications* v. 445, p. 103–126. 2014.

[63] SAAD, Y. *Iterative Methods for Sparse Linear Systems. SIAM*, 2ª edição, 2003.

[64] SAAD, Y.; VORST, V. D. H. A. Iterative solution of linear systems in the 20th century. *Journal of Computational and Applied Mathematics*, v. p. 1-33, 2000.

[65] SAAD, Y.; ZHANG, J. BILUM: 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.

[66] 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.

[67] SALKUYEH, D. K.; ROOHANI, H.. On the Relation between the AINV and the FAPINV Algorithms. *International Journal of Mathematics and Mathematical Sciences*, v.2009, 2009.

[68] SCHNABEL, R. B.; ESKOW, E. A new modified Cholesky factorization. *SIAM J. Sci. and Stat. Comput.*, v. 11(6), p. 1136–1158. 1981.

[69] SEDLACEK, M. Sparse Approximate Inverses for Preconditioning, Smoothing, and Regularization. *PhD thesis*, Universität München, 2012.

[70] THOMAS, G. W.; THURNAU D. H. Reservoir simulation using an adaptive implicit method. *SPE Journal*, v.23, p. 760-768, 1983.

[71] YEREMIN, A.; KOLOTINILINA, L.; NIKISHIN, A. Factorized sparse approximate inverse preconditionings. III. iterative construction of preconditioners. *Journal of Mathematical Sciences*, v. 101(4), p. 3237-3254, 2000.

[72] WILLIAMS, S. W.; WATERMAN, A.; PATTERSON, D. A. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Communications of the ACM*, v. 52, n. 4, p. 65-76, 2009.