Universidade Federal Fluminense

JOSEANA VEIGA DE SOUZA FRANGO

O Método dos Gradientes Conjugados Precondicionado com Estrutura de Dados CSR

VOLTA REDONDA

2022

JOSEANA VEIGA DE SOUZA FRANGO

O Método dos Gradientes Conjugados Precondicionado com Estrutura de Dados CSR

Dissertação apresentada ao Programa de Pós-graduação em Modelagem Computacional em Ciência e Tecnologia da Universidade Federal Fluminense, como requisito parcial para obtenção do título de Mestre em Modelagem Computacional em Ciência e Tecnologia. Área de Concentração: Modelagem Computacional.

Orientador:

Prof. D.Sc. Thiago Jordem Pereira

Coorientador:

Prof. D.Sc. Ricardo Silveira Sousa

Universidade Federal Fluminense

VOLTA REDONDA

2022

O Método dos Gradientes Conjugados Precondicionado com Estrutura de Dados CSR

Joseana Veiga de Souza Frango

Dissertação apresentada ao Programa de Pós-graduação em Modelagem Computacional em Ciência e Tecnologia da Universidade Federal Fluminense, como requisito parcial para obtenção do título de Mestre em Modelagem Computacional em Ciência e Tecnologia. Área de Concentração: Modelagem Computacional.

Aprovada por:

Prof. Nome Nesta Forma, Ph.D. / IC-UFF(Presidente)

Prof. Nome Nesta Forma, D.Sc. / IM-UFRJ

Prof. Nome Nesta Forma, Ph.D. / IC-UFF

Prof. Nome Nesta Forma, D.Sc. / IF-UFRRJ

Prof. Nome Nesta Forma, Ph.D. / DCC-UFMG

Prof. Nome Nesta Forma, D.Sc. / LNCC

Volta Redonda, 27 de Outubro de 2022.

Para meus pais e familia

Agradecimentos

Gratidão a Deus, pela vida.

Gratidão a minha família por fazer o (im)possível, desde sempre, para que eu estude.

Gratidão ao meu namorado Márnel, pelo seu carinho e apoio todos esses anos.

Gratidão à amizade sincera e fraterna dos meus amigos Ítalo, Melise, Valéria e Wesley.

Gratidão aos amigos do mestrado Daniel, Leonardo, Louise e Natália, pela amizade, convivência e inúmeras caronas nesses anos.

Gratidão ao meu orientar Thiago Jordem, pela confiança, orientação e dedicação neste trabalho e em outros momentos fundamentais.

Gratidão, em especial, aos professores Ricardo Sousa pelas contribuições a este trabalho; e ao professor Wagner Telles por inúmeras dicas e sua imensa paciência em suas disciplinas.

Resumo

Ao lidar com problemas de diversas áreas científicas como a Física, Química, Engenharias e até a própria Matemática, muitos deles são modelados por sistemas lineares. Algumas dessas aplicações apresentam certo nível de dificuldade, observados em sistemas lineares de grande porte ou esparsos. Nesses casos, buscar uma solução ao problema diretamente pode ser inviável. Uma alternativa a tal necessidade é utilizar método iterativos para buscar uma solução aproximada, porém precisa, do sistema linear. Com isso, ao longo das últimas décadas, a investigação de métodos numéricos iterativos na resolução de sistemas lineares vem ganhando destaque, principalmente o uso do métodos dos Gradientes Conjugados (GC). Sua convergência eficiente se deve principalmente ao seu conjunto de direções A-conjugadas linearmente independentes. Entretanto, tal eficiência é prejudicada quando a matriz de coeficientes é esparsa ou mal condicionada, por exemplo. A fim de reparar tal dano ao método dos GC, emprega-se o precondicionamento da matriz de coeficientes. Dessa forma, resolve-se um sistema linear equivalente ao original, cuja matriz precondicionadora seja uma aproximação da matriz original de coeficientes. As inúmeras maneiras para se gerar uma matriz pré-condicionadora, faz com que existam diferentes tipos de precondicionadores, dos quais adota-se neste trabalho os de Jacobi, Damped Jacobi, Forward Gauss Seidel (FGS), Backward Gauss-Seidel (BGS), Symmetric Gauss-Seidel (SGS), Forward Successive Over Relaxation (FSOR), Backward Successive Over Relaxation (BSOR) e Symmetric Successive Over Relaxation (SSOR). Além disso, atrelá-se ao GC a estrtutura de dados *Compressed Sparse Row* (CSR). Logo, este trabalho tem como intuito analisar a convergência do método dos Gradientes Conjugados nas condições apresentadas, aplicado a sistemas lineares esparsos obtidos em repositórios de matrizes esparsas.

Abstract

When dealing with problems from various scientific areas such as Physics, Chemistry, Engineering and even Mathematics itself, many of them are modeled by linear systems. Some of these applications present a certain level of difficulty, observed in large or sparse linear systems. In such cases, seeking a solution to the problem directly may be unfeasible. An alternative to this need is to use itative method to seek an approximate but precise solution of the linear system. Thus, over the last decades, the investigation of itethanising numerical methods in the resolution of linear systems has gained prominence, mainly the use of conjugated gradient methods (GC). Its efficient convergence is mainly due to its set of linearly independent A-conjugated directions. However, such efficiency is impaired when the coefficient matrix is sparse or poorly conditioned, for example. In order to repair this damage to the GC method, the preconditioning of the coefficient matrix is used. Thus, a linear system equivalent to the original is solved, whose preconditioning matrix is an approximation of the original matrix of coefficients. The numerous ways to generate a preconditioner matrix, makes there are different types of preconditioners, from which we adopt in this work those of Jacobi, Damped Jacobi, Forward Gauss Seidel (FGS), Backward Gauss-Seidel (BGS), Symmetric Gauss-Seidel (SGS), Forward Successive Over Relaxation (FSOR), Backward Successive Over Relaxation (BSOR) and Symmetric Successive Over Relaxation (SSOR). In addition, the Compressed Sparse Row (CSR) data strtuture is tied to the CG. Therefore, this work aims to analyze the convergence of the conjugated gradients method under the conditions presented, applied to sparse linear systems obtained from sparse matrix repositories.

Palavras-chave

1. Sistemas Lineares Esparsos
2. Gradientes Conjugados
3. Precondicionadores
4. CSR

Glossário

Ja	:	Jacobi
DJa	:	Damped Jacobi
BGS	:	Backward Gauss-Seidel
FGS	:	Forward Gauss-Seidel
SGS	:	Symmetric Gauss-Seidel
SOR	:	Sucessive Over Relaxation
SSOR	:	Symmetric Sucessive Over Relaxation
SPD	:	Simétrica Positiva Definida
DC	:	Direções Conjugadas
GC	:	Gradientes Conjugados
GCPC	:	Gradientes Conjugados Precondicionado
PMD	:	Passo Máximo Descendente
GE	:	Grau de Esparsidade

Lista de Tabelas

Sumário

Capítulo 1

Introdução

O encontro entre matemática e computação ocorre da necessidade de resolução de problemas de diversas áreas científicas e engenharias, os quais envolvem a resolução de sistemas de equações lineares esparsos por meio de técnicas numéricas baseadas em conceitos de álgebra linear. Parte da modelagem matemática desses problemas recaem em sistemas do tipo $\mathbf{Ax} = \mathbf{b}$, em que muitas vezes a formulação da matriz $\mathbf{A}$ tem origem na discretização de equações diferencias parciais. Ainda que sejam, muitas vezes, de interesse de estudo como já foi parte da pesquisa desta autora em momento anterior [7, 8], a natureza desses não se restringe as equações diferenciais parciais [1]. De certo modo, as propriedades de cada sistema linear a ser resolvido está intrisicamente relacionado as características e complexidades do caráter dos modelos adotados. Isso faz com que a solução $\mathbf{x}$ do sistema linear não seja tão simples de se obter.

Atualmente, as estratégias de resolução de sistemas lineares podem ser dividas em dois tipos: métodos diretos e métodos iterativos. Os métodos diretos obtem a solução exata dos sistemas lineares em um número finito de operações aritméticas. Enquanto os métodos iterativos determinam a solução aproximada em um número de passos fixado [4]. Embora esses métodos tenham em comum considerar as propriedades da matriz de coeficientes $\mathbf{A}$, eles se distanciam em termos de eficiência. Como já posto antes, a natureza dos problemas molda a construção da matriz de coeficientes de modo que é comum que sejam de grande porte e/ou esparsas [16]. Desse ponto de vista, os métodos diretos não se mostram eficientes, uma vez que escalam mal com o tamanho do problema em termos de contagens de operação e requisitos de memória [1]. Nesse caso, os métodos iterativos são mais eficientes, já que não ocorrem preenchimentos das posições nulas da matriz, exigindo menos armazenamento e menos operações.

Apesar de aparentarem ser a melhor opção disponível, os método iterativos falham na

convergência à solução. Nesse contexto, o precondicionamento surge como uma técnica para transformar o sistema original em um equivalente, com propriedades mais favoráveis aos solucionadores iterativos [1]. O tipo de precondionamento depende tanto da escolha do método iterativo, quanto das propriedades da matriz de coeficientes [15]. Segundo Saad, equilibrar todas essas particularidades faz da escolha de um bom precondicionador uma combinação entre arte e ciência. Assim, ao selecioná-lo deve considerar que sua contrução tenha um custo aceitável de construção e o sistema precondicionado seja fácil, a um tempo menor do que o sistema original. Mesmo que seja uma área em incessante desenvolvimento, ainda não existe um precondicionador ideal para determinado tipo de problema. Tratar essas matrizes com certa especificidade é limitar sua aplicação. Nesse âmbito, os precondicionadores de tratamento algébrico que levam em consideração os aspectos da matriz **A** se destacam como aplicadores universais [1, 16]. Muitos deles são desenvolvidos com base nas iterações de métodos como Jacobi, Gauss-Seidel, SOR, SSOR, entre outros [10, 15, 20, 21]; ou ainda em fatorações de forma incompleta como LU, Cholesky, por exemplo [1, 15, 19].

Outra forma de impactar a convergência quando se trata de matrizes esparsas é a maneira de armazenar seus elementos. No geral, as matrizes de coeficientes são descritas de forma indexida, ou seja, a partir de um par de índices - linhas e colunas - que indicam em que posição seus elementos estão. Se essas matrizes são esparsas, poucos desses elementos são não nulos e acessá-los nessa estrutura é limitante. Logo, otimizar o armazenamento dessas matrizes é fundamental para proporcionar melhor acesso aos seus elementos [5]. Dentre as estruturas existentes estão a COORD (coordinate storage) ou armazenamento por coordenadas; a CRS (compressed row storage) ou compactado por linha; CCS (compressed column storage) ou compactado por coluna; BCSR (block compressed sparse row); entre outras [15]. Computacionalmente, ao utilizar uma das estruras mencionadas, espera-se economizar armazenamento e tempo.

Diante disso, esta dissertação tenta constribuir na pesquisa do uso de precondicionadores na resolução de sistemas lineares esparsos, ao investigar o desempenho de um dos métodos iterativos mais relevantes do subespaço de Krylov: o método dos Gradientes Conjugados [3, 4, 13, 15, 18]. Desenvolvido, em 1952 por Hestenes e Stiefel [11], esse método demorou cerca de 20 anos para ser reconhecido como um método iterativo. O artigo de Saad e van der Vorst [16] é fundamental para conhecer sua evolução ao longo das últimas décadas, que culmina com a sua ligação com matrizes simétricas positivas definidas [9]. Por não se ter garantia de sua convergência, esta transforma-se em objeto de investigação ao combinar precondicionadores e a estrutura de dados CSR, ao método

dos GC. Essa versão é chamada de método dos Gradientes Conjugados Precondicionado (PCGC).

1.1 Objetivos

1.1.1 Objetivo Geral

Esta pesquisa objetiva utilizar precondicionadores e estrutura de dados no método dos Gradientes Conjugados, visando uma melhoria na eficiência computacional da convergência do método, para sistemas de equações lineares esparsos e/ou de grande porte.

1.1.2 Objetivos Específicos

Além do objetivo já exposto, a seguir, outros particulares são considerados:

- analisar a convergência do método dos Gradientes Conjugados Precondicionado na resolução de sistemas lineares cuja matriz dos coeficientes;
- comparar os resultados encontrados pelo método dos GC e GCPC, com e sem a estrutura de dados CSR;
- comparar os resultados encontrados pelo método dos GC e GCPC, com e sem o refinamento iterativo de solução.

1.2 Estrutura da dissertação

- O Capítulo 1 expõe o método dos Gradientes Conjugados e sua forma precondicionada;
- Já o Capítulo 2 apresenta as classes de precondicionadores escolhidos e a técnica de refinamento iterativo de solução;
- Na sequência, o Capítulo 3 aborda a estrutura de dados CSR, bem como outras formas de armazenamento;
- No Capítulo 4, serão vistos e discutidos os resultados numéricos obtidos nas simulações numéricas com GC e GCPC dos problemas de repositório analisados.

Capítulo 2

Método dos Gradientes Conjugados

2.1 O Método dos Gradientes Conjugados

Proposto por Magnus R. Hestenes e Eduard Stiefel, no artigo *"Methods of Conjugate Gradients for Solving Linear Systems"* em 1952, o método Gradiente Conjugado - GC - é um dos métodos iterativos mais famosos e úteis na resolução de sistemas lineares esparsos [3, 4, 18].

Embora seja, muitas vezes, um método eficiente em termos de convergência, essa pode ser melhorada ao associar o GC a um precondicionador, com intuito de tornar mais rápido e preciso os cálculos visando a solução do sistema linear [3, 15].

Com efeito, neste Capítulo estuda-se o método dos Gradientes Conjugados Pré-Condicionado, o qual pode ser observado com mais detalhes nos trabalhos de Burden [3], Cunha [4], Saad [15] e Shewchuk [18].

2.1.1 O Problema de Minimização

O método GC tem início com problema de minimização de uma função $F(\mathbf{x})$, sendo $F : \mathbb{R}^n \to \mathbb{R}$ uma função quadrática dada por [18]:

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}\mathbf{A}\mathbf{x}^T - \mathbf{b}^T\mathbf{x} + c, \tag{2.1}$$

onde $\mathbf{A} \in \mathbb{R}^{n\times n}$ é uma matriz simétrica ($\mathbf{A}^T = \mathbf{A}$) e positiva definida ($\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$), $\mathbf{x} \in \mathbb{R}^n$ e $\mathbf{b} \in \mathbb{R}^n$ são vetores, e $c \in \mathbb{R}$ é um escalar constante.

Como forma de provar que $F(\mathbf{x})$ é realmente uma função quadrática, toma-se $n = 2$

na Equação (2.1) de modo que

$$\begin{aligned} F(\mathbf{x}) &= \frac{1}{2}\left(a_{11}x_1^2 + a_{12}x_1x_2 + a_{21}x_1x_2 + a_{22}x_2^2\right) - b_1x_1 - b_2x_2 \\ F(\mathbf{x}) &= \frac{1}{2}\left(a_{11}x_1^2 + 2a_{12}x_1x_2 + a_{22}x_2^2\right) - b_1x_1 - b_2x_2, \end{aligned} \tag{2.2}$$

ao usar a simetria da matriz $\mathbf{A}$. Como indica Cunha [4], as curvas de nível da função (2.2) são constantes e, portanto, são elipses e sua superfície é um parabolóide. Além disso, como a matriz $\mathbf{A}$ é definida positiva essa função é fortemente convexa [18] e, consequentemente, possui um único mínimo.

A busca desse mínimo leva ao cálculo do vetor gradiente que aponta na direção em que há maior crescimento na função [4]. Logo, calcula-se o gradiente da função (2.1) por

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \dfrac{\partial F}{\partial \mathbf{x_1}} \\ \dfrac{\partial F}{\partial \mathbf{x_2}} \\ \vdots \\ \dfrac{\partial F}{\partial \mathbf{x_n}} \end{bmatrix} \tag{2.3}$$

$$\nabla F(\mathbf{x}) = \frac{1}{2}\mathbf{A}^T\mathbf{x} + \frac{1}{2}\mathbf{A}\mathbf{x} - \mathbf{b}. \tag{2.4}$$

Quando o vetor gradiente se anula em um ponto $\mathbf{x}$, ou seja,

$$\nabla F(\mathbf{x}) = \mathbf{0}, \tag{2.5}$$

então $\mathbf{x}$ é um ponto crítico da função $F(\mathbf{x})$. Assim,

$$\frac{1}{2}\mathbf{A}^T\mathbf{x} + \frac{1}{2}\mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0} \tag{2.6}$$

Como $\mathbf{A}$ é simétrica, a Equação (2.6) é reduzida para

$$\mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0}. \tag{2.7}$$

Logo, pela Equação (2.7), obtém-se o sistema linear

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2.8}$$

onde a matriz dos coeficientes $\mathbf{A}$, o vetor de variáveis $\mathbf{x}$ e o vetor dos termos independentes $\mathbf{b}$ são dados, respectivamente, por:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \tag{2.9}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \tag{2.10}$$

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}. \tag{2.11}$$

Além disso, retomando novamente a propriedade de $\mathbf{A}$ ser definida positiva, $\mathbf{x}$ será também ponto mínimo de $F(\mathbf{x})$, como se esperava. Portanto, resolver o sistema (2.8) equivale a encontrar um $\mathbf{x}$ que minimize a função quadrática (2.1).

2.1.2 O Método Passo Máximo Descendente

Após assegurar a existência de um ponto que minimiza a função quadrática é preciso construir um caminho que leve ao vértice do parabolóide, ou seja, perto o suficiente da solução do sistema (2.8). Assim, o método do Passo Máximo Descendente (PMD) é utilizado para percorrer tal caminho a partir de uma composição de direções de busca do minimizador.

Desse modo, supondo que $\mathbf{x}^*$ seja o vetor que minimiza a Equação (2.1), busca-se uma série de pontos $x_1, x_2, ..., x_n \in \mathbf{x}$ que sejam suficientemente próximos de $\mathbf{x}^*$, associados às direções de maior redução. Isso ocorre, pois o gradiente aponta na direção de crescimento máximo de F. Dessa forma, a partir de um ponto inicial $\mathbf{x}^{(0)}$, os próximos passos são dados por

$$\mathbf{x}^{(k)} = \mathbf{x}^{(0)} + \alpha^{(k)}\mathbf{r}^{(0)}, \tag{2.12}$$

onde $\mathbf{r}^{(0)}$ é o resíduo associado ao ponto $\mathbf{x}^{(0)}$. Tomando o resíduo como a direção de descida máxima, determinado pela Equação (2.7), tem-se

$$\begin{aligned} \mathbf{r}^{(k)} &= -\nabla F(\mathbf{x}^{(k)}) \\ \mathbf{r}^{(k)} &= -\left[\mathbf{A}\mathbf{x}^{(k)} - \mathbf{b}\right] \\ \mathbf{r}^{(k)} &= \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}. \end{aligned} \tag{2.13}$$

De outro modo, $\mathbf{r}^{(k)}$ determina a diferença entre a solução $\mathbf{x}^*$ e a sua aproximação $\mathbf{x}^{(k)}$.

Observa-se, da Equação (2.12), que α se relaciona com o resíduo $\mathbf{r}$. Essa variável determina o tamanho do passo, minimizando $F(\mathbf{x}^{(k)}+\alpha^{(k)}\mathbf{r}^{(k)})$ quando a derivada direcional de F é nula, isto é,

$$\frac{d}{d\alpha}F(\mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{r}^{(k)}) = 0. \tag{2.14}$$

Se $\mathbf{r}^{(k)}$ e $\mathbf{r}^{(0)}$ forem ortogonais [3], então $\mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{r}^{(k)}$ é o mínimo da função quadrática nesse ponto, e a Equação (2.14) é garantida. E ainda é possível obter o valor de α considerando

$$\frac{d}{d\alpha}F(\mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{r}^{(k)}) = \frac{d}{d\alpha}\left[\frac{1}{2}(\mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{r}^{(k)})^T\mathbf{A}(\mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{r}^{(k)}) - \mathbf{b}^T(\mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{r}^{(k)})\right] \tag{2.15}$$

Derivando (2.15) em relação a α e aplicando as regras do produto e da cadeia:

$$\frac{d}{d\alpha}F(\mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{r}^{(k)}) = \frac{1}{2}\mathbf{r}^T\mathbf{A}(\mathbf{x}^{(k)} + \alpha\mathbf{r}^{(k)}) + \frac{1}{2}(\mathbf{x}^{(k)} + \alpha\mathbf{r}^{(k)})^T\mathbf{A}(\mathbf{r}^{(k)})^T - \mathbf{b}^T\mathbf{r}^{(k)} \tag{2.16}$$

Aplicando a distributiva em (2.16):

$$\frac{d}{d\alpha}F(\mathbf{x}^{(k)}+\alpha^{(k)}\mathbf{r}^{(k)}) = \frac{1}{2}(\mathbf{r}^{(k)})^T\mathbf{A}\mathbf{x}^{(k)}+\frac{1}{2}\alpha(\mathbf{r}^{(k)})^T\mathbf{A}\mathbf{r}^{(k)}+\frac{1}{2}(\mathbf{x}^{(k)})^T\mathbf{A}\mathbf{r}^{(k)}+\frac{1}{2}\alpha(\mathbf{r}^{(k)})^T\mathbf{A}\mathbf{r}^{(k)}-\mathbf{b}^T\mathbf{r}^{(k)} \tag{2.17}$$

Como $\mathbf{x}^T\mathbf{A}\mathbf{r} = (\mathbf{A}\mathbf{r})^T\mathbf{x} = \mathbf{r}^T\mathbf{A}^T\mathbf{x} = \mathbf{r}^T\mathbf{A}\mathbf{x}$ e $\mathbf{b}^T\mathbf{r} = \mathbf{b}\mathbf{r}^T$, já que $\mathbf{A}$ é simétrica e a propriedade de trasposição se aplica aos vetores $\mathbf{r}$ e $\mathbf{b}$, então (2.17) é reescrita como

$$\begin{aligned} \frac{d}{d\alpha}F(\mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{r}^{(k)}) &= \alpha(\mathbf{r}^{(k)})^T\mathbf{A}\mathbf{r}^{(k)} + (\mathbf{r}^{(k)})^T\mathbf{A}\mathbf{x}^{(k)} - \mathbf{b}(\mathbf{r}^{(k)})^T \\ &= \alpha(\mathbf{r}^{(k)})^T\mathbf{A}\mathbf{r}^{(k)} - (\mathbf{r}^{(k)})^T\left(\mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}\right) \\ &= \alpha(\mathbf{r}^{(k)})^T\mathbf{A}\mathbf{r}^{(k)} - (\mathbf{r}^{(k)})^T\mathbf{r}^{(k)}. \end{aligned} \tag{2.18}$$

Igualando as Equações (2.14) e (2.18) segue

$$\begin{aligned} \alpha(\mathbf{r}^{(k)})^T\mathbf{A}\mathbf{r}^{(k)} - (\mathbf{r}^{(k)})^T\mathbf{r}^{(k)} &= 0 \\ \alpha(\mathbf{r}^{(k)})^T\mathbf{A}\mathbf{r}^{(k)} &= (\mathbf{r}^{(k)})^T\mathbf{r}^{(k)} \\ \alpha^{(k)} &= \frac{(\mathbf{r}^{(k)})^T\mathbf{r}^{(k)}}{(\mathbf{r}^{(k)})^T\mathbf{A}\mathbf{r}^{(k)}}, \end{aligned} \tag{2.19}$$

onde α é o parâmetro que regula o tamanho do passo associado ao resíduo.

Logo, a atualização dos pontos é expressa por

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{r}^{(k)}, \tag{2.20}$$

e o resíduo atualizado a partir da Equação (2.13)

$$\mathbf{r}^{(k+1)} = \mathbf{b}^{(k)} - \mathbf{A}\mathbf{x}^{(k+1)}. \tag{2.21}$$

Daí,

$$\begin{aligned} \mathbf{r}^{(k+1)} &= \mathbf{b}^{(k)} - \mathbf{A}(\mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{r}^{(k)}) \\ \mathbf{r}^{(k+1)} &= (\mathbf{b}^{(k)} - \mathbf{A}\mathbf{x}^{(k)}) - \alpha^{(k)}\mathbf{A}\mathbf{r}^{(k)} \\ \mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - \alpha^{(k)}\mathbf{A}\mathbf{r}^{(k)}. \end{aligned} \tag{2.22}$$

É importante ressaltar que no PMD a principal operação envolvida nas Equações (2.20) e (2.22) é o produto matriz-vetor, o que leva a um custo computacional proporcional [18]. E também, o não uso das aproximações $\mathbf{x}^{(k)}$, na Equação (2.22) pode afetar a convergência, ao acumular erros de arredondamento [18].

Além disso, o método PMD é conhecido também por método Gradiente como resumido no Algoritmo 1, baseado em [4]:

Algoritmo 1 Método Gradiente.

1: Entrada: $\mathbf{A}$, $\mathbf{b}$, $\mathbf{x}^{(0)}$ (solução inicial), N (número máximo de iterações) e tol (tolerância)
2: Saída: solução aproximada $\mathbf{x}^{(k+1)}$
3: Faça: $\mathbf{x}^{(0)} = \mathbf{0}$
4: **para** k = 0, ..., N **faça**
5: $\quad \mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$
6: $\quad \alpha^{(k)} = \dfrac{\left(\mathbf{r}^{(k)}\right)^T \mathbf{r}^{(k)}}{\left(\mathbf{r}^{(k)}\right)^T \mathbf{A}\mathbf{r}^{(k)}}$
7: $\quad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{r}^{(k)}$
8: $\quad$ **se** $(\mathbf{r}^{(k)})^T\mathbf{r}^{(k)} < tol$ **então**
9: $\quad\quad$ **solução** $= \mathbf{x}^{(k+1)}$
10: $\quad$ **fim se**
11: **fim para**

2.1.3 O Método Direções Conjugadas

No método Passo Máximo Descendente escolheu-se a direção de maior declive, ou seja,

$$\mathbf{d}^{(k)} = \mathbf{r}^{(k)}.$$

Uma alteranativa a essa escolha é usar um conjunto de vetores $\left\{\mathbf{d}^{(0)}, \mathbf{d}^{(1)}, ..., \mathbf{d}^{(n)}\right\} \in \mathbb{R}^n - 0$ que sejam ortogonais. Dessa forma, o passo seria associado a essas direções

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)}, \tag{2.23}$$

e $\mathbf{r}^{(k)}$ e $\mathbf{d}^{(k)}$ atenderiam ao critério

$$\begin{aligned}
(\mathbf{d}^{(k)})^T\mathbf{r}^{(k+1)} &= 0 \\
(\mathbf{d}^{(k)})^T\left(\mathbf{r}^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)}\right) &= 0 \\
(\mathbf{d}^{(k)})^T\mathbf{r}^{(k)} + \alpha^{(k)}(\mathbf{d}^{(k)})^T\mathbf{d}^{(k)} &= 0 \\
\alpha^{(k)} &= -\frac{(\mathbf{d}^{(k)})^T\mathbf{r}^{(k)}}{(\mathbf{d}^{(k)})^T\mathbf{d}^{(k)}}.
\end{aligned} \tag{2.24}$$

Como ainda $\alpha^{(k)}$ dependeria de $\mathbf{r}^{(k)}$, então não houve avanço na busca, conforme se esperava ao optar por direções ortogonais.

Um modo de solucionar esse problema é trocar as direções ortogonais por direções

A-conjugadas linearmente independentes. A definição e o lema, a seguir, esclarecem essa decisão.

Definição 2.1.1 *Um conjunto de vetores* $\left\{\mathbf{d}^{(0)}, \mathbf{d}^{(1)}, ..., \mathbf{d}^{(n)}\right\} \in \mathbb{R}^n - 0$ *é dito* **A**-*conjugado, se* $(\mathbf{d}^i)^T \mathbf{A}\mathbf{d}^j = 0$, *para todo* $i, j = 0, 1, ..., n; \quad i \neq j$.

Lema 2.1.1 *Para qualquer matriz* $\mathbf{A} \in \mathbb{R}^{n \times n}$ *simétrica definida positiva, qualquer conjunto de direções* **A**-*conjugadas é linearmente independente.*

Se retomada a ideia inicial de buscar um minimizador da função quadrática $F(\mathbf{x})$, apresentada na Subseção 2.1.1, o teorema 2.1.1 mostra que com as direções **A**-conjugadas isso é possível em n passos.

Teorema 2.1.1 *Seja* F *uma função quadrática. Para qualquer ponto inicial* $\mathbf{x}^{(0)} \in \mathbb{R}^n$, *com qualquer escolha de conjunto de vetores* **A**-*conjugados, o ponto* $\mathbf{x}^{(k)}$ *obtido de acordo com* (2.12) , *para* $k = 0, 1, ..., n-1$, *é o minimizador de* (2.1).

Mais uma vez usa-se a ortogonalidade do resíduo em relação as direções de busca. Entretanto, como tais direções são **A**-conjugadas aproveita-se o resultado do seguinte teorema

Teorema 2.1.1 *O resíduo* $\mathbf{r}^{(k)}$, *para* $k = 1, 2, ..., n$ *de um método de direções conjugadas satisfaz*

$$(\mathbf{r}^{(k)})^T \mathbf{d}^j = 0, \tag{2.25}$$

para todo $j < k$ *e* $j = 1, 2, ..., k$.

Em relação a demonstração desse e dos demais teoremas, bem como lema e definições, recomenda-se os trabalhos de Burden [3] e Saad [15].

Na próxima Seção será discutida a construção do conjunto de direções **A**-conjugadas e o método Gradiente Conjugado, propriamente.

2.1.4 O Método Gradiente Conjugado

Tomando como base o método Direções Conjugadas, para o método Gradiente Conjugado as direções conjugadas serão construídas pela conjugação dos resíduos, que devem ser mutuamente ortogonais.

Assim, toma-se uma solução inicial $\mathbf{x}^{(0)} \in \mathbb{R}^n$ tal que

$$\mathbf{d}^{(0)} = \mathbf{r}^{(0)} \tag{2.26}$$

pelo método PMD; e as aproximações por

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)}, \tag{2.27}$$

em que a próxima direção é

$$\mathbf{d}^{(k+1)} = \mathbf{r}^{(k+1)} + \beta^{(k)}\mathbf{d}^{(k)}. \tag{2.28}$$

A escolha do parâmetro $\beta^{(k)}$ é feita de tal forma que as direções $\mathbf{d}^{(k+1)}$ e $\mathbf{d}^{(k)}$ sejam $\mathbf{A}$-ortogonais. Logo, pela Definição 2.1.1, segue

$$\begin{aligned}
(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{d}^{(k+1)} &= 0 \\
(\mathbf{d}^{(k)})^T\mathbf{A}(\mathbf{r}^{(k+1)} + \beta^{(k)}\mathbf{d}^{(k)}) &= 0 \\
(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{r}^{(k+1)} + \beta^{(k)}(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{d}^{(k)} &= 0 \\
\beta^{(k)} &= -\frac{(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{r}^{(k+1)}}{(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{d}^{(k)}}.
\end{aligned} \tag{2.29}$$

Dessa maneira, o método GC pode ser resumido pelas Equações (2.26), (2.19), (2.22), (2.29) e (2.28), que compõem o Algoritmo 2 utilizado na implementação do referido método.

Algoritmo 2 Método Gradiente Conjugado.

1: Dados de entrada: $\mathbf{A}$, $\mathbf{b}$, $\mathbf{x}^{(0)}$ (solução inicial), N (número máximo de iterações) e tol (tolerância)
2: Faça: $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\,\mathbf{x}^{(0)}$, $\mathbf{d}^{(0)} = \mathbf{r}^{(0)}$ e $k = 0$
3: **enquanto** $k \leq N$ **faça**
4: $\quad \alpha^{(k)} = \dfrac{(\mathbf{r}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{r}^{(k)})^T \mathbf{A}\mathbf{r}^{(k)}}$
5: $\quad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\,\mathbf{d}^{(k)}$
6: $\quad \mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)}\,\mathbf{A}\,\mathbf{d}^{(k)}$
7: $\quad erro = \mathbf{r}^{(k+1)} \cdot \mathbf{r}^{(k+1)}$
8: $\quad$ **se** $erro < tol$ **então**
9: $\quad\quad$ **solução** $= \mathbf{x}^{(k+1)}$
10: $\quad\quad$ break
11: $\quad$ **fim se**
12: $\quad \beta^{(k)} = -\dfrac{(\mathbf{d}^{(k)})^T \mathbf{A}\mathbf{r}^{(k+1)}}{(\mathbf{d}^{(k)})^T \mathbf{A}\mathbf{d}^{(k)}}$
13: $\quad \mathbf{d}^{(k+1)} = \mathbf{r}^{(k+1)} + \beta^{(k)}\,\mathbf{d}^{(k)}$
14: $\quad k = k + 1$
15: **fim enquanto**

Em termos de convergência, o método GC converge em n passos, sendo n a dimensão da matriz $\mathbf{A}$. Mas em casos em que a matriz for mal condicionada, esse método fica suscetível a erros de arredondamento, como afirma Burden [3]. Outra questão, segundo Shewchuk [18], é a sua comum aplicação em problemas grandes de modo que fica inviável sua execução em n passos.

2.2 Precondicionamento

A ideia do precondicionamento vem atrelada ao desejo de determinar de maneira mais simples a solução de um sistema de equações lineares que passe por uma transformação que impacte positivamente sua convergência. Para efetuar tal transformação, toma-se $\mathbf{M}$ como a matriz precondicionadora de modo que, ao associá-la ao sistema linear inicial (2.8), espera-se que o sistema precondicionado seja mais fácil de ser resolvido iterativamente do que o sistema original.

Nesse caso, existem basicamente três maneiras de associar um precondicionador ao sistema linear original. São elas:

(i) "precondicionamento à esquerda":

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}; \tag{2.30}$$

(ii) "precondicionamento à direita":

$$\mathbf{A}\mathbf{M}^{-1}\mathbf{y} = \mathbf{b}, \quad \mathbf{x} = \mathbf{M}^{-1}\mathbf{y}; \tag{2.31}$$

(iii) "precondicionamento misto":

$$\mathbf{M}_1^{-1}\mathbf{A}\mathbf{M}_2^{-1}\mathbf{y} = \mathbf{M}_1^{-1}\mathbf{b}, \quad \mathbf{x} = \mathbf{M}_2^{-1}\mathbf{y}, \tag{2.32}$$

onde $\mathbf{M} = \mathbf{M}_1\mathbf{M}_2$ é a matriz precondicionadora.

Para todos os casos, a principal operação envolvida é o produto $\mathbf{M}^{-1}\mathbf{A}$. Logo, para que a ideia inicial do precondicionamento seja mantida, deve ser possível calcular essa operação. Segundo Benzi [1], quando métodos do subespaço de Krylov são usados, não é necessário calcular os precondicionadores envolvidos nas operações $\mathbf{M}^{-1}\mathbf{A}$ e $\mathbf{A}\mathbf{M}^{-1}$ de maneira explícita. Como alternativa, são executados produtos vetoriais de matrizes com $\mathbf{A}$ e determinadas soluções de sistemas de equações da forma

$$\mathbf{M}\mathbf{z} = \mathbf{r}, \tag{2.33}$$

em que $\mathbf{z}$ é um vetor e $\mathbf{r}$ é o resíduo (2.22). Ou ainda, produtos matriz-vetor com $\mathbf{M}^{-1}$ se for simples de operar de maneira explícita.

Além disso, a escolha do precondicionador depende do método iterativo usado. No caso deste trabalho, como optou-se pelo método GC, deve-se garantir que $\mathbf{M}$ seja igualmente simétrica positiva definida (SPD) como $\mathbf{A}$. Desse modo, assegura-se que para qualquer uma das estratégias de precondicionamento, a convergência desse método é a mesma [1].

Com tudo, não há uma regra geral para a escolha de um bom precondicionador. Aliás, taxá-lo como bom ou eficiente é algo relativo, já que depende do seu desempenho frente as características específicas de um problema, por exemplo. Logo, equilibrar todas essas particularidades faz com que encontrar um bom precondicionador para a resolução de um determinado sistema linear seja visto como uma combinação entre arte e ciência, segundo Saad [15].

No Capítulo 2, apresentam-se os precondicionadores escolhidos com o objetivo de auxiliar na convergência do método GC. Tais precondicionadores são baseados em métodos

iterativos básicos, como Jacobi, Gauss-Seidel, SOR e SSOR, que levam em consideração as propriedades da matriz de coeficientes $\mathbf{A}$. Como sugerido na literatura, neste trabalho optou-se pelo precondicionamento à esquerda. Com isso, espera-se que $\mathbf{M}$ seja suficientemente próxima de $\mathbf{A}$, a ponto de preservar suas propriedades SPD; $\mathbf{M}^{-1}\mathbf{A}$ seja bem condicionada; e consequentemente $\mathbf{Mx} = \mathbf{b}$ seja simples de resolver do que o sistema original [4].

2.3 O Método Gradiente Conjugado Precondicionado

Dado o sistema precondicionado à esquerda a seguir, em relação ao sistema original $\mathbf{Ax} = \mathbf{b}$

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

onde $\mathbf{M}$ é a matriz precondicionadora conforme apresentado na Seção 2.2 anterior. A incorporação de $\mathbf{M}$ nas etapas do GC ocorre de maneira implícita, de acordo com a Equação (2.33).

Desse modo, tomando como base o método das Direções Conjugadas, sabe-se que no GC as direções conjugadas serão construídas pela conjugação dos resíduos, os quais devem ser mutuamente ortogonais [18]. Para isso, considerando uma solução inicial $\mathbf{x}^{(0)} \in \mathbb{R}^n$ tal que

$$\mathbf{d}^{(0)} = \mathbf{r}^{(0)} = \mathbf{b} - \mathbf{Ax}^{(0)}, \tag{2.34}$$

pelo método PMD, e a partir daí

$$\mathbf{z}^{(0)} = \mathbf{M}^{-1}\mathbf{r}^{(0)}, \tag{2.35}$$

e consequentemente

$$\mathbf{d}^{(0)} = \mathbf{z}^{(0)}. \tag{2.36}$$

Daí, as atualizações dessas equações seguem

$$\mathbf{d}^{(k)} = \mathbf{z}^{(k)} \tag{2.37}$$

$$\mathbf{z}^{(k)} = \mathbf{M}^{-1}\mathbf{r}^{(k)}. \tag{2.38}$$

Já as próximas soluções acontecem por

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)}, \tag{2.39}$$

em que o tamanho do passo é

$$\alpha^{(k)} = \frac{\mathbf{r}^{(k)}\mathbf{z}^{(k)}}{\mathbf{d}^{(k)}\mathbf{A}\mathbf{d}^{(k)}}. \tag{2.40}$$

O resíduo é atualizado de forma que

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)}\mathbf{d}^{(k)}. \tag{2.41}$$

E por fim, a próxima direção de busca é

$$\mathbf{d}^{(k+1)} = \mathbf{z}^{(k+1)} + \beta^{(k)}\mathbf{d}^{(k)}, \tag{2.42}$$

onde

$$\mathbf{z}^{(k+1)} = \mathbf{M}^{-1}\mathbf{r}^{(k+1)} \tag{2.43}$$

$$\beta^{(k)} = -\frac{\mathbf{r}^{(k+1)}\mathbf{z}^{(k+1)}}{\mathbf{r}^{(k)}\mathbf{z}^{(k)}}. \tag{2.44}$$

Portanto, as Equações (2.42) à (2.44) usadas na implementação do Algoritmo 3, constituem o método dos Gradientes Conjugados Precondicionados (GCPC).

Algoritmo 3 Método dos Gradientes Conjugados Precondicionado.

1: Dados de entrada: $\mathbf{A}$, $\mathbf{b}$, $\mathbf{x}^{(0)}$ (solução inicial), N (número máximo de iterações) e tol (tolerância)
2: Faça: $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\,\mathbf{x}^{(0)}$, $\mathbf{d}^{(0)} = \mathbf{r}^{(0)}$ e $k = 0$
3: Faça $k = 0$
4: **enquanto** $k \leq N$ **faça**
5: $aux = \mathbf{A}\,\mathbf{d}$
6: $\alpha^{(k)} = \dfrac{\mathbf{r}^{(k)}\mathbf{z}^{(k)}}{\mathbf{d}^{(k)}\mathbf{A}\mathbf{d}^{(k)}}$
7: $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)}$
8: $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)}\mathbf{d}^{(k)}$
9: $\mathbf{z}^{(k+1)} = \mathbf{M}^{-1}\mathbf{r}^{(k+1)}$
10: $erro = \mathbf{r}^{(k+1)} \cdot \mathbf{r}^{(k+1)}$
11: **se** $erro < tol$ **então**
12: **solução** $= \mathbf{x}^{(k+1)}$
13: break
14: **fim se**
15: $\beta^{(k)} = -\dfrac{\mathbf{r}^{(k+1)}\mathbf{z}^{(k+1)}}{\mathbf{r}^{(k)}\mathbf{z}^{(k)}}$
16: $\mathbf{d}^{(k+1)} = \mathbf{z}^{(k+1)} + \beta^{(k)}\mathbf{d}^{(k)}$
17: $k = k + 1$
18: **fim enquanto**

Nota-se que o algoritmo descrito é semelhante ao Algoritmo 2, e sua principal diferença está na presença da inversa da matriz precondicionadora. Observe que, se $\mathbf{M} = \mathbf{I}$, no Algoritmo 3, então retorna-se ao método dos Gradientes Conjugados [15].

Capítulo 3

Precondicionadores e Estrtura de Dados CSR

Algumas estratégias são adotadas quando se trata de melhorar convergência de sistemas de equações lineares esparsos. A primeira delas se relaciona com o condicionamento da matriz **A**, visto que se associada uma matriz mal condicionada ao método dos GC, por exemplo, ocorre a ampliação de erros de arredondamente dificultando sua aplicação [4] e, consequentemente, a perda de certa qualidade na solução encontrada. Nesse cenário, engendram-se os precondicionadores. Enquanto, a segunda estratégia acontece da tentativa de tornar a convergência mais rápida, através de esquemas de armazenamento das matrizes de coeficientes, os quais descartam os elementos não nulos. Esse é o caso da estrtura de dados *Compressed Sparse Row* (CSR), escolhida para estudo.

Nas seções que seguem, são apresentadas os precondicionadores clássicos e suas variações escolhidas: Jacobi, Damped Jacobi, Forward Gauss-Seidel, Backward Gauss-Seidel, Symmetric Gauss-Seidel, Forward Successive Over Relaxation, Backward Successive Over Relaxation e Symmetric Successive Over Relaxation; e a estrtura de dados CSR adotada, bem como sua adaptação aos métodos dos GC e GCPC. Nos estudos de Bertacini [2], Junqueira [12], Saad [15] e Silva [6], é possível encontrar mais detalhes sobre essas técnicas.

3.1 Precondicionadores

A busca e o aprimoramento de métodos iterativos que sejam eficientes, em termos de convergência à solução de um sistema linear, leva a utilização de recursos como as técnicas de precondicionamento e os precondicionadores. Com isso, existem diversas categorias de

precondicionadores, das quais destacam-se os precondicionadores baseado em métodos iterativos clássicos como Jacobi, Gauss-Seidel, SOR e algumas de suas variantes, como observado ao longo desta seção.

De modo geral, esses métodos se baseam na decomposição

$$\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U}, \tag{3.1}$$

esquematizada na Figura 3.1, onde

- $\mathbf{D}$ é a diagonal da matriz $\mathbf{A}$ com $d_{ij} = a_{ij}$, para $i = j$;

- $\mathbf{L}$ é sua componente triangular estritamente inferior, isto é,

$$l_{ij} = \begin{cases} a_{ij}, & \text{se} \quad i > j, \\ 0, & \text{se} \quad i \leq j; \end{cases} . \tag{3.2}$$

- $\mathbf{U}$ é sua componente triangular estritamente superior, ou seja,

$$u_{ij} = \begin{cases} a_{ij}, & \text{se} \quad i < j, \\ 0, & \text{se} \quad i \geq j. \end{cases} . \tag{3.3}$$

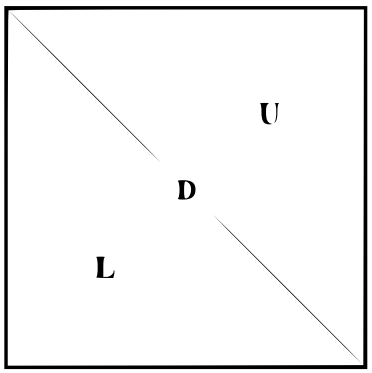


Figura 3.1: Decomposição da matriz $\mathbf{A}$. Adaptada de [15].

Com isso, dado um sistema linear

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

cuja iteração é definida por

$$\mathbf{x}_{k+1} = G\mathbf{x}_k + f, \tag{3.4}$$

em que

$$f = \mathbf{M}^{-1}\mathbf{b} \tag{3.5}$$

e

$$G = \mathbf{M}^{-1}\mathbf{N} \tag{3.6}$$

é baseada na seguinte decomposição de $\mathbf{A}$

$$\mathbf{A} = \mathbf{M} - \mathbf{N}, \tag{3.7}$$

sendo $\mathbf{M}$ uma matriz precondicionadora genérica não singular.

Ao susbttituir as Eq. (3.5) e (3.7) na Eq. (3.4) obtem-se

$$\mathbf{x}_{k+1} = \mathbf{M}^{-1}\mathbf{N}\mathbf{x}_k + \mathbf{M}^{-1}\mathbf{b}, \tag{3.8}$$

que calculará a aproximação da solução a cada iteração da Eq. (2.30).

As demonstrações que se seguem para cada precondicionador, levam em consideração as equações anteriores, e também a proximidade de $\mathbf{M}$ com $\mathbf{A}$ e a facilidade de se calcular $\mathbf{M}$ a cada iteração do sistema precondicionado.

3.1.1 Precondicionador Jacobi

O precondicionador de Jacobi é obitido a partir da decomposição (3.1), ao substituí-la no sistema linear dado:

$$\begin{aligned}
\mathbf{A}\mathbf{x} &= \mathbf{b} \\
(\mathbf{D} + \mathbf{L} + \mathbf{U})\,\mathbf{x} &= \mathbf{b} \\
\mathbf{D}\mathbf{x} + (\mathbf{L} + \mathbf{U})\,\mathbf{x} &= \mathbf{b} \\
\mathbf{D}\mathbf{x} &= \mathbf{b} - (\mathbf{L} + \mathbf{U})\,\mathbf{x} \\
\mathbf{D}\mathbf{x} &= \mathbf{b} + (-\mathbf{L} - \mathbf{U})\,\mathbf{x} \\
\mathbf{x} &= \mathbf{D}^{-1}\,(-\mathbf{L} - \mathbf{U})\,\mathbf{x} + \mathbf{D}^{-1}\mathbf{b}
\end{aligned} \tag{3.9}$$

Ao reescrever a (3.9) como a iteração de Jacobi em sua forma matricial tem-se

$$\mathbf{x}_{k+1} = \mathbf{D}^{-1}\,(-\mathbf{L} - \mathbf{U})\,\mathbf{x}_k + \mathbf{D}^{-1}\mathbf{b}. \tag{3.10}$$

a qual assemelha-se a equação (3.8), de modo que $\mathbf{M}^{-1} = \mathbf{D}^{-1}$ e $\mathbf{N} = -\mathbf{L} - \mathbf{U}$. Logo, a matriz precondicionadora de Jacobi é dada por

$$\mathbf{M}_{Ja} = \mathbf{D}, \tag{3.11}$$

e, consequentemente, $\mathbf{N} = \mathbf{D} - \mathbf{A}$.

Esse precondicionador é relativamente simples de ser obtido e implementado, já que preserva as entradas da diagonal da matriz $\mathbf{A}$, como visto no Algoritmo 4. Além disso, a inversão desta matriz precondicionadora acaba sendo igualmente simples por manter certa semelhança com a matriz original, como em casos de matrizes diagonalmente dominantes [17]. Em especial, a inversa da matriz diagonal é

$$\mathbf{M}_{Ja}^{-1} = \mathbf{D}^{-1} = \left[\frac{1}{d_{ii}}\right] = \left[\frac{1}{a_{ii}}\right]. \tag{3.12}$$

Algoritmo 4 Precondicionador de Jacobi

```
Entrada: A
para i = 1, ..., n faça
    para j = 1, ..., n faça
        se i = j então
            M_Ja = a_ij
            senão
            M_Ja = 0
        fim se
    fim para
fim para
```

3.1.2 Precondicionador Damped Jacobi

O precondicionador Damped Jacobi é uma variação do método de Jacobi, e consequentemente, do precondicionador de mesmo nome apresentado na seção anterior. Para sua constituição considera-se um parâmetro ω na decomposição (3.1) da matriz A, de modo que:

$$\begin{aligned}
\mathbf{A}\mathbf{x} &= \mathbf{b} \\
\omega\left(\mathbf{D}+\mathbf{L}+\mathbf{U}\right)\mathbf{x} &= \omega\mathbf{b} \\
\omega\mathbf{D}\mathbf{x}+\omega\left(\mathbf{L}+\mathbf{U}\right)\mathbf{x} &= \omega\mathbf{b} \\
\omega\mathbf{D}\mathbf{x} &= \omega\mathbf{b}-\omega\left(\mathbf{L}+\mathbf{U}\right)\mathbf{x} \\
\mathbf{x} &= \omega^{-1}\mathbf{D}^{-1}\left(-\mathbf{L}-\mathbf{U}\right)\mathbf{x}+\omega^{-1}\mathbf{D}^{-1}\mathbf{b} \\
\mathbf{x} &= \frac{1}{\omega}\mathbf{D}^{-1}\left(-\mathbf{L}-\mathbf{U}\right)\mathbf{x}+\frac{1}{\omega}\mathbf{D}^{-1}\mathbf{b}
\end{aligned} \tag{3.13}$$

Nota-se que a equação (3.13) é análoga a equação (3.8). Dessa forma, sua reescrita na forma vetorial é considerada como a iteração de Damped Jacobi:

$$\mathbf{x}_{k+1} = \frac{1}{\omega}\mathbf{D}^{-1}\left(-\mathbf{L}-\mathbf{U}\right)\mathbf{x}_k + \frac{1}{\omega}\mathbf{D}^{-1}\mathbf{b}, \tag{3.14}$$

onde $\mathbf{M}^{-1} = \frac{1}{\omega}\mathbf{D}^{-1}$ e $\mathbf{N} = -\mathbf{L}-\mathbf{U}$. Assim, o precondicionador Damped Jacobi é dado por

$$\mathbf{M}_{DJa} = \omega\mathbf{D}, \tag{3.15}$$

onde ω é um escalar, tal que $\omega \in (0,2)$. Nota-se que para $\omega = 1$, retoma-se ao precondicionador de Jacobi.

O Algoritmo 5 expõe a construção dessa variante:

Algoritmo 5 Precondicionador de Damped Jacobi

1: Entrada: $n, \mathbf{A}, \omega$
2: **para** $i = 1, ..., n$ **faça**
3: **para** $j = 1, ..., n$ **faça**
4: $\mathbf{M}_{DJa} = \omega\mathbf{D}$
5: **senão**
6: $\mathbf{M}_{DJa} = 0$
7: **fim para**
8: **fim para**

3.1.3 Precondicionador *Forward Gauss-Seidel* (FGS)

O primeiro dos precondicionadores baseados no método Gauss-Seidel, o *Forward Gauss-Seidel* - FGS é igualmente obtido considerando a decomposição (3.1) de forma que

$$\begin{aligned}
\mathbf{A}\mathbf{x} &= \mathbf{b} \\
\left(\mathbf{D}+\mathbf{L}+\mathbf{U}\right)\mathbf{x} &= \mathbf{b} \\
\left(\mathbf{D}+\mathbf{L}\right)\mathbf{x} + \mathbf{U}\mathbf{x} &= \mathbf{b} \\
\left(\mathbf{D}+\mathbf{L}\right)\mathbf{x} &= -\mathbf{U}\mathbf{x} + \mathbf{b} \\
\mathbf{x} &= \left(\mathbf{D}+\mathbf{L}\right)^{-1}\left(-\mathbf{U}\right)\mathbf{x} + \left(\mathbf{D}+\mathbf{L}\right)^{-1}\mathbf{b}
\end{aligned} \tag{3.16}$$

Novamente, ao considerar a equação (3.8) e compará-la a equação (3.16), a iteração de

Gauss-Seidel em sua forma matricial é dada por

$$\mathbf{x}_{k+1} = (\mathbf{D} + \mathbf{L})^{-1} (-\mathbf{U})\mathbf{x}_k + (\mathbf{D} + \mathbf{L})^{-1} \mathbf{b}. \tag{3.17}$$

Daí, $\mathbf{M}^{-1} = (\mathbf{D} + \mathbf{L})^{-1}$ e $\mathbf{N} = -\mathbf{U}$. Com isso, o precondicionador FGS é definido por

$$\mathbf{M}_{FGS} = \mathbf{D} + \mathbf{L}, \tag{3.18}$$

ou seja, é a parte inferior da matriz $\mathbf{A}$, incluindo sua diagonal.

O Algoritmo 6 evidência a construção da matriz precondicionadora FGS e sua associação a uma matriz, que leva a resolução de um sistema triangular inferior.

Algoritmo 6 Precondicionador FGS

```
1: Entrada: n, A
2: para i = 1, ..., n faça
3:    para j = 1, ..., n faça
4:       A = D + L + U
5:    fim para
6: fim para
7: para i = 1, ..., n faça
8:    para j = 1, ..., n faça
9:       M_FGS = D + L
10:   fim para
11: fim para
```

Em termos de convergência, os precondicionadores de Gauss-Seidel e Jacobi convergem para matrizes diagonalmente dominante e/ou simétricas positivas definidas, segundo Cunha [4]. Esses critérios foram estabelecidos por meio dos autovalores da matriz de iteração $\mathbf{G}$, como abordado por Saad ao generalizar uma iteração dos métodos de mesma denominação. Com isso, recomenda-se a leitura de [15] para maiores detalhes sobre teoremas e demonstrações dessas condições.

3.1.4 Precondicionador *Backward Gauss-Seidel* (BGS)

A segunda variante de Gauss-Seidel, é o precondicionador *Backward Gauss-Seidel* - BGS, o qual adota a decomposição (3.1) de forma similar aos demais aqui já apresentados.

Assim, tem-se

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ (\mathbf{D}+\mathbf{L}+\mathbf{U})\,\mathbf{x} &= \mathbf{b} \\ (\mathbf{D}+\mathbf{U})\,\mathbf{x}+\mathbf{L}\mathbf{x} &= \mathbf{b} \\ (\mathbf{D}+\mathbf{U})\,\mathbf{x} &= -\mathbf{L}\mathbf{x}+\mathbf{b} \\ \mathbf{x} &= (\mathbf{D}+\mathbf{U})^{-1}(-\mathbf{L})\mathbf{x}+(\mathbf{D}+\mathbf{U})^{-1}\mathbf{b} \end{aligned} \tag{3.19}$$

Analogamente, uma iteração vetorial *Backward Gauss-Seidel* - BGS é definida por

$$\mathbf{x}_{k+1} = (\mathbf{D}+\mathbf{U})^{-1}(-\mathbf{L})\mathbf{x}_k + (\mathbf{D}+\mathbf{U})^{-1}\mathbf{b}, \tag{3.20}$$

de maneira que $\mathbf{M}^{-1} = (\mathbf{D}+\mathbf{U})^{-1}$ e $\mathbf{N} = -\mathbf{L}$ e, consequentemente,

$$\mathbf{M}_{BGS} = \mathbf{D}+\mathbf{U}, \tag{3.21}$$

é o precondicionador BGS. Além disso, a matriz $\mathbf{M}_{BGS}$ é a parte superior da matriz $\mathbf{A}$ contendo sua digonal. Dessa forma, no Algoritmo 7 resolve-se um sistema triangular superior.

Algoritmo 7 Precondicionador BGS

1: Entrada: $n, \mathbf{A}$
2: **para** $i = 1, ..., n$ **faça**
3: **para** $j = 1, ..., n$ **faça**
4: $\mathbf{A} = \mathbf{D}+\mathbf{L}+\mathbf{U}$
5: **fim para**
6: **fim para**
7: **para** $i = 1, ..., n$ **faça**
8: **para** $j = 1, ..., n$ **faça**
9: $\mathbf{M}_{BGS} = \mathbf{D}+\mathbf{U}$
10: **fim para**
11: **fim para**

3.1.5 Precondicionador *Symmetric Gauss-Seidel* (SGS)

A última variante de Gauss-Seidel a ser considerada é o precondicionador *Symmetric Gauss-Seidel* - SGS, originado da combinação da decomposição (3.1) da matriz $\mathbf{A}$ com os

efeitos dos precondicionadores FGS e BGS. Assim,

$$\begin{aligned} \mathbf{M}_{SGS} &= \mathbf{M}_{BGS} \cdot \mathbf{M}_{FGS} \\ \mathbf{M}_{SGS} &= (\mathbf{D} - \mathbf{E})\,\mathbf{D}^{-1}\,(\mathbf{D} - \mathbf{F})\,, \end{aligned} \tag{3.22}$$

agrupando a parte inferior, diagonal e parte superior de $\mathbf{A}$. De outro modo, interpreta-se a equação (3.22) como

$$\mathbf{M}_{SGS} = \mathbf{LU}, \tag{3.23}$$

onde

$$\mathbf{L} = (\mathbf{D} - \mathbf{E})\,\mathbf{D}^{-1} = \mathbf{I} - \mathbf{L}\mathbf{D}^{-1}, \tag{3.24}$$

e

$$\mathbf{L} = \mathbf{D} - \mathbf{U}. \tag{3.25}$$

A aplicação do Algoritmo 8 é recomendada em problemas simétricos.

Algoritmo 8 Precondicionador SGS

1: Entrada: $n, \mathbf{A}$
2: **para** $i = 1, ..., n$ **faça**
3: **para** $j = 1, ..., n$ **faça**
4: $\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}$
5: **fim para**
6: **fim para**
7: **para** $i = 1, ..., n$ **faça**
8: **para** $j = 1, ..., n$ **faça**
9: $\mathbf{M}_{SGS} = (\mathbf{D} - \mathbf{E})\,\mathbf{D}^{-1}\,(\mathbf{D} - \mathbf{F})$
10: **fim para**
11: **fim para**

Em particular, o precondicionador SGS é um caso específico do precondicionador SSOR, tomando $\omega = 1$. Na Subseção 3.1.8 apresenta-se mais detalhes a respeito desse último.

3.1.6 Precondicionador *Forward Successive Over Relaxation* (FSOR)

Tal qual os precondicionadores de Jacobi e as variantes de Gauss-Seidel, o preocondicionador *Forward Successive Over Relaxation* - FSOR devira de um método iterativo. Neste caso, tal método é o SOR (*Successive Over Relaxation*), no qual ocorre a *over relaxation* ou sobre-relaxação ao inseir na decomposição inicial (3.1) um escalar ω, denominado parâmetro de sobre-relaxação:

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \omega\left(\mathbf{D}+\mathbf{L}+\mathbf{U}\right)\mathbf{x} &= \omega\mathbf{b} \\ \omega\left(\mathbf{D}+\mathbf{U}\right)\mathbf{x}+\omega\mathbf{L}\mathbf{x} &= \omega\mathbf{b} \\ \omega\mathbf{L}\mathbf{x} &= \omega\mathbf{b}-\omega\left(\mathbf{D}+\mathbf{U}\right)\mathbf{x}. \end{aligned} \tag{3.26}$$

Acrescentando $\mathbf{D}\mathbf{x}$ na equação (3.26), tem-se:

$$\begin{aligned} \mathbf{D}\mathbf{x}+\omega\mathbf{L}\mathbf{x} &= \mathbf{D}\mathbf{x}-\omega\left(\mathbf{D}+\mathbf{U}\right)\mathbf{x}+\omega\mathbf{b} \\ \left(\mathbf{D}+\omega\mathbf{L}\right)\mathbf{x} &= \left[\mathbf{D}-\omega\left(\mathbf{D}+\mathbf{U}\right)\right]\mathbf{x}+\omega\mathbf{b} \\ \left(\mathbf{D}+\omega\mathbf{L}\right)\mathbf{x} &= \left[\mathbf{D}-\omega\mathbf{D}-\omega\mathbf{U}\right]\mathbf{x}+\omega\mathbf{b} \\ \left(\mathbf{D}+\omega\mathbf{L}\right)\mathbf{x} &= \left[-\omega\mathbf{U}+(1-\omega)\mathbf{D}\right]\mathbf{x}+\omega\mathbf{b} \end{aligned} \tag{3.27}$$

Dividindo a equação (3.27) por ω, segue:

$$\frac{1}{\omega}\left(\mathbf{D}+\omega\mathbf{L}\right)\mathbf{x}=\frac{1}{\omega}\left[-\omega\mathbf{U}+(1-\omega)\mathbf{D}\right]\mathbf{x}+\mathbf{b} \tag{3.28}$$

A partir daí, escreve-se a iteração de FSOR como

$$\frac{1}{\omega}\left(\mathbf{D}+\omega\mathbf{L}\right)\mathbf{x}_{k+1}=\frac{1}{\omega}\left[-\omega\mathbf{U}+(1-\omega)\mathbf{D}\right]\mathbf{x}_{k}+\mathbf{b}. \tag{3.29}$$

$$\mathbf{x}_{k+1}=\omega(\mathbf{D}+\omega\mathbf{L})^{-1}\frac{1}{\omega}\left[-\omega\mathbf{U}+(1-\omega)\mathbf{D}\right]\mathbf{x}_{k}+\omega(\mathbf{D}+\omega\mathbf{L})^{-1}\mathbf{b}. \tag{3.30}$$

Ao comparar as equações (3.30) e (3.8), toma-se

$$\mathbf{M}^{-1}=\omega(\mathbf{D}+\omega\mathbf{L})^{-1}, \tag{3.31}$$

de modo que o precondicionador FSOR é dado por

$$\mathbf{M}_{FSOR} = \frac{1}{\omega}\left(\mathbf{D} + \omega\mathbf{L}\right). \tag{3.32}$$

A construção dessa matriz precondicionadora é feita como mostra o Algoritmo 9.

Algoritmo 9 Precondicionador FSOR

1: Entrada: $n, \mathbf{A}, \omega$
2: **para** $i = 1, ..., n$ **faça**
3: **para** $j = 1, ..., n$ **faça**
4: $\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U}$
5: **fim para**
6: **fim para**
7: **para** $i = 1, ..., n$ **faça**
8: **para** $j = 1, ..., n$ **faça**
9: $\mathbf{M}_{FSOR} = \frac{1}{\omega}\left(\mathbf{D} + \omega\mathbf{L}\right)$
10: **fim para**
11: **fim para**

O fator de relaxação ω interfere propositalmente na eficiência do método associado ao precondicionador FSOR, de modo que é necessário garantir a sua convergência. Dessa maneira, como escolheu-se como objeto de estudo o método GCPC, que impõe a matriz $\mathbf{A}$ ser simétrica positiva definida, é possível mostrar que FSOR converge neste caso para qualquer estimativa inicial, ao tomar $\omega \in (0, 2)$. Para maiores detalhes à respeito da demonstração dessa prova, recomenda-se [15].

3.1.7 Precondicionador *Backward Successive Over Relaxation* (BSOR)

Esta nova versão da cetegoria de precondicionadores SOR é elaaorada de maneira análoga a precondicionador FSOR apresentado na seção anterior. Em visto disso, considera-se:

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \omega\left(\mathbf{D} + \mathbf{L} + \mathbf{U}\right)\mathbf{x} &= \omega\mathbf{b} \\ \omega\left(\mathbf{D} + \mathbf{L}\right)\mathbf{x} + \omega\mathbf{U}\mathbf{x} &= \omega\mathbf{b} \\ \omega\mathbf{U}\mathbf{x} &= \omega\mathbf{b} - \omega\left(\mathbf{D} + \mathbf{L}\right)\mathbf{x}. \end{aligned} \tag{3.33}$$

Acrescentando $\mathbf{Dx}$ na equação (3.33), tem-se:

$$\begin{aligned} \mathbf{Dx} + \omega\mathbf{Ux} &= \mathbf{Dx} - \omega\left(\mathbf{D}+\mathbf{L}\right)\mathbf{x} + \omega\mathbf{b} \\ \left(\mathbf{D}+\omega\mathbf{U}\right)\mathbf{x} &= \left[\mathbf{D} - \omega\left(\mathbf{D}+\mathbf{L}\right)\right]\mathbf{x} + \omega\mathbf{b} \\ \left(\mathbf{D}+\omega\mathbf{U}\right)\mathbf{x} &= \left[\mathbf{D} - \omega\mathbf{D} - \omega\mathbf{L}\right]\mathbf{x} + \omega\mathbf{b} \\ \left(\mathbf{D}+\omega\mathbf{U}\right)\mathbf{x} &= \left[-\omega\mathbf{L} + (1-\omega)\mathbf{D}\right]\mathbf{x} + \omega\mathbf{b} \end{aligned} \tag{3.34}$$

Dividindo a equação (3.34) por ω, segue:

$$\frac{1}{\omega}\left(\mathbf{D}+\omega\mathbf{U}\right)\mathbf{x} = \frac{1}{\omega}\left[-\omega\mathbf{L} + (1-\omega)\mathbf{D}\right]\mathbf{x} + \mathbf{b} \tag{3.35}$$

Agora, adota-se a iteração *Backward Successive Over Relaxation* - BSOR por

$$\frac{1}{\omega}\left(\mathbf{D}+\omega\mathbf{U}\right)\mathbf{x}_{k+1} = \frac{1}{\omega}\left[-\omega\mathbf{L} + (1-\omega)\mathbf{D}\right]\mathbf{x}_k + \mathbf{b}. \tag{3.36}$$

$$\mathbf{x}_{k+1} = \omega(\mathbf{D}+\omega\mathbf{U})^{-1}\frac{1}{\omega}\left[-\omega\mathbf{L} + (1-\omega)\mathbf{D}\right]\mathbf{x}_k + \omega(\mathbf{D}+\omega\mathbf{U})^{-1}\mathbf{b}. \tag{3.37}$$

Ao relacionar as equações (3.37) e (3.8), designa-se o precondicionador BSOR como

$$\mathbf{M}_{BSOR} = \frac{1}{\omega}\left(\mathbf{D}+\omega\mathbf{U}\right). \tag{3.38}$$

O Algoritmo 10 evidencia a construção dessa ferramenta.

Algoritmo 10 Precondicionador BSOR

1: Entrada: $n, \mathbf{A}, \omega$
2: **para** $i = 1, ..., n$ **faça**
3: **para** $j = 1, ..., n$ **faça**
4: $\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U}$
5: **fim para**
6: **fim para**
7: **para** $i = 1, ..., n$ **faça**
8: **para** $j = 1, ..., n$ **faça**
9: $\mathbf{M}_{BSOR} = \frac{1}{\omega}\left(\mathbf{D}+\omega\mathbf{U}\right)$
10: **fim para**
11: **fim para**

Para esta variante, o fator de relaxação $\omega \in (0, 2)$, analogamente ao que foi apresen-

tado e considerado na Seção 3.1.6, no que se refere a convergência.

3.1.8 Precondicionador *Symmetric Successive Over Relaxation* (SSOR)

A *Symmetric Successive Over Relaxation* - SSOR é a variante do precondicionador SOR, comumente utilizada para preservar a simetria de um problema inicial, de modo que permita o emprego de métodos numéricos de característica similar, como o Gradiente Conjugado. Desse modo, considera-se novamente a decomposição (3.1) e o passo iterativo como a combinação das iterações de FSOR seguido de BSOR.

Assim, associando respectivamente as equações (3.29) e (3.36) tem-se

$$\frac{1}{\omega}\left(\mathbf{D}-\omega\mathbf{L}\right)\mathbf{x}_{k+1/2} = \frac{1}{\omega}\left[\omega\mathbf{F}+(1-\omega)\mathbf{D}\right]\mathbf{x}_k+\mathbf{b} \tag{3.39}$$

$$\frac{1}{\omega}\left(\mathbf{D}-\omega\mathbf{U}\right)\mathbf{x}_{k+1} = \frac{1}{\omega}\left[\omega\mathbf{E}+(1-\omega)\mathbf{D}\right]\mathbf{x}_{k+1/2}+\mathbf{b}. \tag{3.40}$$

As iterações anteriores levam à recorrência

$$\mathbf{x}_{k+1} = G_\omega \mathbf{x}_k + f_\omega, \tag{3.41}$$

onde

$$\begin{aligned} G_\omega &= \mathbf{M}^{-1}\mathbf{N} \\ &= \left(\mathbf{D}-\omega\mathbf{U}\right)^{-1}\left[\omega\mathbf{L}+(1-\omega)\mathbf{D}\right]\times\left(\mathbf{D}-\omega\mathbf{L}\right)^{-1}\left[\omega\mathbf{U}+(1-\omega)\mathbf{D}\right], \end{aligned} \tag{3.42}$$

e

$$\begin{aligned} f_\omega &= \mathbf{M}^{-1}\mathbf{b} \\ &= \omega(\mathbf{D}-\omega\mathbf{L})^{-1}\left(I+\left[\omega\mathbf{U}+(1-\omega)\mathbf{D}\right]\left(\mathbf{D}-\omega\mathbf{L}\right)^{-1}\right)\mathbf{b} && (3.43)\\ &= \omega(\mathbf{D}-\omega\mathbf{U})^{-1}\left(I-I+(2-\omega)\mathbf{D}(\mathbf{D}-\omega\mathbf{L})^{-1}\right)\mathbf{b} && (3.44)\\ &= \omega(2-\omega)(\mathbf{D}-\omega\mathbf{U})^{-1}\mathbf{D}(\mathbf{D}-\omega\mathbf{L})^{-1}\mathbf{b} && (3.45) \end{aligned}$$

Multiplicando f_ω por $\mathbf{M}$, tem-se

$$\mathbf{M}f_\omega = b, \tag{3.46}$$

e, consequentemente, a equação (3.46) é reescrita por

$$\frac{1}{\omega(2-\omega)}\left(\mathbf{D}-\omega\mathbf{U}\right)\mathbf{D}^{-1}\left(\mathbf{D}-\omega\mathbf{L}\right)f_\omega = b. \tag{3.47}$$

Logo, a matriz precondicionadora SSOR é definida por

$$\mathbf{M}_{SSOR} = \frac{1}{\omega(2-\omega)} \left(\mathbf{D} - \omega\mathbf{L}\right) \mathbf{D}^{-1} \left(\mathbf{D} - \omega\mathbf{U}\right), \quad \omega \in (0,2), \tag{3.48}$$

e sua composição é realizada pelo Algoritmo 11.

Algoritmo 11 Precondicionador SSOR

Entrada: $n, \mathbf{A}, \omega$
para $i = 1, ..., n$ **faça**
  para $j = 1, ..., n$ **faça**
    $\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}$
  fim para
fim para
para $i = 1, ..., n$ **faça**
  para $j = 1, ..., n$ **faça**
    $\mathbf{M}_{SSOR} = \frac{1}{\omega(2-\omega)} \left(\mathbf{D} - \omega\mathbf{L}\right) \mathbf{D}^{-1} \left(\mathbf{D} - \omega\mathbf{U}\right)$
  fim para
fim para

Como citado na Seção 3.1.5, ao tomar $\omega = 1$, reduz-se a matriz precondicionadora SSOR ao precondicionador SGS. Ambos, conservam a simetria da matriz $\mathbf{A}$ associada.

3.2 Estrutura de Dados CSR

Enquanto o domínio dos sistemas de equações, os quais pretende-se solucionar pelo método do GC e GCPC, envolver dezenas ou centenas de váriáveis, a maneira de manipular e acessar as entradas na matriz **A** pode ser feita através dos índices de suas linhas e colunas. Isto é, o elemento a_{ij} é acessado ao verificar a posição i de sua linha e a posição j de sua coluna. Dessa forma, é possível determinar qualquer elemento da matriz de coeficientes, sejam esses diferentes de zero ou não.

Entretanto, ao lidar com sistemas esparsos e/ou de grande porte, além de tratar de dezenas e centenas de milhares de variáveis, parte desses dados são elementos nulos. É justamente a grande porcentagem de elementos nulos presentes que define simplificadamente uma matriz esparsa. Dessa perspectiva, o uso de uma estrturua de dados é necessária, ao pensar em não armazenar esses elementos zero. Tal estrutura teria como principal objetivo armazenar os elementos diferentes de zero e, ainda assim, realizar operações comuns para as matrizes, como soma e produto, por exemplo.

Posto o principal objetivo das estruturas de dados, agora é necessário estabelecer um critério para o esquema de armazenamento desses elementos não nulos. São inúmeros critérios estabelecidos, armazenamentos por coordenadas, linhas, colunas, diagonais, entre outros. Para cada critério são identificadas as diferentes estrturas, das quais escolheu-se para estudo o formato *Compressed Sparse Row* (CSR), tido como um dos mais populares esquemas de armazenamento de matrizes esparsas, segundo Saad.

A estrutura CSR armazena os elementos diferentes de zero percorrendo de maneira ordenada linha a linha da matriz. O armazenamento desses elementos e suas posições ocorre em três vetores:

i) **aa** é o vetor que contém os elementos não nulos reais da matrizes de coeficientes, ou seja, $a_{ij} \neq 0$, para a linha variando de 1 até a dimensão n de matriz. Sua dimensão é a quantidade de elementos não nulos, denotada por nz.

ii) **jr** é o vetor que contém os índices inteiros das colunas de cada elemento não nulo a_{ij} da matrizes de coeficientes guardado em **aa**. Esse vetor tem o tamanho de nz.

iii) **jc** é o vetor de ponteiros que contém a posição inteira do primeiro elemento não nulo de cada linha da matriz de coeficientes no vetor **aa**. O tamanho desse vetor é $n + 1$, sendo a sua posição $\mathbf{jc(n + 1)}$ igual ao número $\mathbf{jc(1)} + 1$.

Para melhor compreensão, o exemplo seguinte executa a composição dos três vetores da estrtura CSR. Para isso, toma-se uma das matrizes utilizadas posteriormente no Capítulo 4, na Seção 4.2:

$$\mathbf{A} = \begin{bmatrix} 2.0 & 0 & 0.5 & 0 & 0.25 \\ 0 & 4.0 & 0 & 1.0 & 0 \\ 1.5 & 0 & 6.0 & 0 & 1.5 \\ 0 & 2.0 & 0 & 8.0 & 0 \\ 1.25 & 0 & 2.5 & 0 & 10 \end{bmatrix}$$

Como forma de facilitar a visualização de cada elemento e suas posições, anota-se os índices de linhas e colunas, que caminham de 1 até 5, como no esquema abaixo.

	1	**2**	**3**	**4**	**5**
1	2.0	0	0.5	0	0.25
2	0	4.0	0	1.0	0
3	1.5	0	6.0	0	1.5
4	0	2.0	0	8.0	0
5	1.25	0	2.5	0	10

Tomando o elemento a_{11} como início para percorrer a linha 1 até a linha 5 da matriz, todos os 13 elementos não nulos são inseridos no vetor **aa**, cuja dimensão é $nz = 13$, como segue:

$$\mathbf{aa} = \boxed{\begin{array}{ccccccccccccc} 2.0 & 0.5 & 0.25 & 4.0 & 1.0 & 1.5 & 6.0 & 1.5 & 2.0 & 8.0 & 1.25 & 2.5 & 10.0 \end{array}}$$

Agora observando apenas os componentes de **aa**, em **jr** localizam-se cada um dos índices das colunas, na respectiva ordem apresentada dos elementos não zero.

$$\mathbf{jr} = \boxed{\begin{array}{ccccccccccccc} 1 & 3 & 5 & 2 & 4 & 1 & 3 & 5 & 2 & 4 & 1 & 3 & 5 \end{array}}$$

Por fim, o ponteiro **jc** indica a posição do elemento incial de cada linha da matriz. Nota-se que na linha 1 o primeiro elemento não nulo é $a_{11} = 1.0$, enquanto na linha 2 esse dado é $a_{22} = 4.0$; e assim para cada linha que se segue. É importante dizer que a última posição de **jc**, ou seja, $n + 1 = 5 + 1 = 6$ recebe o valor de $nz + 1 = 13 + 1 = 14$.

$$\mathbf{jc} = \boxed{\begin{array}{cccccc} 1 & 4 & 6 & 9 & 11 & 14 \end{array}}$$

Além disso, segundo Rodrigues [14], a diferença entre as posições $\mathbf{jc(i+1)}$ e $\mathbf{jc(i)}$, para $i = 1, ...n$, possibilita determinar o número de elementos não nulos na i-ésima linha da

matriz. Observa-se para o exemplo dado que a linha 1 tem 3 elementos não nulos. Ao calcular a diferença entre $\mathbf{jc}(\mathbf{2}) = 4$ e $\mathbf{jc}(\mathbf{1}) = 1$, obtém-se 3 que é justamente a quantidade indicada antes.

3.2.1 Estrutura CSR e o método dos GC e GCPC

Após compreender o esquema de armazenamento que será utilizado no tratamento das matrizes analisadas neste trabalho, é fundamental garantir que as operações envolvidas nos métodos iterativos escolhidos (GC e GCPC) e nos precondicionadores adotados, sejam mantidas dentro da concepção de cada um. A principal operação envolvida é o produto matriz-vetor, presente principalmente no métodos dos GC e GCPC.

Para a adaptação dessa operação, tomou-se como base a rotina apresentada em [14] e adaptada conforme o Algoritmo 12:

Algoritmo 12 Produto matriz simétrica-vetor CSR.

1: Dados de entrada: n, **jr**, **jc**, **aa**, **vet**
2: Saída: vetor **ResultProd** resultante do produto matriz-vetor **vet**
3: **para** $j = 1, ..., n$ **faça**
4: $\quad$ **ResultProd**$[j] = 0$
5: $\quad$ **para** $k = jc[j], ..., jc[j+1] - 1$ **faça**
6: $\quad\quad$ **ResultProd**$[j]$ = **ResultProd**$[j] + aa[k] * vet[jr(k)]$
7: $\quad\quad$ **se** jr[k] $<$ j **então**
8: $\quad\quad\quad$ **ResultProd**$[j]$ = **ResultProd**$[jr(k)] + aa[k] * vet[j]$
9: $\quad\quad$ **fim se**
10: $\quad$ **fim para**
11: **fim para**

Algoritmo 13 Método dos GC adaptado com CSR.

1: Dados de entrada: arrumar
2: Leitura do arquivo de entrada da matriz **A**
3: Ordenação do arquivo lido
4: **para** $j = 1, ..., n$ **faça**
5: **para** $i = jc[j], ..., jc[j+1] - 1$ **faça**
6: $r^0[j] = b[j] - aa[i] * x^0[j]$
7: **fim para**
8: **fim para**
9: **para** $i = 1, ..., n$ **faça**
10: **para** $j = 1, ..., n$ **faça**
11: $z^0[i] = r^0[j]$
12: **fim para**
13: **fim para**
14: **para** $i = 1, ..., n$ **faça**
15: $z^0[i] = r^0[i]$
16: $d^0[i] = z^0[i]$
17: **fim para**
18: $k = 0$
19: **enquanto** $k \leq N$ **faça**
20: **para** $i = 1, ..., n$ **faça**
21: **para** $j = jc[j], ..., jc[j+1] - 1$ **faça**
22: $aux[i] = aa[j] * d^0[jr[j]]$
23: **fim para**
24: $\alpha = \dfrac{r^0[i] * d^0[i]}{d^0[i] * aux[i]}$
25: $x[i] = x^0[i] + \alpha * d^0[i]$
26: $r[i] = r^0[i] + \alpha * aux[i]$
27: $z[i] = r[i]$
28: $erro = r[i] * r[i]$
29: **se** erro $<$ tol **então**
30: **solução** $= x[i]$
31: $k = k + 1$
32: break
33: **fim se**
34: $\beta = \dfrac{r[i] * z[i]}{r^0[i] * z^0[i]}$
35: $d[i] = z[i] + \beta * d^0[i]$
36: $k = k + 1$
37: **fim para**
38: **fim enquanto**

Algoritmo 14 Método dos GCPC adaptado com CSR.

Dados de entrada: arrumar
Leitura do arquivo de entrada da matriz **A**
Ordenação do arquivo lido
para $j = 1, ..., n$ **faça**
  para $i = jc[j], ..., jc[j+1] - 1$ **faça**
    $r^0[j] = b[j] - aa[i] * x^0[j]$
  fim para
fim para
inversa $= \mathbf{M}^{-1}$
para $i = 1, ..., n$ **faça**
  para $j = 1, ..., n$ **faça**
    $z^0[i] = \mathbf{M}^{-1} * r^0[j]$
  fim para
fim para
para $i = 1, ..., n$ **faça**
  $z^0[i] = r^0[i]$
  $d^0[i] = z^0[i]$
fim para
$k = 0$
enquanto $k \leq N$ **faça**
  para $i = 1, ..., n$ **faça**
    para $j = jc[j], ..., jc[j+1] - 1$ **faça**
      $aux[i] = aa[j] * d^0[jr[j]]$
    fim para
    $\alpha = \dfrac{r^0[i] * d^0[i]}{d^0[i] * aux[i]}$
    $x[i] = x^0[i] + \alpha * d^0[i]$
    $r[i] = r^0[i] + \alpha * aux[i]$
    $z[i] = r[i]$
    erro $= r[i] * r[i]$
    se erro $<$ tol **então**
      solução $= x[i]$
      $k = k + 1$
      break
    fim se
    $\beta = \dfrac{r[i] * z[i]}{r^0[i] * z^0[i]}$
    $d[i] = z[i] + \beta * d^0[i]$
    $k = k + 1$
  fim para
fim enquanto

Capítulo 4

Resultados Numéricos

Neste Capítulo primeiramente apresentam-se as especificações das simulações realizadas para os métodos GC e GCPC, com e sem estrtura de dados CSR. Em seguida, são postos os problemas analisados nos testes computacionais. E, finalmente, são expostos os resultados numéricos e suas análises.

4.1 Especificações das simulações

Todas os resultados das simulações foram realizados em um notebook com processador Intel Core i5-7200 de 2.5 GHz, 8 GB de memória RAM e sistema operacional de 64bits - Windows 10 Home Single Language. Os programas foram tratados no ambiente do compilador DevC++, versão 5.11. Nessa implementações, as matrizes e vetores foram alocadas dinamicamente, devido a sua dimensão e sua quantidade de elementos não nulos.

Para as simulações, a princípio, foram estabelecidos dois critérios de parada: 10^{-8} e 10^{-12}; e solução inicial $\mathbf{x}^0 = (1, 1, ..., 1)^T$.

Os testes executados foram distribuídos da seguinte maneira:

- **Estudo 1:** simulações para as matrizes extraídas em [3], respectivamente, para o GC e o GC com CSR (caso I) e para o GCPC e o GCPC com CSR (caso II);
- **Estudo 2:** simulações para as matrizes retiradas do repositório online SuiteSparse Matrix Collection (https://sparse.tamu.edu/), analogamente, para o GCPC sem e com CSR (nessa ordem, casos III e IV).

4.2 Problemas analisados

Os problemas analisados, assim como a distribuição desse estudo numérico, é divido em dois grupos: os sistemas gerados a partir de modelos escolhidos em [3] e os sistemas selecionados do repositório online SuiteSparse Matrix Collection. Na sequência são mostradas as características desses problemas.

4.2.1 Gerador de matrizes

Os primeiros sistemas lineares utilizados foram selecionados a partir do livro do Burden [3]. Eles foram usados como geradores, em que as amostras obtidas variam conforme a dimensão n escolhida previamente. A escolha por esses geradores, se deve a estrutura das matrizes de coeficientes. Essas matrizes são, além de esparsas, tridiagonal e pentadiagonal. É sabido, de estudos anterior realizados pela autora [7] que os métodos tratados apresentaram bom desempenho para esses tipos de matrizes. Porém, naquele momento não utilizou-se a estrtura CSR.

Para a identificação do primeiro problema, esse foi nomeado de *burden*9, cuja matriz de coeficientes esparsa e tridiagonal é dada por

$$a_{ij} = \begin{cases} 2i, & \text{quando} \quad j = i \quad \text{e} \quad i = 1, 2, ..., n, \\ -1, & \text{quando} \begin{cases} j = i + 1 \quad \text{e} \quad i = 1, 2, ..., n - 1, \\ j = i - 1 \quad \text{e} \quad i = 2, 3, ..., n, \end{cases} \\ 0, & \text{caso contrário} \end{cases} \tag{4.1}$$

e o vetor independente é

$$b_i = 1.5i - 6, \quad \text{para} \quad i = 1, 2, ..., n. \tag{4.2}$$

Já o segundo problema tratado, tem como gerador a matriz $\mathbf{A}$ pentadiagonal esparsa na forma

$$a_{ij} = \begin{cases} 2i, & \text{quando} \quad j = 1 \quad \text{e} \quad i = 1, 2, ..., n, \\ 0.5i, & \text{quando} \begin{cases} j = i + 2 \quad \text{e} \quad i = 1, 2, ..., n - 2, \\ j = i - 2 \quad \text{e} \quad i = 3, 4, ..., n, \end{cases} \\ 0.25i, & \text{quando} \begin{cases} j = i + 4 \quad \text{e} \quad i = 1, 2, ..., n - 4, \\ j = i - 4 \quad \text{e} \quad i = 5, 6, ..., n, \end{cases} \\ 0, & \text{caso contrário} \end{cases} \tag{4.3}$$

e o vetor independente dado por

$$b1_i = \pi, \quad \text{para} \quad i = 1, 2, ..., n. \tag{4.4}$$

Além disso, enquanto o **b** do problema 1 se modifica a medida que i varia, o vetor **b1** do segundo sistema é vinculado a uma constante. Diante disso, considerou-se variar seu valor, de modo que vinculou-se também a essse último modelo os seguintes vetores:

$$b2_i = 1{,}0, \quad \text{para} \quad i = 1, 2, ..., n. \tag{4.5}$$

$$b3_i = \phi, \quad \text{para} \quad i = 1, 2, ..., n. \tag{4.6}$$

Assim, para cada sistema gerado pelo Eq. (4.3) e as Eq. (4.4), Eq. (4.5) e Eq. (4.6), deu-se nome de *burden*15*b*1, *burden*15*b*2 e *burden*15*b*3, respectivamente.

Além disso, é importante ressaltar que as matrizes vinculadas a esses problemas são simétricas positivas definidas conforme a exigência dos métodos iterativos adotados.

4.2.2 Matrizes do repositório

O segundo grupo de matrizes foi selecionado a partir do repositório online SuiteSparse Matrix Colletion (https://sparse.tamu.edu/), onde foi feita uma triagem prévia nas matrizes de entradas reais, cuja estrutura fosse simétrica e positiva definida de acordo com os filtros disponíveis. Outra condição de escolha foi a presença do vetor **b** nesses arquivos. Desse modo, foram selecionadas as seguintes matrizes dispostas na Tabela 4.1.

Tabela 4.1: Matrizes do repositório.

Nome	Grupo	SPD	Dimensão	nz	Tipo
$raefsky4$	Simon	sim	19.779	1.328.611	Problema Estrutural
$bodyy4$	Pothen	sim	17.546	121.928	Problema Estrutural
$Press\_Poisson$	ACUSIM	sim	14.822	715.804	Problema Dinâmica de Fluidos
$s1rmq4m1$	Cylshell	sim	5.489	281.111	Problema Estrutural
$s1rmt3m1$	Cylshell	sim	5.489	219.521	Problema Estrutural
$nasa4704$	Nasa	sim	4.704	104.756	Problema Estrutural
$nasa2910$	Nasa	sim	2.910	174.296	Problema Estrutural
$mesh2e1$	Pothen	sim	306	2.018	Problema Estrutural
$mesh3em5$	Pothen	sim	289	1.889	Problema Estrutural
$mesh1em6$	Pothen	sim	48	306	Problema Estrutural

Perceba que essas matrizes se separam de acordo com sua dimensão. No primeiro grupo variam de 10.000 a 20.000, enquanto no próximo grupo variam entre 1.000 e 6.000 e o último envolve dimensões de 40 a 500. Outra curiosidade, é a predominância de problemas estruturais entre os dados selecionados no repositório.

4.3 Ivestigação

4.3.1 Estudo 1

4.3.1.1 Caso I: GC e GC com CSR

Os resultados expostos nesta Seção foram gerados para os problemas apresentados na Subseção 4.2.1, primeiramente, a partir do método dos GC e, em seguida, pelo método dos GC com CSR. As Tabelas 4.2 e 4.3 apresentam esse cenário para a tolerância 10^{-8}, solução inicial $\mathbf{x}^0 = (1, 1, ..., 1)^T$ e número máximo de iterações $N = 10.000$.

Tabela 4.2: GC para tolerância 10^{-8}.

		Dimensão da matriz A		
Problemas		1.000	5.000	10.000
$burden15\_b1$	Iter	340	732	1033
	Tempo	1,774	91,416	452,222
$burden15\_b2$	Iter	343	755	NC
	Tempo	1,807	87,059	NC
$burden15\_b3$	Iter	312	679	NC
	Tempo	1,680	88,258	NC
$burden9$	Iter	161	367	522
	Tempo	1,911	42,099	213,234

Tabela 4.3: GC com CSR para tolerância 10^{-8}.

		Dimensão da matriz A		
Problemas		1.000	5.000	10.000
*burden*15_*b*1	Iter	340	732	1033
	Tempo	1,416	1,275	3,182
*burden*15_*b*2	Iter	340	732	1033
	Tempo	1,518	1,590	2,083
*burden*15_*b*3	Iter	340	732	1033
	Tempo	1,499	1,507	3,711
*burden*9	Iter	161	367	522
	Tempo	1,352	1,036	2,789

Note que na Tabela 4.2 os problemas de dimensão 10.000 não convergiram (NC) para a matriz pentadiagonal, enquanto o problema tridiagonal *burden*9 nas três dimensões 1.000, 5.000 e 10.000 atingiu o número de iterações em quantidade menor do que o limite máximo adotado. Para todos os casos dos GC, as matrizes são "cheias", ou seja, foram preenchidas com seus elementos iguais e diferentes de zero.

Já na Tabela 4.3, para os problemas *burden*15 o número de iterações é o mesmo para cada dimensão adotada. Ao mesmo tempo que os números de iterações para as variações do *burden*9 se mantiveram em relação aos dados da Tabela 4.2.

Por fim, comparando as Tabelas 4.2 e 4.3 percebe-se a influência da CSR na redução do tempo de execução dos GC. Isso se deve ao descarte dos elementos nulos, na construção do esquema de armazenamento.

As Tabelas 4.4 e 4.5 mostram os resultados obtidos para a tolerância 10^{-12}, solução inicial $\mathbf{x}^0 = (1, 1, ..., 1)^T$ e número máximo de iterações $N = 10.000$.

Tabela 4.4: GC para tolerância 10^{-12}.

		Dimensão da matriz A		
Problemas		1.000	5.000	10.000
*burden*15_*b*1	Iter	470	980	1366
	Tempo	2,449	112,442	586,409
*burden*15_*b*2	Iter	477	1019	1426
	Tempo	2,437	118,691	579,042
*burden*15_*b*3	Iter	438	923	1295
	Tempo	2,285	110,958	577,145
*burden*9	Iter	187	426	606
	Tempo	2,034	47,884	250,600

Tabela 4.5: GC com CSR para tolerância 10^{-12}.

		Dimensão da matriz A		
Problemas		1.000	5.000	10.000
*burden*15_*b*1	Iter	470	980	1366
	Tempo	1,585	1,779	4,424
*burden*15_*b*2	Iter	470	980	1366
	Tempo	1,635	1,683	4,265
*burden*15_*b*1	Iter	470	980	1366
	Tempo	1,666	1,730	4,111
*burden*9	Iter	187	426	606
	Tempo	1,428	1,060	2,497

Na Tabela 4.4 o número de iterações aumento, a medida que a dimensão da matriz **A** aumentou para os problemas estudados. O tempo computacional apresentou comportamente análogo ao número de iterações.

Observando, agora, a Tabela 4.3 ainda que as iterações e o tempo sigam a lógica anterior, é notável a redução do último em comparação aos resultados mostrados para o método do GC sem a estrtutura de dados CSR. Isto, reafirma a validade desse esquema de armazenamento, em ignorar os elementos nulos.

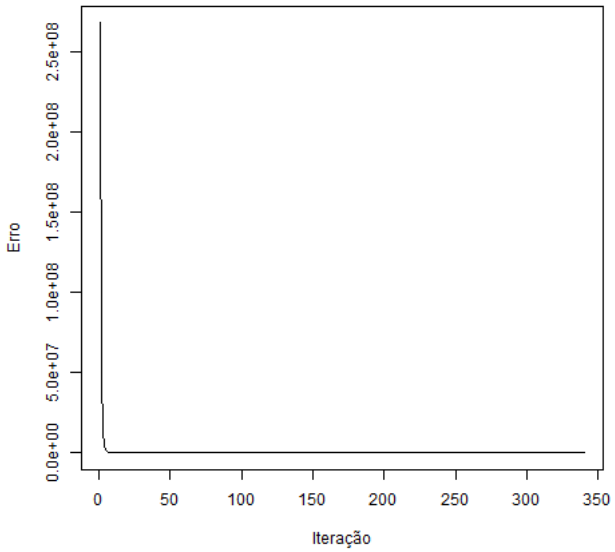
Outro dado relevante para a investigação numérica de matrizes esparsas é a esparsidade. O grau de esparsidade é calculado a partir de

$$\text{GE} = \frac{\text{quantidade de elementos nulos}}{\text{quantidade de elementos não nulos}} \times 100\% \tag{4.7}$$

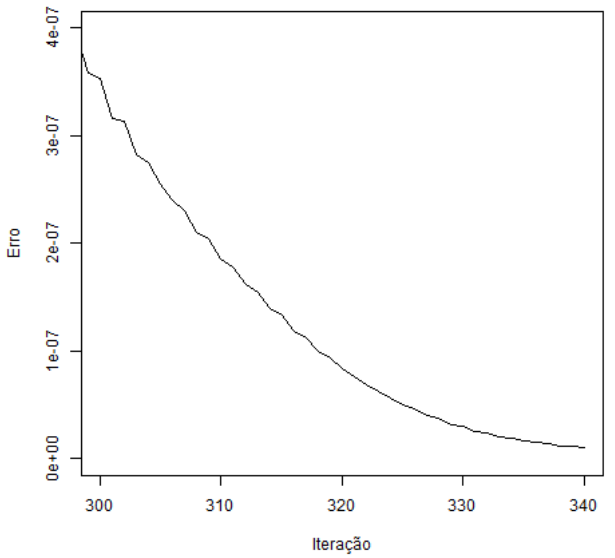
A Tabela 4.6 mostra o GE para as matrizes dessa parte do estudo.

Tabela 4.6: Esparsidade das matrizes dos geradores.

		Dimensão da matriz A		
Problemas		1.000	5.000	10.000
*burden*15	*nz*	4988	24988	49998
	GE (%)	99,501200	99,900048	99,950012
*burden*9	*nz*	2998	14998	29998
	GE (%)	99,700200	99,940008	99,970002

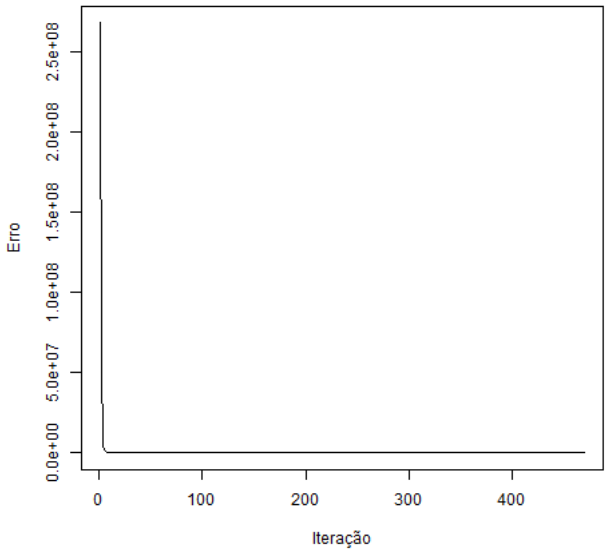


(a) Norma - $burden15\_b1$ para $tol = 10^{-8}$.

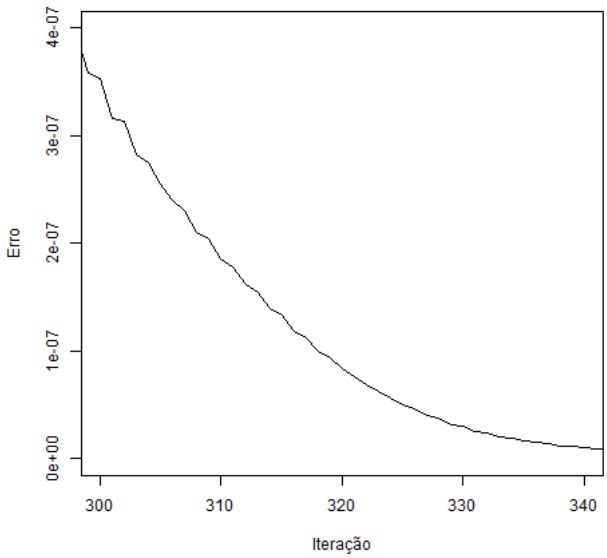


(b) Região de decaimento do erro da concentração da norma - $burden15\_b1$ para $tol = 10^{-8}$.

Figura 4.1: Norma do problema $burden15\_b1$ para $tol = 10^{-8}$. Fonte: A autora, 2022.

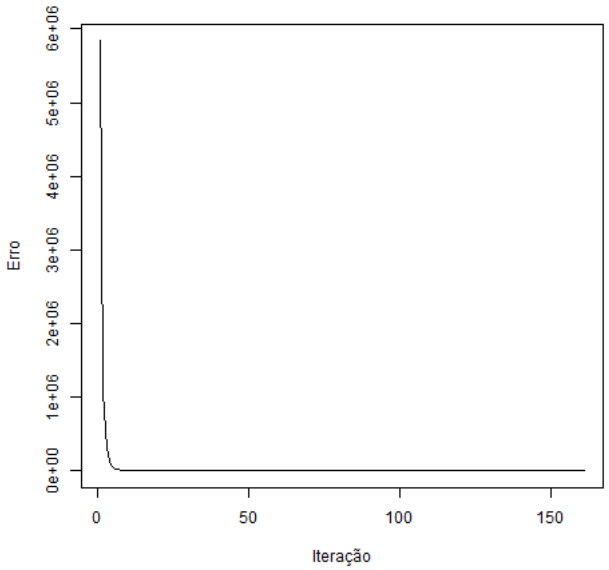


(a) Norma - $burden15\_b1$ para $tol = 10^{-12}$.

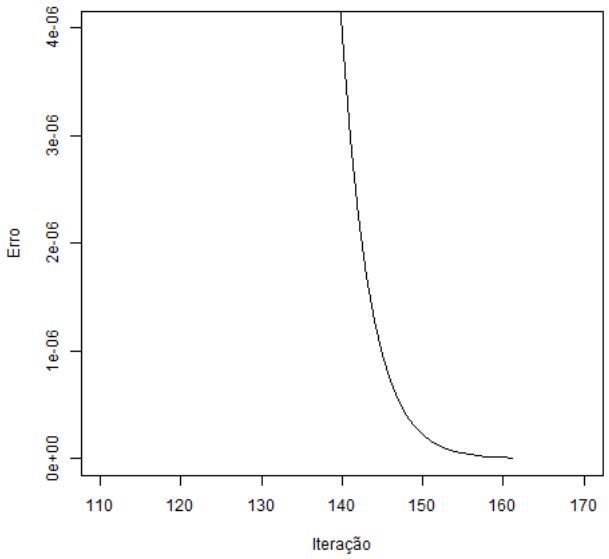


(b) Região de decaimento do erro da concentração da norma - $burden15\_b1$ para $tol = 10^{-12}$.

Figura 4.2: Norma do problema $burden15\_b1$ para $tol = 10^{-12}$. Fonte: A autora, 2022.

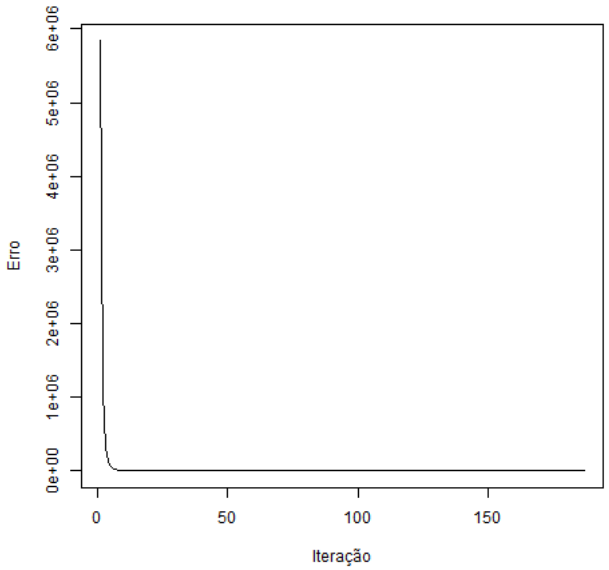


(a) Norma - *burden*9_*b*1 para $tol = 10^{-8}$.

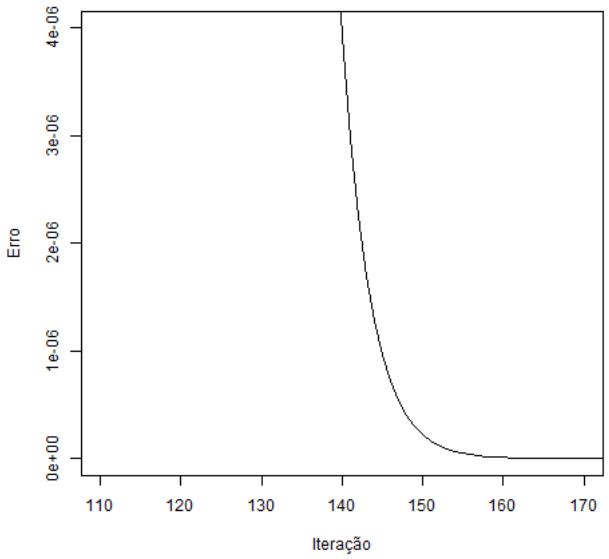


(b) Região de decaimento do erro da concentração da norma - *burden*9_*b*1 para $tol = 10^{-8}$.

Figura 4.3: Norma do problema *burden*9_*b*1 para $tol = 10^{-8}$. Fonte: A autora, 2022.



(a) Norma - *burden*9_*b*1 para $tol = 10^{-12}$.



(b) Região de decaimento do erro da concentração da norma - *burden*9_*b*1 para $tol = 10^{-12}$.

Figura 4.4: Norma do problema *burden*9_*b*1 para $tol = 10^{-12}$. Fonte: A autora, 2022.

4.3.1.2 Caso II: GCPC e GCPC com CSR

Tabela 4.7: GCPC para tolerância 10^{-8}.

		Precondicionadores											
Problemas	n	Ja		DJa		BGS		FGS		BSOR		FSOR	
		Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp
*burden*15_*b*1	1000												
	5.000												
	10.000												
*burden*15_*b*2	1000												
	5.000												
	10.000												
*burden*15_*b*3	1000												
	5.000												
	10.000												
*burden*9	1000												
	5.000												
	10.000												

Tabela 4.8: GCPC com CSR para tolerância 10^{-8}.

		Precondicionadores											
Problemas	n	Ja		DJa		BGS		FGS		BSOR		FSOR	
		Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp
*burden*15_*b*1	1000												
	5.000												
	10.000												
*burden*15_*b*2	1000												
	5.000												
	10.000												
*burden*15_*b*3	1000												
	5.000												
	10.000												
*burden*9	1000												
	5.000												
	10.000												

Tabela 4.9: GCPC para tolerância 10^{-12}.

		Precondicionadores											
Problemas	n	Ja		DJa		BGS		FGS		BSOR		FSOR	
		Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp
*burden*15_*b*1	1000												
	5.000												
	10.000												
*burden*15_*b*2	1000												
	5.000												
	10.000												
*burden*15_*b*3	1000												
	5.000												
	10.000												
*burden*9	1000												
	5.000												
	10.000												

Tabela 4.10: GCPC com CSR para tolerância 10^{-12}.

		Precondicionadores											
Problemas	n	Ja		DJa		BGS		FGS		BSOR		FSOR	
		Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp
*burden*15_*b*1	1000												
	5.000												
	10.000												
*burden*15_*b*2	1000												
	5.000												
	10.000												
*burden*15_*b*3	1000												
	5.000												
	10.000												
*burden*9	1000												
	5.000												
	10.000												

4.3.2 Estudo 2

4.3.2.1 Caso IV: GC com CSR para as matrizes do repositório

Tabela 4.11: GC com CSR para tolerância 10^{-8}.

Problemas	n	GE (%)	Iter	Tempo
$raefsky4$	19.779	99,660383		
$bodyy4$	17.546	99,960392		
$Press\_Poisson$	14.822	99,674178		
$s1rmq4m1$	5.489	99,066979		
$s1rmt3m1$	5.489	99,271399		
$nasa4704$	4.704	99,526583		
$nasa2910$	2.910	99,941734		
$mesh2e1$	306	97,844846		
$mesh3em5$	289	97,738293		
$mesh1em6$	48	86,718750		

Tabela 4.12: GC com CSR para tolerância 10^{-12}.

Problemas	n	GE (%)	Iter	Tempo
$raefsky4$	19.779	99,660383		
$bodyy4$	17.546	99,960392		
$Press\_Poisson$	14.822	99,674178		
$s1rmq4m1$	5.489	99,066979		
$s1rmt3m1$	5.489	99,271399		
$nasa4704$	4.704	99,526583		
$nasa2910$	2.910	99,941734		
$mesh2e1$	306	97,844846		
$mesh3em5$	289	97,738293		
$mesh1em6$	48	86,718750		

4.3.2.2 Caso V: GCPC com CSR para as matrizes do repositório

Tabela 4.13: GCPC com CSR para tolerância 10^{-8}.

Problemas	**Precondicionadores**											
	Ja		DJa		BGS		FGS		BSOR		FSOR	
	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp
$raefsky4$												
$bodyy4$												
$Press\_Poisson$												
$s1rmq4m1$												
$s1rmt3m1$												
$nasa4704$												
$nasa2910$												
$mesh2e1$												
$mesh3em5$												
$mesh1em6$												

Tabela 4.14: GCPC com CSR para tolerância 10^{-12}.

Problemas	**Precondicionadores**											
	Ja		DJa		BGS		FGS		BSOR		FSOR	
	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp
$raefsky4$												
$bodyy4$												
$Press\_Poisson$												
$s1rmq4m1$												
$s1rmt3m1$												
$nasa4704$												
$nasa2910$												
$mesh2e1$												
$mesh3em5$												
$mesh1em6$												

4.3.2.3 Caso VI: Análise do parâmetro ω nos precondicionadores DJa, BSOR e FSOR

Tabela 4.15: GCPC com CSR para tolerância 10^{-8}.

Problemas	ω											
	0,25		0,50		0,75		1,25		1,50		1,75	
	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp
$raefsky4$												
$bodyy4$												
$Press\_Poisson$												
$s1rmq4m1$												
$s1rmt3m1$												
$nasa4704$												
$nasa2910$												
$mesh2e1$												
$mesh3em5$												
$mesh1em6$												

Tabela 4.16: GCPC com CSR para tolerância 10^{-12}.

Problemas	ω											
	0,25		0,50		0,75		1,25		1,50		1,75	
	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp	Iter	Temp
$raefsky4$												
$bodyy4$												
$Press\_Poisson$												
$s1rmq4m1$												
$s1rmt3m1$												
$nasa4704$												
$nasa2910$												
$mesh2e1$												
$mesh3em5$												
$mesh1em6$												

Capítulo 5

Conclusões e Trabalhos Futuros

5.1 Conclusões

Espera-se que o uso da estrtura de dados CSR influencie na rapidez dos métodos GC e GCPC, havendo ganho de tempo computacional. Enquanto os precondicionadores dimunuam o número de iterações à busca da solução aproximada aos problemas analisados.

5.2 Trabalhos Futuros

Pretende-se explorar outras estruturas de dados, como CSC, e outros precondicionadores, como os de fatoração incompleta: ILU e ICC. Além disso, para melhoramento da solução é viável a inclusão do refinamento iterativo no cenário atual deste estudo e nas futuras modificações. Assim, realizar publicações de artigos deste e dos próximos estudos sobre o tema.

Referências

[1] Benzi, M. Preconditioning techniques for large linear systems: a survey. *Journal of computational Physics 182*, 2 (2002), 418–477.

[2] Bertaccini, D., Durastante, F. *Iterative methods and preconditioning for large and sparse linear systems with applications*. CRC Press, 2018.

[3] Burden, R. L., Faires, J. D. *Análise Numérica*, 8 ed. Cengage Learning, 2008.

[4] Cunha, M. C. C. *Métodos numéricos*. Editora da UNICAMP, 2000.

[5] da Costa, V. S. *Técnicas computacionais para resolução de sistemas lineares aplicadas a problemas de enchimento de reservatórios (Dissertação de Mestrado)*. PhD thesis, IPRJ/UERJ, Nova Friburgo, 2008.

[6] da Silva, L. H. *Métodos Iterativos para Sistemas Lineares: Pré-condicionadores, Estruturas de Dados e Outras Técnicas (Dissertação de Mestrado)*. PhD thesis, UFF, Volta Redonda, 2021.

[7] Frango, J. V. d. S. O método dos gradientes conjugados pré-condicionado por fatoração incompleta lu de nível zero.

[8] Frango, J. V. d. S., Pereira, T. J. O método dos gradientes conjugados pré-condicionado pela fatoração ilu (0). *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics 6*, 2 (2018).

[9] Golub, G. H., O'Leary, D. P. Some history of the conjugate gradient and lanczos algorithms: 1948–1976. *SIAM review 31*, 1 (1989), 50–102.

[10] Golub, G. H., Van Loan, C. F. *Matrix Computations*. Baltimore, The Johns Hopkins University Press, 2013.

[11] Hestenes, M. R., Stiefel, E., others. *Methods of conjugate gradients for solving linear systems*, vol. 49. NBS Washington, DC, 1952.

[12] Junqueira, L. A. C. M. *Estudo de suavizadores para o método Multigrid algébrico baseado em wavelet*. PhD thesis, Universidade de São Paulo, 2008.

[13] Meurant, G. *The Lanczos and conjugate gradient algorithms: from theory to finite precision computations*. SIAM, 2006.

[14] Rodrigues, C. F. Análise comparativa entre os métodos decomposição em valores singulares e análise de componentes principais envolvendo matrizes esparsas de grande porte.

[15] Saad, Y. *Iterative methods for sparse linear systems*. SIAM, 2003.

[16] Saad, Y., Van Der Vorst, H. A. Iterative solution of linear systems in the 20th century. *Numerical Analysis: Historical Developments in the 20th Century* (2001), 175–207.

[17] Sauer, T. *Numerical Analysis*. Pearson, 2019.

[18] Shewchuk, J. R., others. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[19] Van der Vorst, H. A. *Iterative Krylov methods for large linear systems*. No. 13. Cambridge University Press, 2003.

[20] Varga, R. S. *Iterative analysis*. Springer, 2000.

[21] Young, D. M. *Iterative solution of large linear systems*. Elsevier, 1970.