UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
ENGENHARIA CIVIL
CÂMPUS APUCARANA

THAYS ROLIM MENDES DE OLIVEIRA

AVALIAÇÃO DE FORMATOS DE ARMAZENAMENTO COM COMPRESSÃO PARA RESOLUÇÃO DE SISTEMAS DE EQUAÇÕES LINEARES ESPARSOS

APUCARANA
2019

THAYS ROLIM MENDES DE OLIVEIRA

AVALIAÇÃO DE FORMATOS DE ARMAZENAMENTO COM COMPRESSÃO PARA RESOLUÇÃO DE SISTEMAS DE EQUAÇÕES LINEARES ESPARSOS

Monografia apresentada como parte dos requisitos necessários para aprovação no componente curricular Trabalho de Conclusão do Curso de Engenharia Civil da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Gylles Ricardo Ströher

APUCARANA
2019

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Apucarana
COECI – Coordenação do Curso Superior de Engenharia Civil

TERMO DE APROVAÇÃO

Avaliação de formatos de armazenamento com compressão para resolução de sistemas de equações lineares esparsos

por

Thays Rolim Mendes de Oliveira

Este Trabalho de Conclusão de Curso foi apresentado aos 19 dias do mês de novembro do ano de 2019, às 10 horas e 00 minutos, como requisito parcial para a obtenção do título de Bacharel em Engenharia Civil, linha de pesquisa métodos numéricos, do Curso Superior em Engenharia Civil da UTFPR – Universidade Tecnológica Federal do Paraná. A candidata foi arguida pela banca examinadora composta pelos professores abaixo assinados. Após deliberação, a banca examinadora considerou o trabalho aprovado.

_____
Prof. Dr. Gylles Ricardo Ströher – ORIENTADOR

_____
Prof. Dr. Luiz Antonio Farani de Souza – EXAMINADOR

_____
Prof. Dr. Ricardo de Almeida Simon – EXAMINADOR

_____
Prof. Dr. Rodolfo Krul Tessari – EXAMINADOR

“A Folha de Aprovação assinada encontra-se na Coordenação do Curso”.

À minha família pelo constante apoio e incentivo ao longo desta caminhada.

AGRADECIMENTOS

Primeiramente a Deus por sua infinita bondade e por ter me concedido saúde e força para chegar aqui.

Gostaria de deixar meu profundo agradecimento ao professor Dr. Gylles Ricardo Ströher pela imensa paciência e incentivo durante os quase cinco anos de orientação.

À Universidade Tecnológica Federal do Paraná e a todos seus colaboradores que de alguma forma contribuíram para a realização desse trabalho.

Ao corpo docente por todo aprendizado e paciência dedicados.

E em especial à minha família, pelo apoio e amor incondicional que me dedicaram, sem vocês nada disso seria possível.

RESUMO

Os esquemas de compressão de matrizes esparsas têm como objetivo reduzir o consumo de memória no armazenamento de matrizes com elevada quantidade de elementos nulos. O presente trabalho aborda o uso dos métodos de compressão para redução do consumo de memória, suprimir operações desnecessárias entre elementos nulos e consequentemente redução do tempo de processamento na resolução de sistemas lineares esparsos. Os esquemas de compressão implementados foram o *Compressed Sparse Row* (CSR), o *Compressed Sparse Columm* (CSC), o *Compressed Sparse Vector* (CSV) e o *Compressed Diagonal Storage* (CDS). Esses esquemas foram implementados para os métodos iterativos de resolução de sistemas lineares Jacobi, Gauss-Seidel e Gradiente Conjugado. Os resultados encontrados apontam para a redução no tempo de processamento ao suprimir as operações com elementos nulos. Os resultados apresentam também a economia de memória fornecida por cada método de compressão. Além disso, realizou-se uma análise comparativa entre a linguagem de programação científica JULIA e o aplicativo MATLAB, possibilitando avaliar o tempo e processamento em cada linguagem.

Palavras-chave: Sistemas Lineares Esparsos. Compressão de Matrizes. Métodos Numéricos.

LISTA DE FIGURAS

Figura 2.1	Matriz exemplo em que são armazenados os elementos nulos e não nulos.	17
Figura 2.2	Matriz tipo banda.	19
Figura 4.1	Resíduo obtido pelo método Jacobi, matriz 1600x1600 e esparsidade de 98%.	45
Figura 4.2	Resíduo obtido pelo método Gauss-Seidel, matriz 1600x1600 e esparsidade de 98%.	45
Figura 4.3	Resíduo obtido pelo método Gradiente Conjugado, matriz 1600x1600 e esparsidade de 98%.	45

LISTA DE QUADROS

Quadro 3.1	Algoritmo CSR	26
Quadro 3.2	Algoritmo CSC	27
Quadro 3.3	Algoritmo CSV	27
Quadro 3.4	Algoritmo CDS	28
Quadro 3.5	Algoritmo de Jacobi para estrutura CSR	29
Quadro 3.6	Algoritmo de Gauss-Seidel para estrutura CSR	30
Quadro 3.7	Algoritmo do Gradiente Conjugado para estrutura CSR	30
Quadro 3.8	Algoritmo de Jacobi para estrutura CSC	32
Quadro 3.9	Algoritmo de Gauss-Seidel para estrutura CSC	33
Quadro 3.10	Algoritmo do Gradiente Conjugado para estrutura CSC	33
Quadro 3.11	Algoritmo de Jacobi para estrutura CSV	34
Quadro 3.12	Algoritmo de Gauss-Seidel para estrutura CSV	35
Quadro 3.13	Algoritmo do Gradiente Conjugado para estrutura CSV	36
Quadro 3.14	Algoritmo de Jacobi para estrutura CDS	38
Quadro 3.15	Algoritmo de Gauss-Seidel para estrutura CDS	38
Quadro 3.16	Algoritmo do Gradiente Conjugado para estrutura CDS	39

LISTA DE TABELAS

Tabela 4.1	Consumo de memória em Bytes para os métodos CSR e CSC	42
Tabela 4.2	Consumo de memória em Bytes para os métodos CSV e CDS	42
Tabela 4.3	Tempo de compressão (s)	43
Tabela 4.4	Tempo de processamento para resolução do sistema de equações lineares para o método de Jacobi (s)	46
Tabela 4.5	Tempo de processamento em segundos para resolução do sistema de equações lineares para o método de Gauss-Seidel	47
Tabela 4.6	Tempo de processamento para resolução do sistema de equações lineares para o método Gradiente Conjugado (s)	48
Tabela 4.7	Redução do tempo de processamento para o método CSR	48
Tabela 4.8	Redução do tempo de processamento para o método CSC	49
Tabela 4.9	Redução do tempo de processamento para o método CSV	49
Tabela 4.10	Redução do tempo de processamento para o método CDS	50
Tabela 4.11	Tempo de compressão em segundos para a linguagem JULIA	51
Tabela 4.12	Redução do tempo de processamento da compressão para a linguagem JULIA	51
Tabela 4.13	Tempo de processamento para resolução do sistema de equações lineares para o método Gradiente Conjugado (s)	52
Tabela 4.14	Redução do tempo de processamento para a linguagem JULIA (s)	52

GLOSSÁRIO

CDS: *Compressed Diagonal Storage*

CSC: *Compressed Sparse Colunm*

CSR: *Compressed Sparse Row*

CSV: *Compressed Sparse Vector*

GC: Gradiente Conjugado

IE: Índice de Esparsidade

SUMÁRIO

LISTA DE FIGURAS 6
LISTA DE QUADROS 7
LISTA DE TABELAS 8
GLOSSÁRIO 9
1 INTRODUÇÃO 12
1.1 Justificativa 13
1.2 Objetivos 14
1.2.1 Objetivo Geral 14
1.2.2 Objetivo específicos 14
2 FUNDAMENTAÇÃO TEÓRICA 16
2.1 Esparsidade 16
2.2 Métodos de compressão 16
2.2.1 Compressed Sparse Row (CSR) 17
2.2.2 Compressed Sparse Column (CSC) 17
2.2.3 Compressed Sparse Vector (CSV) 18
2.2.4 Compressed Diagonal Storage (CDS) 19
2.3 Métodos iterativos para solução de sistemas lineares 20
2.3.1 Jacobi 20
2.3.2 Gauss-Seidel 21
2.3.3 Gradiente Conjugado 21
2.3.4 Matriz diagonalmente dominante 22
2.4 Critérios de parada 22
2.5 Geração de sistemas lineares 24
2.6 Problemas aplicáveis à engenharia civil 24
3 MATERIAIS E MÉTODOS 26
3.1 Algoritmos dos esquemas de compressão 26
3.2 Algoritmos dos métodos iterativos 29
3.3 Geração dos sistemas lineares 40
3.4 Linguagem de programação 41
4 RESULTADOS E DISCUSSÃO 42
4.1 Eficiência dos métodos de compressão quanto ao uso de memória 42
4.2 Avaliação dos métodos de compressão quanto ao tempo de processamento 43
4.3 Eficiência dos métodos de resolução quanto à convergência do resultado 44
4.4 Avaliação dos métodos de resolução quanto ao tempo de processamento 46
4.5 Comparação entre a linguagem JULIA e o aplicativo MATLAB 50

5 CONCLUSÃO 54
REFERÊNCIAS 55

1 INTRODUÇÃO

Uma matriz é dita esparsa quando contém uma grande quantidade de elementos nulos (Coelho, 2014). Matrizes esparsas recorrentemente surgem de forma direta no processo de solução de muitos problemas práticos de engenharia, como, por exemplo, em modelo de balanço de massa e de energia, sistemas de distribuição de eletricidade, água e gases. Matrizes esparsas também surgem de forma indireta, como nos casos em que equações diferenciais ordinárias ou parciais são resolvidas numericamente aplicando métodos como de diferenças finitas, elementos finitos ou volumes finitos. Após a discretização das equações diferenciais, tem origem um sistema linear do tipo $\boldsymbol{Ax} = \boldsymbol{b}$, que, dependendo das dimensões e características do problema, pode resultar em uma matriz $\boldsymbol{A}$ esparsa e de grande dimensão.

A fim de exemplificação, considere o caso da resolução da equação de Laplace bidimensional ($\partial^2 \frac{\theta}{\partial x^2} + \partial^2 \frac{\theta}{\partial y^2} = 0$) em coordenadas cartesianas com condições de contorno Dirichlet. Este caso pode representar vários problemas práticos de engenharia, como a transferência de calor em um meio sólido ou a transferência de massa de um soluto em meio estagnado. Utilizando-se a técnica de volumes finitos (VERSTEEG e MALALASEKERA, 2007) com malhas estruturadas de 10x10, 20x20 e 40x40 volumes, os índices de esparsidade , porcentagem de elementos nulos, das matrizes obtidas são de 95%, 98,75% e 99,68%, respectivamente. Isto é, apenas 5%, 1,25% e 0,312% são elementos não nulos. Em particular para a malha mais refinada, a matriz possui 256.104 elementos, sendo apenas 8.000 diferentes de zero. Em Duff (1977) é apresentada uma longa lista de aplicações práticas, com referências, em que sistemas lineares esparsos surgem no processo de solução de problemas. Ainda como afirma Rizwan (2007), estima-se que em 75% dos problemas científicos a solução de um sistema linear de equações aparece em algum estágio da solução.

Historicamente, o interesse em matrizes esparsas se intensificou na década de 50, quando pesquisadores de sistemas de potência elétrica começaram a resolver problemas reais usando computadores. Foi verificado que, quando problemas reais eram modelados por meio de sistemas de equações lineares, as matrizes resultantes eram esparsas com uma estrutura de elementos não nulos altamente padronizada. O tema matrizes esparsas tornou-se então cada vez mais importante. A primeira conferência sobre esse tema foi realizada pela IBM *Research Center* em 1968. Desde então, surgiram diversas técnicas para o armazenamento de matrizes associadas à resolução de sistemas de equações lineares (CONNOR, 1985).

Segundo Saad (2003), ainda na década de 50 e 60 os métodos diretos para resolução de sistemas lineares eram preferidos em aplicações reais por causa de sua robustez. No entanto, foram desenvolvidos uma série de métodos iterativos e a necessidade de se resolver grandes sistemas lineares provocou uma visível e rápida mudança na direção destes métodos em muitas aplicações. Essa tendência pôde ser observada em meados das décadas de 1960 e 1970, quando dois acontecimentos revolucionaram os métodos de solução para grandes sistemas lineares. O primeiro foi a intenção de se tirar proveito da esparsidade dos sistemas para projetar métodos diretos de resolução de sistemas lineares esparsos, que pode ser bem mais eficiente que os métodos diretos convencionais. O segundo foi o surgimento do método do Gradiente Conjugado.

Atualmente, dentre os métodos iterativos mais empregados nos sistemas de engenharia destacam-se o método de Jacobi, Gauss-Seidel, *Successive Over-Relaxation* (SOR) e Gradiente Conjugado, além de suas derivações. A fim de tirar proveito da esparsidade de sistemas lineares e também com o intuito de economizar memória computacional, surgiram diversos formatos de armazenamento, também denominados esquemas de compressão de matrizes. De acordo com Saad (1994), há mais de treze formatos de armazenamento diferentes para matrizes. Entre estes pode-se destacar o *Compressed Sparse Row* (CSR), *Compressed Sparse Colunm* (CSC), *Compressed Diagonal Storage* (CDS), *Skyline* (SAAD, 2003; SOLIN, 2006) e *Compressed Sparse Vector* (CSV) (FARZANEH et al., 2009).

De acordo com o tipo da matriz, alguns formatos podem ser mais adequados do que outros. Por exemplo, o CDS, segundo Coelho (2014), é mais indicado para matrizes do tipo banda, em que somente a diagonal principal e um pequeno número de diagonais acima e abaixo da diagonal principal são não nulas. Já o formato *Skyline*, de acordo com Bathe e Wilson (1976), é interessante para matrizes simétricas, decorrente da análise de elementos finitos. De um modo geral os esquemas de compressão visam aproveitar o grande número de elementos nulos da matriz e armazenar somente os elementos não nulos.

1.1 Justificativa

A utilização de métodos de compressão em conjunto com métodos de solução de sistemas lineares de equações provê uma diminuição significativa no número de entradas do algoritmo, além de suprimir operações desnecessárias entre elementos nulos. Ao suprimir estas operações, obtém-se tanto ganho computacional quanto economia de memória de armazenamento e de tempo de processamento. Contudo, existem pouquíssimos trabalhos

disponíveis relatando um estudo sistemático do uso de formatos de compressão de matrizes em função do tamanho do sistema linear, índice de esparsidade e métodos iterativos. Não há referências indicando para quais valores de esparsidade e tamanhos de matrizes os esquemas de compressão tornam-se vantajosos. Uma vez que a compressão demanda tempo computacional para ser executada, também é importante ter indicativos de qual seria a redução no tempo de processamento de solução de sistemas lineares por métodos iterativos que operem no formato de armazenamento comprimido.

Além disso, o uso reduzido de memória computacional é particularmente importante ao lidar com conjuntos de dados que são maiores que a memória RAM (*Random Access Memory* - Memória de Acesso Aleatório) disponível. Os sistemas operacionais complementam a memória RAM física com armazenamento em disco rígido. Acessar dados armazenados no disco rígido é, em geral, uma ordem de magnitude mais lenta que acessar dados armazenados na memória RAM. Assim, é evidente que para obter um bom desempenho, o uso da memória deve ser baixo o suficiente para impedir que o sistema operacional fique sem memória RAM suficiente quando um código estiver em execução.

1.2 Objetivos

1.2.1 Objetivo Geral

O presente trabalho tem o intuito de apresentar um estudo de formatos de compressão aplicados aos métodos de resolução de sistemas lineares Jacobi, Gauss- Seidel e Gradiente Conjugado. Pretende-se comparar o tempo de processamento entre métodos iterativos sem e com a aplicação dos métodos de compressão. Pretende-se testar os métodos para diversas esparsidades e ordens de matrizes, a fim de verificar quando os métodos se tornam vantajosos. Adicionalmente, objetiva-se avaliar a redução de memória obtida por meio do armazenamento comprimido das matrizes.

1.2.2 Objetivo específicos

São objetivos específicos desse trabalho:

1. Implementar os métodos de compressão CSR, CSV e CDS para os métodos de resolução de sistemas lineares Jacobi, Gauss-Seidel e Gradiente Conjugado;

2. Comparar o desempenho dos métodos de resolução com e sem o uso dos métodos de compressão, avaliando a eficiência quanto ao uso de memória e tempo de processamento;

3. Avaliar entre os métodos citados qual apresenta maior vantagem de utilização;

4. Analisar quando é viável aplicar a compressão, levando em conta o tempo demandado para comprimir a matriz; e

5. Avaliar o tempo de processamento dos algoritmos implementados no aplicativo Matlab e na liguaguem científica Julia.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Esparsidade

Segundo Coelho (2014) uma matriz é dita esparsa quando contém uma grande quantidade de elementos nulos, possibilitando explorar essa característica para obter economia de memória e de tempo de processamento. Em linguagem matemática, a matriz $A \epsilon \mathbb{R}^{(m,n)}$ é dita como esparsa se $r_{máx} \ll m$, onde m é o número de colunas e $r_{máx}$ é o número de elementos não nulos da linha.

A esparsidade de uma matriz pode ser quantificada pela porcentagem de elementos nulos presentes nela. Essa porcentagem é chamada Índice de Esparsidade (IE), que é expresso pela equação (MALISKA, 2004):

$$IE = \frac{número\ de\ elementos\ nulos}{número\ total\ de\ elementos} 100\% \qquad (1)$$

Em matrizes quadradas, como é o caso de sistemas de equações, o número total de elementos é dado por n^2, onde n é a ordem da matriz. Desta forma, pode-se escrever o IE como sendo:

$$IE = \frac{número\ de\ elementos\ nulos}{n^2} 100\% \qquad (2)$$

Os modelos de compressão surgiram com objetivo de explorar a esparsidade das matrizes, de forma a obter redução na memória de armazenamento. Mendes et al. (2017) afirmam que é possível obter uma redução de até 96% na memória de armazenamento, para matrizes de alta esparsidade.

2.2 Métodos de compressão

Neste tópico serão apresentados algumas das estruturas existentes para a compressão, bem como as particularidades e indicações de cada método.

2.2.1 Compressed Sparse Row (CSR)

Originalmente sugerido por Rose e Willoughby (1972) e posteriormente por Brameller et al. (1976), o CSR é o mais popular esquema de armazenamento para matrizes esparsas. O método realiza o armazenamento em três vetores, $\boldsymbol{aa}$, $\boldsymbol{jr}$ e $\boldsymbol{jc}$, com respectivos comprimentos N_z,N_z e $n+1$, onde n é o número de linhas da matriz e N_z é o número de elementos não nulos da matriz. O vetor $\boldsymbol{aa} \epsilon \mathbb{R}^{N_z}$ contém os valores dos elementos não nulos da matriz $\boldsymbol{A}$, armazenados seguindo a sequência em que se encontram nas linhas; $\boldsymbol{jr} \in \mathbb{N}^{N_z}$ contém os índices da coluna à qual corresponde o elemento não nulo; e $\boldsymbol{jc} \in \mathbb{N}^{n+1}$ contém os índices da posição que o primeiro elemento não nulo de cada linha ocupa no vetor $\boldsymbol{aa}$, percorrendo da linha 1 a $n+1$. Sendo assim, o elemento $\boldsymbol{jc}$(n+1) contém o valor $N_z + 1$, que sugere o término das linhas da matriz. Para exemplificação, considere a matriz quadrada 5 x 5 mostrada na Fig. 2.1.

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

Figura 2.1: Matriz exemplo em que são armazenados os elementos nulos e não nulos.

O formato CSR armazenará a matriz mostrada na Fig. 2.1 da seguinte forma:

$$\boldsymbol{aa} = [\,1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11\,]$$
$$\boldsymbol{jr} = [\,1 \quad 3 \quad 2 \quad 4 \quad 1 \quad 3 \quad 5 \quad 3 \quad 4 \quad 1 \quad 5\,]$$
$$\boldsymbol{jc} = [\,1 \quad 3 \quad 5 \quad 8 \quad 10 \quad 12\,]$$

2.2.2 Compressed Sparse Column (CSC)

Analogamente ao formato CSR, o CSC armazena a matriz em três vetores. Os dois formatos se diferem apenas pela ordem que os elementos são armazenados nos vetores. Enquanto foi visto que no CSR os elementos são armazenados seguindo a sequência em que se encontram nas linhas, no CSC serão armazenados na ordem em que se encontram nas colunas. O esquema CSC é simplesmente o CSR para a matriz $\boldsymbol{A}^T$ (SAAD, 1994).

Seguindo esta lógica, a matriz apresentada na Fig. 2.1 será armazenada da seguinte forma:

$$\boldsymbol{aa} = [\,1 \quad 5 \quad 10 \quad 3 \quad 2 \quad 6 \quad 8 \quad 4 \quad 9 \quad 7 \quad 11\,]$$
$$\boldsymbol{ja} = [\,1 \quad 3 \quad 5 \quad 2 \quad 1 \quad 3 \quad 4 \quad 2 \quad 4 \quad 3 \quad 5\,]$$
$$\boldsymbol{jc} = [\,1 \quad 4 \quad 5 \quad 8 \quad 10 \quad 12]$$

2.2.3 Compressed Sparse Vector (CSV)

O método CSV foi proposto por Farzaneh et al. (2009). Nesse, os elementos não nulos das matrizes são armazenados em dois vetores, $\boldsymbol{aa}$ e $\boldsymbol{ia}$. O vetor $\boldsymbol{aa}$ deve armazenar os valores dos elementos não nulos de cada matriz e o vetor $\boldsymbol{ia}$ sua posição na matriz. Na ideia inicial desse método, cada posição da matriz receberia um índice, sendo que uma matriz quadrada de ordem 5 teria 25 índices. Seguindo essa lógica, o vetor $\boldsymbol{ia}$ salvaria o índice correspondente ao elemento não nulo salvo em $\boldsymbol{aa}$, devendo os elementos serem salvos na ordem em que se encontrarem nas linhas da matriz. Para simplificar o entendimento, a matriz apresentada na Figura 2.1 seria salva da seguinte forma:

$$\boldsymbol{aa} = [\,1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11\,]$$
$$\boldsymbol{ia} = [\,1 \quad 3 \quad 7 \quad 9 \quad 11 \quad 13 \quad 15 \quad 18 \quad 19 \quad 21 \quad 25\,]$$

Contudo, Farzaneh et al. (2009) notaram que conforme as dimensões da matriz $\boldsymbol{A}$ aumentam, se torna inviável esse armazenamento por gerar valores numéricos muito grandes no vetor $\boldsymbol{ia}$. Para solucionar esse problema, sugeriram uma alteração no método, no qual a contagem da posição do elemento não nulo deve ser feita com relação ao elemento não nulo anterior e a contagem começaria partindo do primeiro elemento da matriz, $a_{1,1}$. Dessa forma, caso o elemento $a_{1,1}$ seja nulo, a contagem parte dessa posição: $a_{1,1}$ receberá o índice 1, $a_{1,2}$ índice 2, e assim sucessivamente, até o primeiro elemento não nulo, onde a contagem é reinicializada. Além disso, acrescentou-se o elemento $N_z + 1$ aos dois vetores. No elemento $\boldsymbol{aa}(N_z + 1)$ será armazenado o número de linhas da matriz e no $\boldsymbol{ia}(N_z + 1)$ o número de colunas da matriz.

Seguindo essa metodologia a matriz apresentada na Figura 2.1 deve ser armazenada da seguinte forma:

$$\boldsymbol{aa} = [\,1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11 \quad 5\,]$$
$$\boldsymbol{ia} = [\,1 \quad 2 \quad 4 \quad 2 \quad 2 \quad 2 \quad 2 \quad 3 \quad 1 \quad 2 \quad 4 \quad 5\,]$$

Segundo Farzaneh et al. (2009), o CSV apresenta-se mais viável em termos de redução de memória do que o CSR.

2.2.4 Compressed Diagonal Storage (CDS)

A estrutura CDS é recomendada para matrizes do tipo diagonalmente definidas, também chamadas de matrizes tipo banda. Uma matriz é considerada banda se existirem duas constantes positivas p e q, chamadas *half-band width*, à esquerda e à direita, respectivamente, de forma que $a(i,j) \neq 0$ se, e somente se, $i - p < j < i + q$. Neste caso, a matriz pode ser escrita na forma de vetores que salvem as diagonais do seguinte modo $VAL(1{:}\,n, -p{:}\,q)$ (WORD COMMUNITY GRID,1995). De forma mais simplificada, pode-se dizer que p e q são a quantidade de diagonais não nulas à esquerda e à direita da diagonal principal, respectivamente. Para simplificar o entendimento, segue exemplo do armazenamento para a matriz da Figura 2.2.

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 \\ 3 & 4 & 5 & 0 & 0 \\ 0 & 6 & 7 & 8 & 0 \\ 0 & 0 & 9 & 10 & 11 \\ 0 & 0 & 0 & 12 & 13 \end{bmatrix}$$

Figura 2.2: Matriz tipo banda.

Sendo a ordem da matriz $n = 5$, então $VAL(1{:}\,5, -1{:}\,1)$, pois tem-se uma diagonal acima e uma abaixo da principal. Logo, o armazenamento pelo CDS será:

$$VAL(:,-1) = [0 \quad 3 \quad 6 \quad 9 \quad 12]$$
$$VAL(:,0) = [1 \quad 4 \quad 7 \quad 10 \quad 13]$$
$$VAL(:,+1) = [2 \quad 5 \quad 8 \quad 11 \quad 0]$$

Vale chamar atenção para os elementos nulos em $VAL(:,-1)$ e $VAL(:,+1)$. Eles surgem porque nesse esquema cada posição do vetor determina a linha em que o elemento está posicionado na matriz. Sendo assim, a diagonal dada pelo vetor $VAL(:,-1)$ não apresenta nenhum elemento na primeira linha da matriz e a diagonal $VAL(:,+1)$ não apresenta nenhum elemento correspondente à última linha.

Um ponto importante que deve ser ressaltado sobre esse método é que caso ocorra alguma diagonal nula no intervalo $-p$ a q ela será armazenada também.

Almeida (2003) destaca a importância de se utilizar métodos de resolução de sistemas lineares adaptados para matrizes esparsas, usando os esquemas de compressão para otimizar a resolução dos métodos iterativos.

2.3 Métodos iterativos para solução de sistemas lineares

Os métodos iterativos se caracterizam pela busca da solução de um sistema linear em etapas. Na medida em que o número de etapas tende a infinito, a aproximação gerada pela respectiva etapa tende a solução exata do problema (FREITAS, 2000). Segundo Cunha (2000), os métodos iterativos são indicados para sistemas lineares grandes e esparsos, como os que aparecem frequentemente da discretização de equações diferenciais. A seguir serão apresentados os principais métodos iterativos para a resolução de sistemas lineares.

2.3.1 Jacobi

O método que recebe o nome de seu autor, Carl Gustav Jacobi, foi introduzido em 1845. Neste método, usando uma estimativa inicial, $\boldsymbol{x}^{(1)}$, calcula-se os valores de uma sequência de aproximações $\boldsymbol{x}^{(2)}$, $\boldsymbol{x}^{(3)}$, ... , $\boldsymbol{x}^{(n)}$, onde $\boldsymbol{x}^{(n)}$ consiste na solução do sistema (CUNHA, 2000).

O método de Jacobi fornece os valores de $\boldsymbol{x}$ da respectiva iteração através da equação disponível em Freitas (2000):

$$\boldsymbol{x}_i^{(k+1)} = \frac{1}{a_{ii}} \left(\boldsymbol{b}_i - \sum_{\substack{j \neq i \\ j=1}} a_{ij} \cdot \boldsymbol{x}_j^{(k)} \right), \qquad i = 1,2,3, \dots, n. \tag{3}$$

Onde é necessária uma estimativa inicial para os valores de $\boldsymbol{x}$. Essa estimativa inicial é dada em $k = 1$, sendo k a iteração corrente. As iterações seguintes utilizam os valores da iteração anterior, como expresso na Equação (3).

É importante ressaltar que, para aplicação desse método, os elementos pertencentes à diagonal principal da matriz devem obrigatoriamente ser não nulos. Ou seja, $a_{ii} \neq 0$ para todo i.

2.3.2 Gauss-Seidel

O método de Gauss-Seidel é semelhante ao de Jacobi, porém, segundo Freitas (2000) o de Jacobi utiliza sempre os valores da iteração de ordem (k), enquanto o de Gauss-Seidel utiliza os valores já calculados da iteração de ordem $(k-1)$, ou seja, quando $i > j$. Cunha (2000) afirma que Seidel foi aluno de Jacobi e publicou o método em 1874 como uma melhoria para o método de Jacobi. Como o Gauss- Seidel apresenta uma atualização imediata dos dados, se espera que ele apresente uma convergência mais rápida, no entanto isso não pode ser generalizado.

A Equação (4) apresenta o método de Gauss-Seidel, segundo Freitas (2000):

$$\boldsymbol{x}_i^{(k+1)} = \frac{1}{a_{ii}}\left(\boldsymbol{b}_i - \sum_{j<i} a_{ij} \cdot \boldsymbol{x}_j^{(k+1)} - \sum_{j>i} a_{ij} \cdot \boldsymbol{x}_j^{(k)}\right), \qquad i = 1,2,3, \dots, n. \tag{4}$$

Assim como no método de Jacobi, o método de Gauss-Seidel precisa de uma estimativa inicial na iteração de ordem $(k = 1)$ e os elementos da diagonal principal da matriz devem ser não nulos.

2.3.3 Gradiente Conjugado

Inicialmente estudado na década de 50, muitos autores atribuem à Hestenes e Stiefel (1952) a base do método gradiente conjugado (CUNHA,2000). A motivação para tal estudo foi a solução de sistemas provenientes da discretização de equações diferenciais parciais e viabilização pela entrada dos computadores nos cálculos científicos.

O método gradiente conjugado, segundo o disposto em Cunha (2000), é dado por:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - s^{(k)} \cdot \boldsymbol{v}^{(k+1)} \tag{5}$$

em que:

$$\boldsymbol{r}^{(k)} = -grad\, F\left(x^{(k)}\right) = \boldsymbol{b} - \boldsymbol{A} \cdot \boldsymbol{x}^{(k)} \tag{6}$$

$$s^{(k+1)} = \frac{\boldsymbol{r}^{(k)^T} \cdot \boldsymbol{v}^{(k+1)}}{\boldsymbol{v}^{(k+1)^T} \cdot \boldsymbol{A} \cdot \boldsymbol{v}^{(k+1)}} \tag{7}$$

$$\boldsymbol{v}^{(k+1)} = \boldsymbol{r}^{(k+1)} + \left(\frac{\boldsymbol{r}^{(k+1)^T} \cdot \boldsymbol{r}^{(k+1)}}{\boldsymbol{r}^{(k)^T} \cdot \boldsymbol{r}^{(k)}}\right) \boldsymbol{v}^{(k)} \tag{8}$$

O método Gradiente Conjugado possui uma particularidade adicional para a matriz dos coeficientes: a mesma necessariamente deve ser simétrica. Desta forma, para o funcionamento do método, a matriz dos coeficientes deve ser multiplicada pela sua transposta.

2.3.4 Matriz diagonalmente dominante

Freitas (2000) afirma que ao se trabalhar com matrizes diagonalmente dominantes a sequência de iterações converge para a solução nos métodos Jacobi e Gauss-Seidel. Uma matriz é dita diagonalmente dominante quando o elemento da diagonal principal é em módulo maior que a soma dos demais elementos da linha. Ou seja, a matriz $\boldsymbol{A}(n,n)$ será dita diagonalmente dominante se:

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \qquad i = 1, \ldots, n \tag{9}$$

Neste trabalho serão utilizadas matrizes diagonalmente dominantes.

2.4 Critérios de parada

O critério de parada para as iterações é o que impede que o computador opere iterações por um número infinito de vezes. Dentre as possibilidades de aplicação do critério de parada, uma delas seria determinar um número máximo de iterações para o problema, não sendo uma solução muito eficaz por não garantir a convergência do resultado. Seguindo para uma abordagem mais eficiente, tem-se a utilização do cálculo do erro ou ainda a utilização do resíduo.

A bibliografia apresenta dois possíveis critérios de parada, o absoluto e o relativo. O erro absoluto consiste na máxima diferença entre o valor obtido pela iteração atual e o valor obtido pela iteração anterior, em módulo. Contudo essa abordagem pode não ser eficiente ao se trabalhar com problemas nos quais a solução seja muito pequena ou próxima ao erro estipulado, uma vez que o critério de parada pode ser atingido sem que o sistema tenha convergido para a solução. Com intuito de eliminar esse problema surgiu o erro relativo, que é o erro absoluto dividido pelo módulo do valor obtido na iteração atual. Ao levar em consideração os valores do vetor solução, o erro relativo elimina falhas decorrentes de soluções próximas ao erro estipulado (RUGGIERO e LOPES, 1996; VALLE, 2019; WANDRESEN, 1980).

As Equações (10) e (11) mostram os critérios de parada absolutos e relativos, respectivamente:

$$E_a = \left\| \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)} \right\|_\infty \tag{10}$$

$$E_r = \frac{\left\| \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)} \right\|_\infty}{\left\| \boldsymbol{x}^{(k+1)} \right\|_\infty} \tag{11}$$

Embora o erro relativo elimine problemas decorrentes de soluções muito pequenas, ele pode não ser suficiente para se chegar à solução do problema. Isso ocorre porque o valor do vetor solução $\boldsymbol{x}$ é desconhecido, de modo que nos casos em que as iterações apresentem pequenas variações, o erro pode indicar a convergência de forma equivocada. A alternativa encontrada para esse problema seria calcular o resíduo gerado pela solução do problema. O resíduo é dado pela diferença, em módulo, entre o vetor $\boldsymbol{b}$ e a multiplicação $\boldsymbol{A}\boldsymbol{x}^{(k+1)}$, (VALLE, 2019; WANDRESEN, 1980)

Assim como para o erro, pode-se obter o resíduo absoluto e o relativo, sendo o relativo o mais eficiente. As Equações (12) e (13) apresentam o resíduo relativo e absoluto, respectivamente:

$$R_a = \left\| \boldsymbol{b} - \boldsymbol{A} \cdot \boldsymbol{x}^{(k+1)} \right\| \tag{12}$$

$$R_r = \frac{\left\| \boldsymbol{b} - \boldsymbol{A} \cdot \boldsymbol{x}^{(k+1)} \right\|}{\|\boldsymbol{b}\|} \tag{13}$$

2.5 Geração de sistemas lineares

Para a geração de sistemas lineares do tipo $\boldsymbol{Ax} = \boldsymbol{b}$ com $\boldsymbol{A} \in \mathbb{R}^{(\mathrm{n}\times\mathrm{n})}$ e $\mathbf{b} \in \mathbb{R}^{(\mathrm{n})}$, pretende-se elaborar um algoritmo a partir de alguns ajustes no sistema linear obtido da discretização por diferenças finitas da seguinte equação diferencial parcial:

$$\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} + a \cdot \varphi + b(x, y) = 0 \tag{14}$$

com condições de contorno:

$$\frac{\partial \varphi}{\partial \eta} = c \tag{15}$$

onde η representa a direção normal a cada aresta do domínio computacional e a, b e c são constantes.

A escolha da equação diferencial se deve ao fato de ser uma equação genérica para a modelagem de diversos problemas práticos bidimensionais.

2.6 Problemas aplicáveis à engenharia civil

Stern (1994) afirma que diversos problemas da engenharia geram sistemas lineares cujas matrizes são esparsas. Entre eles o autor cita a resolução de equações diferenciais, análise de sistemas, circuitos ou estruturas e programação linear. O mesmo ainda destaca que os usuários tendem a resolver problemas de grandes dimensões, e que o IE da matriz cresce

juntamente com a dimensão do problema. Como visto anteriormente, quanto maior o IE, mais vantajoso se torna o uso dos métodos de compressão. Com o intuito de justificar a importância desse trabalho para a engenhara civil, encontram-se exemplificados quatro problemas, de diferentes áreas da engenharia civil, que recaem em sistemas lineares.

Valiente (2015) demonstra que problemas de resolução de treliças e determinação de vibração em estruturas recaem em sistemas lineares esparsos. O estudo aborda métodos diretos e iterativos para a resolução dos sistemas, onde o autor afirma que os métodos iterativos são melhores para problemas reais, porque esses apresentam grande quantidade de incógnitas.

Almeida (2003) realiza uma análise da interação solo/subestrutura/superestrutura considerando a subestrutura imersa em solo não homogêneo. Utiliza o método dos elementos de contorno em abordagem 3D juntamente com o método de elementos finitos para a modelagem da superestrutura, recaindo em sistemas lineares esparsos. O autor ressalta a importância da utilização de métodos de resolução de sistemas lineares esparsos a fim de tomar proveito dessa característica das matrizes na resolução.

Os autores Mattos et al. (2018) apresentam uma simulação da infiltração de águas em solos não saturados, em comparação com os estudos de Guterres et al. (2013). Para isso, empregam o método de diferenças finitas para a discretização da equação de Richards bidimensional, recaindo em um sistema de equações lineares. O sistema é resolvido pelo método de Gauss-Seidel.

Colaço et al. (2014) apresentam um estudo numérico sobre a evolução do estado de tensão geomecânica induzido pelo tráfego ferroviário, considerando uma interação entre o sistema composto pelo comboio, via-férrea e maciço de fundação. Assim como Almeida (2003), é empregado o método dos elementos de contorno juntamente com o método dos elementos finitos, recaindo em sistemas lineares esparsos.

De um modo geral, como afirma Cunha (2000), a resolução de forma numérica de equações diferenciais recai em sistemas lineares. Sabendo que grande parte dos problemas da engenharia civil podem ser modelados por equações diferenciais, tem-se uma vasta lista de problemas aos quais esse trabalho se aplica.

3 MATERIAIS E MÉTODOS

3.1 Algoritmos dos esquemas de compressão

Para implementação dos esquemas de compressão para resolução de sistemas lineares de alta ordem, primeiramente foram criados algoritmos dos métodos CSR, CSC, CSV e CDS. A escolha dos métodos se justifica pela intenção de desenvolver algoritmos que possam ser aplicados a qualquer formato de matriz. O CSR, CSC e CSV são métodos aplicáveis a matrizes esparsas, cujo padrão de distribuição de elementos não nulos não apresenta uma forma estruturada ou sistemática, não apresentando limitações quanto ao formato da matriz. O CDS, por outro lado, é indicado para matrizes do tipo banda, tipo este frequentemente encontrado em problemas resolvidos por meio da técnica de diferenças finitas, volumes finitos ou elementos finitos (COELHO, 2014).

Os Quadros 3.1, 3.2, 3.3 e 3.4 mostram os algoritmos dos métodos CSR, CSC, CSV e CDS, respectivamente.

Quadro 3.1: Algoritmo CSR

$cont = 1$
$cont2 = 1$
$para\ i = 1, \ldots, ordem$
    $\boldsymbol{jc}(cont) = cont2$
    $cont = cont + 1$
    $para\ j = 1, \ldots, ordem$
        $se\ \boldsymbol{A}(i,j)\ \neq 0$
            $\boldsymbol{jr}(cont2) = j$
            $\boldsymbol{aa}(cont2) = \boldsymbol{A}(i,j)$
            $cont2 = cont2 + 1$
        $fim\ se$
    $fim\ para$
$fim\ para$
$\boldsymbol{jc}(ordem + 1) = cont2$

Fonte: Autor (2019).

Quadro 3.2: Algoritmo CSC

$cont = 1$
$cont2 = 1$
$para\ j = 1, \ldots, ordem$
    $\boldsymbol{jc}(cont) = cont2$
. $cont = cont + 1$
    $para\ i = 1, \ldots, ordem$
      $se\ \boldsymbol{A}(i,j)\ \neq 0$
        $\boldsymbol{ja}(cont2) = i$
        $\boldsymbol{aa}(cont2) = \boldsymbol{A}(i,j)$
        $cont2 = cont2 + 1$
      $fim\ se$
    $fim\ para$
$fim\ para$
$\boldsymbol{jc}(ordem + 1) = cont2$

Fonte: Autor (2019).

Quadro 3.3: Algoritmo CSV

$cont = 1$
$cont2 = 1$
$para\ i = 1, \ldots, ordem$
    $para\ j = 1, \ldots, ordem$
      $se\ \boldsymbol{A}(i,j)\ \neq 0$
        i$\boldsymbol{a}(cont2) = cont$
        $\boldsymbol{aa}(cont2) = \boldsymbol{A}(i,j)$
        $cont2 = cont2 + 1$
        $cont = 0$
      $fim\ se$
      $cont = cont + 1$
    $fim\ para$
$fim\ para$
$\boldsymbol{aa}(não\ nulos + 1) = nº\ de\ linhas$
$\boldsymbol{ia}(não\ nulos + 1) = nº\ de\ colunas$

Fonte: Autor (2019).

Quadro 3.4: Algoritmo CDS

```
Para as diagonais abaixo da principal e a
principal
cont=1
para Diagonal=ordem, ... ,1
   para i=Diagonal, ... ,ordem
      se i=Diagonal
         j=1
      fim se
      se A(i,j)≠0
         VAL(cont,1)=-Diagonal+1
         VAL(cont,i+1)=A(i,j)
      fim se
      j=j+1
   fim para
   se a diagonal for não nula
      cont=cont+1
   fim se
fim para
Para diagonais acima da principal
para Diagonal=2, ... ,ordem
   para j=Diagonal, ... ,ordem
      se j=Diagonal
         i=1
      fim se
      se A(i,j)≠0
         VAL(cont,1)=Diagonal-1
         VAL(cont,i+1)=A(i,j)
      fim se
      i=i+1
   fim para
   se a diagonal for não nula
      cont=cont+1
   fim se
fim para
```

Fonte: Autor (2019).

Quanto ao formato das variáveis para o CSR adotou-se $\boldsymbol{aa}$ no formato *double* (8 *bytes*), $\boldsymbol{jr}$ no formato int16 (2 *bytes*) e $\boldsymbol{jc}$ no formato int32 (4 $bytes$); para o CSC adotou-se $\boldsymbol{aa}$ no formato *double* (8 *bytes*), $\boldsymbol{ja}$ no formato int16 (2 *bytes*) e $\boldsymbol{jc}$ no formato int32 (4 $bytes$); para o CSV adotou-se $\boldsymbol{aa}$ no formato *double* (8 *bytes*) e $\boldsymbol{ia}$ no formato int16 (2 *bytes*); para o CDS adotou-se $\boldsymbol{VAL}$ no formato *double* (8 *bytes*). Os formatos adotados foram pensados com o intuito de atender as dimensões e esparsidades de matrizes abordados no presente trabalho, podendo ser alterado para problemas com dimensões diferentes.

3.2 Algoritmos dos métodos iterativos

Dando sequência, foram criados algoritmos dos principais métodos iterativos adaptados à utilização dos dados dos métodos de compressão, trabalhando apenas com os elementos não nulos da matriz. Foram implementados os métodos de Jacob, Gauss-Seidel e Gradiente Conjugado.

Os Quadros 3.5 a 3.7 apresentam os algoritmos dos métodos Jacobi, Gauss-Seidel e Gradiente Conjugado para o método de compressão CSR, respectivamente.

Quadro 3.5: Algoritmo de Jacobi para estrutura CSR

```
k=1
enquanto Critério de Parada>tolerância
    cont=0
    para i=1, ... , ordem
        para ii=1, ... ,jc(i+1)-jc(i)
          cont=cont+1
          se jr(cont)≠i
              x^(k+1)(i)=x^(k+1)(i) - x^(k)(jr(cont))·aa(cont)
          senão
            x^(k+1)(i)=x^(k+1)(i)+b(i)
            j=cont
          fim
        fim
        x^(k+1)(i)=x^(k+1)(i)/aa(j)
    fim
    Cálculo do Critério de Parada
    x^(k) = x^(k+1)
    k=k+1
fim
```

Fonte: Autor (2019).

Quadro 3.6: Algoritmo de Gauss-Seidel para estrutura CSR

```
k=1
enquanto Critério de Parada>tolerância
    cont=0
    para i=1, ... ,ordem
        para ii=1, ... ,jc(i+1)-jc(i)
            cont=cont+1
            se jr(cont)>i
                x^(k+1)(i)=x^(k+1)(i) - x^(k)(jr(cont))·aa(cont)
            senão se jr(cont)<i
                x^(k+1)(i)=x^(k+1)(i) - x^(k+1)(jr(cont))·aa(cont)
            senão
                x^(k+1)(i)=x^(k+1)(i)+b(i)
                j=cont
            fim
        fim
        x^(k+1)(i)=x^(k+1)(i)/aa(j)
    fim
    Cálculo do Critério de Parada
    x^(k) = x^(k+1)
    k=k+1
fim
```

Fonte: Autor (2019).

Quadro 3.7: Algoritmo do Gradiente Conjugado para estrutura CSR

(Continua)

```
para j = 1, ... ,ordem
  para i = jc(j), ... ,jc(j+1)-1
        bb(jr(i)) = bb(jr(i)) + aa(i)·b(j)
  fim
fim
r=bb
aux=r^T ·r
v=bb
k=1
enquanto Critério de Parada>tolerância
  para i=1, ... ,ordem
```

Quadro 3.7: Algoritmo do Gradiente Conjugado para estrutura CSR
(Conclusão)

```
    para j = jc(i), ... ,jc(i+1)-1
      z(i) = z(i) + aa(j) · v(jr(j))
    fim
  fim
  para j = 1, ... ,ordem
    para i = jc(j), ... ,jc(j+1)-1
      zz(jr(i)) = zz(jr(i)) + aa(i) · z(j)
    fim
  fim
  s^(k) = aux/(v' · zz)
  x^(k+1) = x^(k) + s^(k) · v
  r = r - s^(k) · zz
  aux1 = r' · r
  v = r + (aux1/aux) · v
  aux = aux1
  Cálculo do Critério de Parada
  k = k + 1
  x^(k) = x^(k+1)
fim
```

Fonte: Autor (2019).

Os Quadros 3.8 a 3.10 apresentam os algoritmos dos métodos Jacobi, Gauss-Seidel e Gradiente Conjugado para o método de compressão CSC, respectivamente.

Para o método de Gauss-Seidel foi necessário realizar um processamento prévio à resolução do sistema, pois o método apresenta uma atualização imediata dos resultados do vetor $\boldsymbol{x}$, ou seja, para cada posição de $\boldsymbol{x}$ é necessário conhecer os valores anteriores a ele, sendo que cada valor de $\boldsymbol{x}$ corresponde ao processamento de uma linha da matriz. Desta forma, para a solução do problema é necessário que os elementos sejam processados ordenadamente de acordo com as linhas da matriz. Como o CSC armazena os elementos ordenados conforme a ordem das colunas, não é possível processar os elementos do modo em que se encontram armazenados. A alternativa encontrada para solucionar o problema foi a criação de três vetores adicionais: o vetor $\boldsymbol{ii}(N_z)$ para guardar o índice dos elementos

ordenados de acordo com as linhas; o $\boldsymbol{j}(N_z)$ para armazenar a coluna do elemento correspondente; e o $\boldsymbol{ic}$(n) a quantidade de elementos de cada linha. Vale ressaltar que essa pode não ser a única forma de contornar o problema, podendo existir outras mais eficientes.

Salienta-se ainda que se optou pela criação de novos vetores para diminuir o número de operações nas iterações. Caso esses vetores não fossem criados, haveria uma grande quantidade de operações para cada iteração realizada. Para exemplificar a metodologia abordada, para a matriz comprimida na forma CSC representada abaixo, tem-se os vetores $\boldsymbol{ii}$, $\boldsymbol{ja}$ e $\boldsymbol{jc}$.

$$\boldsymbol{aa} = [1 \quad 5 \quad 10 \quad 3 \quad 2 \quad 6 \quad 8 \quad 4 \quad 9 \quad 7 \quad 11]$$
$$\boldsymbol{ja} = [1 \quad 3 \quad 5 \quad 2 \quad 1 \quad 3 \quad 4 \quad 2 \quad 4 \quad 3 \quad 5]$$
$$\boldsymbol{jc} = [1 \quad 4 \quad 5 \quad 8 \quad 10 \quad 12]$$

$$\boldsymbol{ii} = [1 \quad 5 \quad 4 \quad 8 \quad 2 \quad 6 \quad 10 \quad 7 \quad 9 \quad 3 \quad 11]$$
$$\boldsymbol{ja} = [1 \quad 3 \quad 2 \quad 4 \quad 1 \quad 3 \quad 5 \quad 3 \quad 4 \quad 1 \quad 5]$$
$$\boldsymbol{jc} = [2 \quad 2 \quad 3 \quad 2 \quad 2]$$

Quadro 3.8: Algoritmo de Jacobi para estrutura CSC

```
k=1
enquanto Critério de Parada>tolerância
    cont=1
    para j=1, ... , ordem
        para jj=1, ... ,jc(j+1)-jc(j)
          se ja(cont)≠j
              x^(k+1)(ja(cont))=x^(k+1)(ja(cont)) - x^(k)(i)·aa(cont)
          senão
              x^(k+1)(j)=x^(k+1)(j)+b(j)
              J(j)=cont
          fim
          cont=cont+1
        fim
    fim
    para i=1, ... , ordem
        x^(k+1)(i)=x^(k+1)(i)/aa(j(i))
    fim
    Cálculo do Critério de Parada
    x^(k) = x^(k+1)
    k=k+1
fim
```

Fonte: Autor

Quadro 3.9: Algoritmo de Gauss-Seidel para estrutura CSC

```
k=1
enquanto Critério de Parada>tolerância
   l=0
     para i=1, ... , ordem
          para l=l+1, ... , ic(i)
            se i<j(i)
                 x^(k+1)(i)=x^(k+1)(i) - x^(k)(j(i))·aa(ii(l))
            senão se
                 x^(k+1)(i)=x^(k+1)(i) - x^(k+1)(j(i))·aa(ii(l))
            senão
                 x^(k+1)(i)=x^(k+1)(i)+b(j(i))
                 aux=j(i)
            fim
          fim
     fim
     x^(k+1)(i)=x^(k+1)(i)/aa(aux)
     Cálculo do Critério de Parada
     x^(k) = x^(k+1)
     k=k+1
fim
```

Fonte: Autor

Quadro 3.10: Algoritmo do Gradiente Conjugado para estrutura CSC

(Continua)

```
para j = 1, ... ,ordem
   para i = jc(j), ... ,jc(j+1)-1
              bb(j) = bb(j) + aa(i)·b(ja(i))
   fim
fim
r=bb
aux=r^T ·r
v=bb
k=1
```

Quadro 3.10: Algoritmo do Gradiente Conjugado para estrutura CSC

(Conclusão)

```
enquanto Critério de Parada>tolerância
  para i=1, ... ,ordem
    para j = jc(i), ... ,jc(i+1)-1
      z(ja(j)) = z(a(j)) + aa(j) · v(i)
    fim
  fim
  para j = 1, ... ,ordem
    para i = jc(j), ... ,jc(j+1)-1
      zz(j) = zz(j) + aa(i) · z(ja(i))
    fim
  fim
  s^(k) =aux/(v'·zz)
  x^(k+1) =x^(k) +s^(k) · v
  r=r-s^(k) · zz
  aux1=r'· r
  v=r+(aux1/aux) · v
  aux=aux1
  Cálculo do Critério de Parada
  k=k+1
  x^(k) =x^(k+1)
fim
```

Fonte: Autor (2019).

Os Quadros 3.11 a 3.13 apresentam os algoritmos dos métodos Jacobi, Gauss-Seidel e Gradiente Conjugado para o método de compressão CSV, respectivamente.

Quadro 3.11: Algoritmo de Jacobi para estrutura CSV

(Continua)

```
k=1
enquanto Critério de Parada>tolerância
    cont=0
    para i=1, ... , ordem
        enquanto soma ≤ i · ordem
          se resto da divisão soma/ordem≠0
              j =resto da divisão soma/ordem
          senão
              j =ordem
          fim
          se i≠j
              x^(k+1) (i)=x^(k+1) (i) - x^(k) (j)· aa(cont)
```

Quadro 3.11: Algoritmo de Jacobi para estrutura CSV

(Conclusão)

```
            senão
              x^(k+1)(i)=x^(k+1)(i)+b(i)
              jj=cont
            fim
            cont=cont+1
            soma=soma+ia(cont)
          fim
          x^(k+1)(i)=x^(k+1)(i)/aa(jj)
    fim
    Cálculo do Critério de Parada
    x^(k) = x^(k+1)
    k=k+1
fim
```

Fonte: Autor (2019).

Quadro 3.12: Algoritmo de Gauss-Seidel para estrutura CSV

```
k=1
enquanto Critério de Parada>tolerância
    cont=0
    para i=1, … , ordem
        enquanto soma ≤ i · ordem
          se resto da divisão soma/ordem≠0
              j=resto da divisão soma/ordem
          senão
              j=ordem
          fim
          se i<j
              x^(k+1)(i)=x^(k+1)(i) - x^(k)(j)·aa(cont)
          senão se i>j
              x^(k+1)(i)=x^(k+1)(i) - x^(k+1)(j)·aa(cont)
          senão
              x^(k+1)(i)=x^(k+1)(i)+b(i)
              jj=cont
          fim
          cont=cont+1
          soma=soma+ia(cont)
        fim
        x^(k+1)(i)=x^(k+1)(i)/aa(jj)
    fim
    Cálculo do Critério de Parada
    x^(k) = x^(k+1)
    k=k+1
fim
```

Fonte: Autor (2019).

Quadro 3.13: Algoritmo de Gradiente Conjugado para estrutura CSV

(Continua)

```
cont=1
soma=ia(cont)
para i = 1, ... ,ordem
  enquanto soma>(i − 1) · ordem e soma ≤ i · ordem
    j= resto da divisão soma/ordem
    se j ≠ 0
      bb(j) = bb(j) + aa(cont)·b(i)
    senão
      bb(ordem) = bb(ordem) + aa(cont)·b(i)
    fim
  fim
fim
r=bb
aux=r^T ·r
v=bb
k=1
enquanto Critério de Parada>tolerância
  cont=1
  soma= soma+ia(cont)
  para i=1, ... ,ordem
    enquanto soma>(i − 1) · ordem e soma ≤ i · ordem
      j= resto da divisão soma/ordem
      se j ≠ 0
        z(i) = z(i) + aa(cont)·b(j)
      senão
        z(i) = z(i) + aa(cont)·b(ordem)
      fim
      cont=cont+1
      soma=soma+ia(cont)
    fim
  fim
  cont=1
  soma= soma+ia(cont)
```

Quadro 3.13: Algoritmo de Gradiente Conjugado para estrutura CSV
(Conclusão)

para i=1, … ,ordem
enquanto soma>$(i-1)\cdot ordem$ *e soma* $\leq i\cdot ordem$
j= resto da divisão $\frac{soma}{ordem}$
se j $\neq 0$
$\boldsymbol{zz}(j)=\boldsymbol{zz}(j)+\boldsymbol{aa}(cont)\cdot \boldsymbol{z}(i)$
senão
$\boldsymbol{zz}(ordem)=\boldsymbol{zz}(ordem)+\boldsymbol{aa}(cont)\cdot \boldsymbol{z}(i)$
fim
cont=cont+1
*soma=soma+***ia***(cont)*
fim
fim
$s^{(k)}=aux/(\boldsymbol{v}'\cdot \boldsymbol{zz})$
$\boldsymbol{x}^{(k+1)}=\boldsymbol{x}^{(k)}+s^{(k)}\cdot \boldsymbol{v}$
$\boldsymbol{r}=\boldsymbol{r}-s^{(k)}\cdot \boldsymbol{zz}$
$aux1=\boldsymbol{r}'\cdot \boldsymbol{r}$
$\boldsymbol{v}=\boldsymbol{r}+(aux1/aux)\cdot \boldsymbol{v}$
aux=aux1
Cálculo do Critério de Parada
k=k+1
$x^{(k)}=x^{(k+1)}$
fim

Fonte: Autor (2019).

Os Quadros 3.11 a 3.13 apresentam os algoritmos dos métodos Jacobi, Gauss-Seidel e Gradiente Conjugado para o método de compressão CDS, respectivamente.

Quadro 3.14: Algoritmo de Jacobi para estrutura CDS

```
k=1
j=índice da diagonal principal
enquanto Critério de Parada>tolerância
    cont=1
    para i=2, ... , ordem+1
        para d=1, ... , nº de diagonais não nulas
            se VAL(d,1)≠0 e VAL(d,c)≠0
                x^(k+1) (i-1)=x^(k+1) (i-1) - x^(k) (VAL(d,1)+cont)· VAL(d,i)
            fim
            x^(k+1) (i-1)=(x^(k+1) (i-1)+b(i-1))/VAL(j,i)
        fim
        cont=cont+1
    fim
    Cálculo do Critério de Parada
    x^(k) = x^(k+1)
    k=k+1
fim
```

Fonte: Autor (2019).

Quadro 3.15: Algoritmo de Gauss-Seidel para estrutura CDS

```
k=1
j=índice da diagonal principal
enquanto Critério de Parada>tolerância
    cont=1
    para i=2, ... , ordem+1
        para d=1, ... , nº de diagonais não nulas
            se VAL(d,1)≠0 e VAL(d,c)>0
                x^(k+1) (i-1)=x^(k+1) (i-1) - x^(k) (VAL(d,1)+cont)· VAL(d,i)
            senão se VAL(d,1)≠0 e VAL(d,c)<0
                x^(k+1) (i-1)=x^(k+1) (i-1) - x^(k+1) (VAL(d,1)+cont)· VAL(d,i)
            fim
            x^(k+1) (i-1)=(x^(k+1) (i-1)+b(i-1))/VAL(j,i)
        fim
        cont=cont+1
    fim
    Cálculo do Critério de Parada
    x^(k) = x^(k+1)
    k=k+1
fim
```

Fonte: Autor (2019).

Quadro 3.16: Algoritmo do Gradiente Conjugado para estrutura CDS

```
para i = 2, ... ,ordem+1
  para d= 1, ... , nº de diagonais não nulas
    se VAL(d,i)≠0
          bb(VAL(d,1)) = bb(VAL(d,1)) + VAL(d,i)·b(i-1)
    end
  fim
Fim
r=bb
aux=r^T ·r
v=bb
k=1
enquanto Critério de Parada>tolerância
 Cont=1;
  para i=2, ... ,ordem+2
    para j = nº de diagonais não nulas
      se VAL(j,i)≠0
          z(i-1) = z(i-1) + VAL(j,i)· v(VAL(j,1)+cont)
      fim
    fim
    cont=cont+1
  fim
  cont=1
  para i = 2, ... ,ordem +1
    para j = nº de diagonais não nulas
      se VAL(j,i)≠0
        zz(VAL(j,1)+cont) = zz(VAL(j,1)+dist) + VAL(j,i)··z(i+1)
      fim
    fim
   cont=cont+1
  fim
  s^(k) =aux/(v'·zz)
  x^(k+1) =x^(k) +s^(k) ·v
  r=r-s^(k) ·zz
  aux1=r'·r
  v=r+(aux1/aux)·v
  aux=aux1
  Cálculo do Critério de Parada
  k=k+1
  x^(k) =x^(k+1)
fim
```

Fonte: Autor (2019).

Quanto ao critério de parada, adotou-se o resíduo relativo gerado pela diferença entre $\boldsymbol{x}^{(k+1)}$ e o valor da solução exata do problema. A solução exata foi calculada antes de iniciar a

solução pelo método iterativo, de forma a não prejudicar o tempo de processamento. Com isso, garantiu-se a convergência da solução para o resultado exato do problema. A Equação (16) demonstra o resíduo utilizado.

$$R(k) = máx\left(\frac{\left|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}_{\boldsymbol{sol}}\right|}{\left|\boldsymbol{x}^{(k+1)}\right|}\right) \tag{16}$$

Como tolerância foi adotado $1 \cdot 10^{-4}$ para todos os casos testados.

3.3 Geração dos sistemas lineares

Para a geração do sistema linear, efetuou-se a discretização por diferenças finitas das Equações (12) e (13). Sabendo que o problema em questão gera um sistema de alta esparsidade, foi necessário adicionar elementos à matriz, reduzindo sua esparsidade. Para isso, foram abordadas duas sistemáticas diferentes: a primeira objetivou a geração de matrizes cujo padrão de distribuição dos elementos não apresente uma forma sistemática, adicionando elementos de forma aleatória à matriz; a segunda teve o intuito de gerar matrizes do tipo banda, adicionando diagonais à matriz.

A primeira sistemática foi criada com o intuito de avaliar a eficiência dos métodos CSR, CSC e CSV, indicados para matrizes sem uma forma estruturada. Já a segunda sistemática foi criada para de avaliar o método CDS, recomendado para matrizes com as diagonais definidas.

No CDS, a posição das diagonais definidas influencia diretamente na quantidade de elementos salvos na compressão. Isso ocorre porque é armazenada uma mesma quantidade de elementos, independente da dimensão da diagonal, sendo proporcional ao tamanho da diagonal principal. Desse modo, uma mesma esparsidade distribuída em diagonais diferentes pode ocasionar uma diferença na demanda de memória. Desta forma, com o propósito de evitar equívocos de avaliação gerados pela distribuição de uma mesma esparsidade em diagonais diferentes, optou-se por padronizar a escolha das diagonais a serem preenchidas. Para todos os casos convencionou-se que as diagonais fossem preenchidas da maior para a menor, de forma a gerar o menor número de diagonais possível. Além disso, todas as diagonais escolhidas como não nulas foram completamente preenchidas, sendo a última diagonal a ser preenchida escolhida de acordo com a quantidade de elementos remanescentes para atingir a esparsidade desejada.

No presente trabalho foram abordadas matrizes com ordens 900, 1600 e 3600 e esparsidades de 30%, 60%, 90% e 98%, optou-se por trabalhar com esparsidades maiores por apresentarem maiores ganhos computacionais, Mendes et al. (2017) afirmam que o ganho computacional está diretamente relacionado com a esparsidade da matriz, e que para esparsidades menores a utilização da compressão não se mostra vantajosa.

3.4 Linguagem de programação

Os algoritmos gerados foram implementados no aplicativo MATLAB e na linguagem de programação científica JULIA, possibilitando comparar o desempenho entre as duas linguagens de programação. Escolheu-se o aplicativo comercial MATLAB por se tratar de uma linguagem de programação de alto desempenho amplamente utilizada por profissionais da engenharia. Contudo, o aplicativo apresenta melhor desempenho quando programado utilizando vetorização, possuindo limitações quando programado de forma geral. A linguagem JULIA, por sua vez é uma linguagem de programação gratuita e de código aberto, projetada para atender os requisitos da computação de alto desempenho numérico e científico, sendo uma linguagem nova construída com herança das linguagens de programação matemática, a qual os pesquisadores afirmam ser mais rápida em tempo de processamento que o MATLAB para a programação não vetorizada (PEREIRA e SIQUEIRA, 2016; FIORITE et al., 2012; BEZANSON et al., 2017; SINAIE, 2017).

O tempo de processamento dos experimentos numéricos foi obtido por meio da média simples do tempo de execução em triplicada dos algoritmos implementados e todos os resultados apresentados no presente trabalho foram gerados com um notebook com as seguintes configurações: Processador Core i7 2.8 GHz, memória RAM de 8 GB e sistema operacional *Windows* 10 - 64 bits.

4 RESULTADOS E DISCUSSÃO

4.1 Eficiência dos métodos de compressão quanto ao uso de memória

As Tabelas 4.1 e 4.2 apresentam o consumo de memória no armazenamento sem fazer uso da compressão e usando a compressão. Além disso, é apresentada a economia de memória, o qual consiste na porcentagem de memória economizada no armazenamento usando os métodos de compressão.

Tabela 4.1: Consumo de memória em Bytes para os métodos CSR e CSC

Ordem	Esparsidade	Sem compressão	CSR	Economia de Memória	CSC	Economia de Memória
900	30%	6.48E+06	5.69E+06 ± 3.14E+02	12.2%	5.69E+06 ± 3.14E+02	12.2%
	60%		3.26E+06 ± 3.95E+03	49.6%	3.26E+06 ± 3.95E+03	49.6%
	90%		8.53E+05 ± 1.71E+03	86.8%	8.53E+05 ± 1.71E+03	86.8%
	98%		2.09E+05 ± 1.33E+03	96.8%	2.09E+05 ± 1.33E+03	96.8%
1600	30%	2.05E+07	1.79E+07 ± 4.50E+03	12.4%	1.79E+07 ± 4.50E+03	12.4%
	60%		1.03E+07 ± 6.08E+03	49.8%	1.03E+07 ± 6.08E+03	49.8%
	90%		2.64E+06 ± 1.10E+03	87.1%	2.64E+06 ± 1.10E+03	87.1%
	98%		5.97E+05 ± 1.55E+03	97.1%	5.97E+05 ± 1.55E+03	97.1%
3600	30%	1.04E+08	9.08E+07 ± 3.05E+04	12.4%	9.08E+07 ± 3.05E+04	12.4%
	60%		5.19E+07 ± 3.67E+04	49.9%	5.19E+07 ± 3.67E+04	49.9%
	90%		1.31E+07 ± 6.58E+03	87.3%	1.31E+07 ± 6.58E+03	87.3%
	98%		2.78E+06 ± 8.78E+02	97.3%	2.78E+06 ± 8.78E+02	97.3%

Fonte: Autor (2019).

Tabela 4.2: Consumo de memória em Bytes para os métodos CSV e CDS

Ordem	Esparsidade	Sem compressão	CSV	Economia de Memória	CDS	nº de Diagonais	Economia de Memória
900	30%	6.48E+06	5.68E+06 ± 3.12E+02	12.3%	5.87E+06	815	9.3%
	60%		3.26E+06 ± 3.95E+03	49.7%	2.93E+06	407	54.7%
	90%		8.49E+05 ± 1.71E+03	86.9%	6.70E+05	93	89.7%
	98%		2.05E+05 ± 1.33E+03	96.8%	1.37E+05	19	97.9%
1600	30%	2.05E+07	1.79E+07 ± 4.50E+03	12.4%	1.86E+07	1449	9.4%
	60%		1.03E+07 ± 6.08E+03	49.8%	9.26E+06	723	54.8%
	90%		2.63E+06 ± 1.10E+03	87.2%	2.11E+06	165	89.7%
	98%		5.90E+05 ± 1.55E+03	97.1%	4.23E+05	33	97.9%
3600	30%	1.04E+08	9.08E+07 ± 3.05E+04	12.4%	9.38E+07	3257	9.5%
	60%		4.88E+07 ± 5.43E+06	52.9%	4.68E+07	1623	54.9%
	90%		1.31E+07 ± 6.58E+03	87.4%	1.07E+07	371	89.7%
	98%		2.76E+06 ± 8.78E+02	97.3%	2.10E+06	73	98.0%

Fonte: Autor (2019).

Observa-se das tabelas que para uma mesma ordem e esparsidade de matriz a economia de memória é relativamente próxima para os diferentes métodos de compressão. Além disso, mesmas esparsidades apresentam valores igualmente similares. De um modo geral, para todos os métodos e tamanhos de matrizes testados, a compressão resultou em economia de memória. Verifica-se também que apesar dos dados apresentarem uma discreta diferença entre os formatos CSV e CSR, os resultados corroboram a afirmação de Farzaneh et al. (2009), a qual declara que o formato CSV é mais viável em termos de redução de memória do que o CSR.

É importante destacar que para o método CDS o número de diagonais não nulas é um fator importante na avaliação do consumo de memória, uma vez que a quantidade de elementos armazenados é diretamente proporcional à quantidade de diagonais. Desta forma, para esse método acrescentou-se a informação do número de diagonais dos casos testados (problemas com maior número de diagonais acarretarão em uma diminuição de desempenho quanto ao consumo de memória).

4.2 Avaliação dos métodos de compressão quanto ao tempo de processamento

A Tabela (4.3) apresenta o tempo necessário para operar a compressão da matriz de acordo com cada método empregado.

Tabela 4.3: Tempo de compressão (s).

Ordem	Esparsidade	CSR	CSC	CSV	CDS
900	30%	5.46E-02 ± 8.73E-04	4.63E-02 ± 9.63E-04	6.14E-02 ± 9.88E-04	1.80E-01 ± 1.94E-02
	60%	4.77E-02 ± 1.41E-04	3.78E-02 ± 2.05E-04	4.98E-02 ± 8.60E-04	5.81E-02 ± 3.40E-04
	90%	2.73E-02 ± 8.34E-04	1.83E-02 ± 3.74E-04	2.34E-02 ± 2.87E-04	2.13E-02 ± 4.92E-04
	98%	2.13E-02 ± 4.11E-04	1.28E-02 ± 9.43E-05	1.49E-02 ± 2.05E-04	1.52E-02 ± 1.70E-04
1600	30%	2.33E-01 ± 2.30E-02	1.73E-01 ± 1.92E-02	2.46E-01 ± 2.36E-02	1.64E+00 ± 1.46E-02
	60%	1.79E-01 ± 5.58E-03	1.20E-01 ± 4.65E-03	1.57E-01 ± 2.21E-03	3.47E-01 ± 5.67E-03
	90%	1.15E-01 ± 1.39E-03	6.10E-02 ± 2.00E-03	8.66E-02 ± 5.44E-04	6.42E-02 ± 1.54E-03
	98%	9.57E-02 ± 3.33E-03	3.87E-02 ± 3.56E-04	5.28E-02 ± 3.56E-04	4.44E-02 ± 8.73E-04
3600	30%	1.05E+00 ± 3.68E-02	7.23E-01 ± 8.52E-03	1.05E+00 ± 3.58E-03	2.14E+01 ± 9.40E-02
	60%	8.86E-01 ± 7.38E-03	5.90E-01 ± 1.44E-02	7.87E-01 ± 1.99E-02	5.48E+00 ± 3.04E-02
	90%	5.71E-01 ± 7.45E-03	2.75E-01 ± 2.20E-03	3.73E-01 ± 1.69E-02	5.00E-01 ± 3.35E-03
	98%	4.79E-01 ± 1.28E-02	1.79E-01 ± 2.36E-03	2.31E-01 ± 9.02E-03	2.53E-01 ± 1.78E-03

Fonte: Autor (2019).

Para todos os casos abordados, o tempo gasto para operar a compressão é inversamente proporcional à esparsidade da matriz Isso porque, quanto maior a esparsidade,

menor a quantidade de elementos armazenados. Além disso, quanto ao tempo gasto por cada método, observa-se que o método CSC se apresentou como o mais rápido, fato que pode ser provavelmente justificado pela forma como as matrizes são operadas no MATLAB, sendo processadas por colunas. A MathWorks recomenda que, ao processar matrizes 2-D ou N-D, os dados sejam acessados e armazenados em colunas.

Segundo McGarrity (2019), o desempenho da memória computacional não aumentou na mesma proporção que o desempenho de CPUs. Atualmente os códigos são limitados pela memória, sendo o desempenho geral limitado pelo tempo necessário para acessar a memória. Ainda em relação ao acesso e armazenamento de dados, McGarrity (2019) afirma que as CPUs modernas usam um cache rápido para reduzir o tempo médio necessário para acessar a memória principal. Códigos alcançam a eficiência máxima do cache quando percorrem locais de memória com aumento monotônico. Como o MATLAB armazena colunas da matriz em locais de memória que aumentam monotonicamente, o processamento de dados em colunas resulta em eficiência máxima do cache.

Nota-se também que o CDS apresentou um desempenho menor na maioria dos casos abordados. Isso explica-se pela complexidade necessária para processar a matriz na sequência necessária para a compressão usando esse método.

4.3 Eficiência dos métodos de resolução quanto à convergência do resultado

Um fator importante na verificação da convergência é o resíduo. Esse deve ser o mesmo, independente do formato de armazenamento utilizado. Isso ocorre porque o formato de armazenamento modifica apenas a forma como a matriz é processada, não apresentando nenhum tipo de alteração nos resultados exibidos pelos métodos iterativos. Desta forma, para um mesmo método iterativo, o perfil do resíduo em função do número de iterações deve ser o mesmo, tanto para a matriz na forma original quanto utilizando os formatos de compressão.

As Figuras 4.1, 4.2 e 4.3 mostram o valor do resíduo em função do número de iterações para os métodos Jacobi, Gauss-Seidel e Gradiente Conjugado, respectivamente. O resíduo apresentado foi obtido da solução de uma matriz 1600x1600 com 98% de esparsidade.

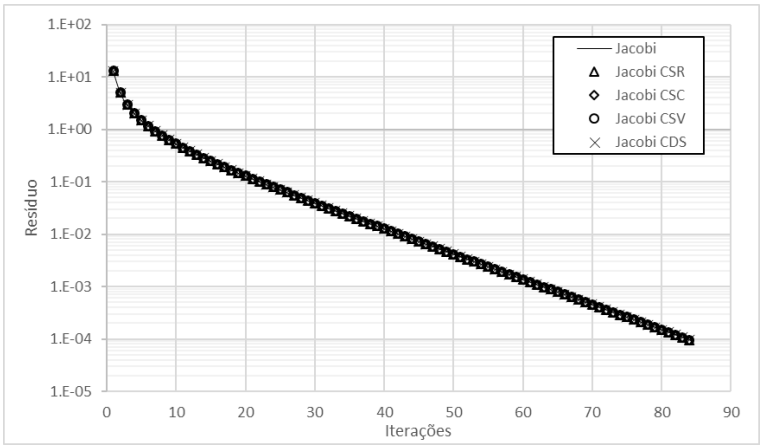

Figura 4.1: Resíduo obtido pelo método Jacobi, matriz 1600x1600 e esparsidade de 98%.

Fonte: Autor (2019).

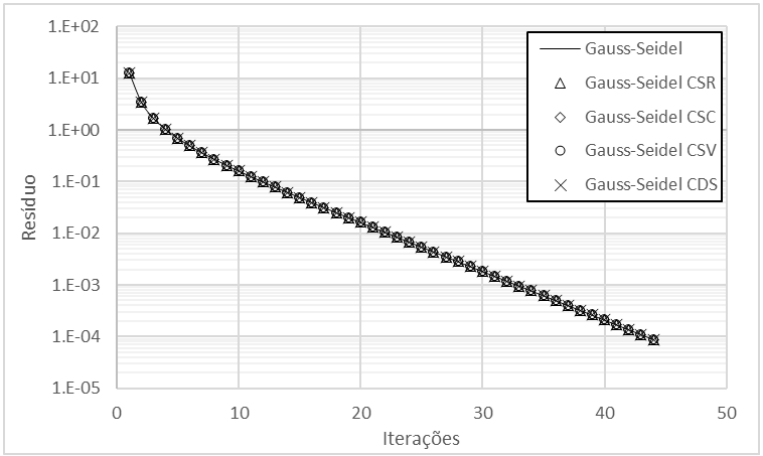

Figura 4.2: Resíduo obtido pelo método Gauss-Seidel, matriz 1600x1600 e esparsidade de 98%.

Fonte: Autor (2019).

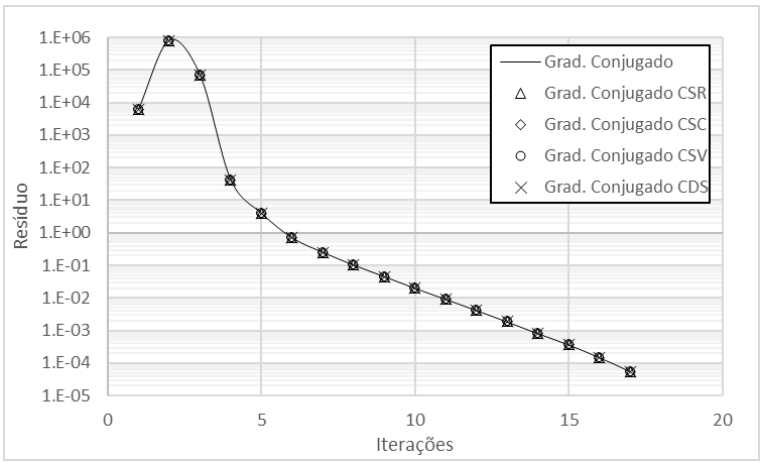

Figura 4.3: Resíduo obtido pelo método Gradiente Conjugado, matriz 1600x1600 e esparsidade de 98%.

Fonte: Autor (2019).

Como pode ser observado, os valores do resíduo coincidem em todos os casos abordados, indicando a correta implementação dos métodos iterativos de solução utilizando o armazenamento na forma comprimida.

4.4 Avaliação dos métodos de resolução quanto ao tempo de processamento

As Tabelas 4.4, 4.5 e 4.6 apresentam os tempos de processamento para solução dos sistemas lineares para os métodos Jacobi, Gauss-Seidel e Gradiente Conjugado, respectivamente.

Tabela 4.4: Tempo de processamento para resolução do sistema de equações lineares para o método de Jacobi (s)

Ordem	IE	Sem compressão	CSR	CSC	CSV	CDS
900	30%	43.46 ± 0.24	21.95 ± 0.13	24.15 ± 0.16	62.15 ± 0.48	33.19 ± 0.45
	60%	25.27 ± 0.52	8.48 ± 0.12	8.94 ± 0.11	22.09 ± 0.31	11.68 ± 0.14
	90%	6.60 ± 0.16	0.92 ± 0.22	0.98 ± 0.02	2.00 ± 0.01	1.36 ± 0.02
	98%	1.66 ± 0.03	0.19 ± 0.00	0.15 ± 0.00	0.25 ± 0.00	0.20 ± 0.00
1600	30%	356.01 ± 35.75	134.53 ± 14.90	152.00 ± 16.53	413.88 ± 42.80	177.07 ± 0.31
	60%	181.02 ± 4.86	42.19 ± 1.59	46.78 ± 1.46	121.72 ± 4.47	59.22 ± 0.10
	90%	47.23 ± 0.16	4.57 ± 0.03	4.47 ± 0.12	12.06 ± 0.07	5.60 ± 0.04
	98%	10.96 ± 0.22	0.64 ± 0.02	0.56 ± 0.02	1.10 ± 0.02	0.66 ± 0.00
3600	30%	3561.10 ± 45.19	1230.37 ± 5.05	1524.20 ± 172.97	4007.27 ± 5.59	1938.37 ± 5.54
	60%	2049.20 ± 20.09	422.38 ± 4.86	480.34 ± 3.33	1350.83 ± 0.29	620.76 ± 0.81
	90%	514.05 ± 4.59	35.54 ± 0.08	37.32 ± 0.31	96.97 ± 0.90	47.66 ± 0.17
	98%	108.91 ± 0.47	3.31 ± 0.03	2.97 ± 0.05	6.38 ± 0.24	4.16 ± 0.10

Fonte: Autor (2019).

Quanto ao método de Jacobi, observa-se que os métodos CSR e CSC apresentam um menor tempo de processamento, seguido pelos métodos CDS e CSV. Esse resultado pode ser justificado pelo padrão de armazenamento dos métodos. Ambos os métodos armazenam dados da linha e coluna do respectivo elemento na matriz. As técnicas de compressão CSV e CDS demandam uma maior quantidade de operações para encontrar a linha e coluna do respectivo elemento na matriz.

Tabela 4.5: Tempo de processamento em segundos para resolução do sistema de equações lineares para o método de Gauss-Seidel

Ordem	IE	Sem compressão	CSR	CSC	CSV	CDS
900	30%	22.27 ± 0.04	12.94 ± 0.09	23.80 ± 0.21	31.69 ± 0.37	731.80 ± 4.77
	60%	13.12 ± 0.28	4.96 ± 0.08	10.19 ± 0.13	11.22 ± 0.22	237.72 ± 1.08
	90%	3.44 ± 0.06	0.60 ± 0.00	1.91 ± 0.05	1.03 ± 0.00	15.83 ± 0.37
	98%	0.96 ± 0.02	0.11 ± 0.00	0.45 ± 0.01	0.14 ± 0.00	0.80 ± 0.01
1600	30%	181.67 ± 17.17	80.22 ± 8.57	131.71 ± 2.03	210.15 ± 22.11	4158.43 ± 12.24
	60%	84.25 ± 8.40	24.44 ± 0.23	53.76 ± 0.23	60.92 ± 1.35	1351.77 ± 1.73
	90%	24.42 ± 0.23	2.60 ± 0.02	8.67 ± 0.05	6.14 ± 0.03	86.99 ± 0.26
	98%	5.82 ± 0.08	0.37 ± 0.01	1.75 ± 0.01	0.60 ± 0.01	3.90 ± 0.04
3600	30%	1807.00 ± 8.01	731.02 ± 8.64	1272.58 ± 8.12	2039.67 ± 3.37	Não avaliado
	60%	1041.00 ± 7.19	251.02 ± 1.86	597.77 ± 8.47	688.75 ± 8.70	Não avaliado
	90%	260.41 ± 1.90	20.51 ± 0.07	92.93 ± 0.04	48.51 ± 0.66	Não avaliado
	98%	56.31 ± 0.08	1.90 ± 0.01	16.10 ± 0.28	3.21 ± 0.10	Não avaliado

Fonte: Autor (2019).

Pode-se observar que o CSC demandou mais tempo no método de Gauss-Seidel que no de Jacobi, fato que se justifica pelo preparo necessário para iniciar o processamento. Como explicado na seção 3.2, a forma de armazenamento do CSC impossibilita o processamento de forma direta do Gauss-Seidel, sendo necessário a criação de novos vetores. O tempo demandado nessa operação fez com que o método de Gauss-Seidel usando CSC tivesse uma perda de eficiência. Vale ressaltar que essa não é a única forma de contornar o problema, podendo existir outras mais eficientes.

Ainda com relação ao método de Gauss-Seidel, observou-se também que o método apresentou uma grande perda de desempenho com relação ao Jacobi para o formato CDS. Quanto a isso não se encontrou uma explicação plausível, uma vez que o método Gauss-Seidel converge com um número menor de iterações e ambos os métodos não apresentam uma diferença que pudesse justificar a redução de desempenho. Além disso, ao processar o algoritmo usando a ferramenta *Rum and Time* do MATLAB, esse comportamento se inverte e o Gauss-Seidel passa a apresentar um tempo de processamento menor que o de Jacobi, conforme o esperado. Dessa forma, os valores obtidos para o tempo de processamento desse método não podem ser levados em consideração no resultado final do presente trabalho, sendo necessário um estudo mais aprofundado para descobrir a origem do problema.

De um modo geral, o CSR foi o que apresentou melhor desempenho para o método Gauss-Seidel, o que está condicionado à ordem com que os elementos são guardados nesse formato de compressão. O CSR armazena os elementos ordenados pelas linhas da matriz, mesmo padrão usado pelo Gauss-Seidel na solução do sistema linear. O CSV, embora

armazene os elementos ordenados por linha, não guarda informações diretas da posição do respectivo elemento na matriz, o que pode justificar o menor desempenho em termos de tempo de processamento.

Tabela 4.6: Tempo de processamento para resolução do sistema de equações lineares para o método Gradiente Conjugado (s)

Ordem	IE	Sem compressão	CSR	CSC	CSV	CDS
900	30%	12.13 ± 0.13	0.17 ± 0.01	0.17 ± 0.01	0.62 ± 0.05	1.07 ± 0.00
	60%	12.09 ± 0.10	0.13 ± 0.00	0.12 ± 0.00	0.45 ± 0.01	0.78 ± 0.00
	90%	12.22 ± 0.11	0.07 ± 0.00	0.07 ± 0.00	0.20 ± 0.01	0.41 ± 0.00
	98%	12.29 ± 0.06	0.05 ± 0.00	0.05 ± 0.00	0.10 ± 0.01	0.13 ± 0.00
1600	30%	68.27 ± 0.54	0.48 ± 0.00	0.47 ± 0.00	1.86 ± 0.01	3.41 ± 0.02
	60%	68.31 ± 0.55	0.37 ± 0.01	0.37 ± 0.01	1.43 ± 0.00	3.05 ± 0.80
	90%	68.71 ± 0.56	0.15 ± 0.00	0.15 ± 0.00	0.52 ± 0.01	1.38 ± 0.13
	98%	69.27 ± 0.32	0.10 ± 0.00	0.09 ± 0.00	0.24 ± 0.00	0.43 ± 0.01
3600	30%	796.08 ± 7.54	2.43 ± 0.05	2.38 ± 0.06	9.91 ± 0.19	18.75 ± 1.64
	60%	749.38 ± 1.16	1.57 ± 0.01	1.54 ± 0.02	6.35 ± 0.04	13.90 ± 0.44
	90%	760.63 ± 6.38	0.62 ± 0.01	0.62 ± 0.00	2.38 ± 0.02	7.65 ± 0.64
	98%	763.30 ± 8.51	0.29 ± 0.02	0.29 ± 0.02	0.88 ± 0.05	3.42 ± 0.21

Fonte: Autor (2019).

Com relação ao Gradiente Conjugado, todos os métodos de compressão apresentaram um ganho significativo no tempo de processamento, sendo o CSC o que apresentou melhor desempenho.

As Tabelas 4.7, 4.8 e 4.9 apresentam a porcentagem de redução no tempo de processamento ocasionado pelos métodos de compressão

Tabela 4.7: Redução do tempo de processamento para o método CSR

Ordem	IE	Jacobi	Gauss-Seidel	Gradiente Conjugado
900	30%	49.50%	41.88%	98.58%
	60%	66.45%	62.20%	98.96%
	90%	86.11%	82.44%	99.45%
	98%	88.75%	88.28%	99.61%
1600	30%	62.21%	55.84%	99.30%
	60%	76.70%	70.99%	99.46%
	90%	90.31%	89.34%	99.78%
	98%	94.15%	93.68%	99.86%
3600	30%	65.45%	59.55%	99.69%
	60%	79.39%	75.89%	99.79%
	90%	93.09%	92.12%	99.92%
	98%	96.96%	96.62%	99.96%

Fonte: Autor (2019).

Tabela 4.8: Redução do tempo de processamento para o método CSC

Ordem	IE	Jacobi	Gauss-Seidel	Gradiente Conjugado
900	30%	44.43%	-6.86%	98.62%
	60%	64.62%	22.32%	98.98%
	90%	85.10%	44.61%	99.45%
	98%	90.65%	53.07%	99.61%
1600	30%	57.30%	27.50%	99.31%
	60%	74.16%	36.18%	99.46%
	90%	90.53%	64.47%	99.78%
	98%	94.88%	69.93%	99.86%
3600	30%	57.20%	29.58%	99.70%
	60%	76.56%	42.58%	99.79%
	90%	92.74%	64.31%	99.92%
	98%	97.27%	71.41%	99.96%

Fonte: Autor (2019).

Tabela 4.9: Redução do tempo de processamento para o método CSV

Ordem	IE	Jacobi	Gauss-Seidel	Gradiente Conjugado
900	30%	-43.02%	-42.27%	94.85%
	60%	12.58%	14.46%	96.27%
	90%	69.75%	70.06%	98.34%
	98%	85.04%	85.53%	99.20%
1600	30%	-16.26%	-15.68%	97.28%
	60%	32.76%	27.69%	97.91%
	90%	74.47%	74.87%	99.24%
	98%	89.94%	89.77%	99.66%
3600	30%	-12.53%	-12.88%	98.75%
	60%	34.08%	33.84%	99.15%
	90%	81.14%	81.37%	99.69%
	98%	94.14%	94.29%	99.88%

Fonte: Autor (2019).

Tabela 4.10: Redução do tempo de processamento para o método CDS

Ordem	IE	Jacobi	Gauss-Seidel	Gradiente Conjugado
900	30%	23.64%	-3185.47%	91.15%
	60%	53.77%	-1712.21%	93.53%
	90%	79.38%	-359.81%	96.61%
	98%	88.09%	16.58%	98.95%
1600	30%	50.26%	-2188.99%	95.01%
	60%	67.29%	-1504.50%	95.53%
	90%	88.14%	-256.28%	97.99%
	98%	94.01%	32.94%	99.38%
3600	30%	45.57%	Não Avaliado	97.64%
	60%	69.71%	Não Avaliado	98.15%
	90%	90.73%	Não Avaliado	98.99%
	98%	96.18%	Não Avaliado	99.55%

Fonte: Autor (2019).

Pela análise das Tabelas 4.7 a 4.10, tem-se que o método CSR apresenta redução no tempo de processamento em todos os casos avaliados. O CSC apresentou resultados positivos na maioria dos casos testados, tendo uma perda de desempenho apenas para o método de Gauss-Seidel. O CSV também apresenta economia no tempo de processamento, com exceção das matrizes de menor esparsidade (30% de esparsidade). Quanto ao CDS, observou-se resultados positivos para os métodos de Jacobi e Gradiente Conjugado, não sendo possível avaliar corretamente o método de Gauss-Seidel.

Outra análise possível seria somar o tempo demandado no processamento da compressão ao requerido para a solução do sistema. No entanto, observa-se que o tempo para compressão da matriz é muito pequeno se comparado ao necessário para solução do problema. Desta forma, pode-se considerar que o mesmo não apresenta uma influência significativa no tempo total.

4.5 Comparação entre a linguagem JULIA e o aplicativo MATLAB

Com o intuito de comparar o desempenho da linguagem JULIA e o aplicativo MATLAB, foram implementados na linguagem JULIA os algoritmos do Gradiente

Conjugado para o CSR, CSC e CSV. A escolha desses três métodos se deve ao fato de terem apresentado o melhor desempenho no MATLAB.

A Tabela 4.11 apresenta os resultados relativos ao tempo demandado para comprimir as matrizes.

Tabela 4.11: Tempo de compressão em segundos para a linguagem JULIA

Ordem	Esparsidade	CSR	CSC	CSV
900	30%	5.90E-02 ± 4.08E-03	5.37E-02 ± 5.31E-03	5.13E-02 ± 1.25E-03
	60%	4.13E-02 ± 2.49E-03	3.33E-02 ± 2.36E-03	3.50E-02 ± 1.63E-03
	90%	1.03E-02 ± 4.71E-04	1.03E-02 ± 1.70E-03	1.27E-02 ± 2.05E-03
	98%	6.33E-03 ± 9.43E-04	8.33E-03 ± 4.03E-03	6.33E-03 ± 1.89E-03
1600	30%	1.97E-01 ± 1.11E-02	9.16E-01 ± 2.85E-02	1.86E-01 ± 2.82E-02
	60%	1.33E-01 ± 1.19E-02	9.67E-02 ± 1.25E-03	1.91E-01 ± 5.66E-02
	90%	5.23E-02 ± 7.04E-03	4.00E-02 ± 1.63E-03	4.00E-02 ± 1.63E-03
	98%	2.53E-02 ± 4.71E-04	2.00E-02 ± 2.94E-03	3.20E-02 ± 6.16E-03
3600	30%	4.40E+00 ± 2.31E-01	3.93E+00 ± 1.50E-01	3.96E+00 ± 6.82E-02
	60%	4.09E+00 ± 4.90E-01	3.56E+00 ± 2.31E-01	3.54E+00 ± 2.93E-02
	90%	3.10E-01 ± 1.25E-03	2.05E-01 ± 5.73E-03	6.04E-01 ± 1.74E-02
	98%	2.15E-01 ± 9.74E-03	1.29E-01 ± 1.18E-02	1.84E-01 ± 3.74E-03

Fonte: Autor (2019).

Observa-se que novamente o CSC demonstrou ser mais rápido devido à forma como as matrizes são operadas.

A Tabela 4.12 apresenta um comparativo entre os dados de tempo de compressão do MATLAB e da linguagem JULIA, mostrando a porcentagem de redução no tempo de processamento para a linguagem JULIA.

Tabela 4.12: Redução do tempo de processamento da compressão para a linguagem JULIA

Ordem	IE	CSR	CSC	CSV
900	30%	-8.0%	-15.9%	16.3%
	60%	13.3%	11.7%	29.7%
	90%	62.1%	43.5%	45.8%
	98%	70.2%	35.1%	57.6%
1600	30%	15.4%	-430.6%	24.3%
	60%	25.5%	19.5%	-21.9%
	90%	54.5%	34.4%	53.8%
	98%	73.5%	48.3%	39.4%
3600	30%	-320.0%	-442.9%	-275.6%
	60%	-361.6%	-503.0%	-349.9%
	90%	45.8%	25.2%	-61.9%
	98%	55.1%	27.7%	20.2%

Fonte: Autor (2019).

Da Tabela 4.12 tem-se que a linguagem JULIA apresenta resultados positivos para a maioria dos casos avaliados, demonstrando perda de desempenho apenas para os menores índices de esparsidade.

As Tabelas 4.13 e 4.14 mostram o tempo de processamento do método gradiente conjugado e a redução do tempo de processamento comparado com o aplicativo MATLAB, respectivamente.

Tabela 4.13: Tempo de processamento para resolução do sistema de equações lineares para o método Gradiente Conjugado (s)

Ordem	IE	CSR	CSC	CSV
900	30%	0.01 ± 0.00	1.72 ± 0.10	0.04 ± 0.00
	60%	0.01 ± 0.00	1.50 ± 0.23	0.03 ± 0.00
	90%	0.01 ± 0.00	0.46 ± 0.02	0.01 ± 0.00
	98%	0.00 ± 0.00	0.19 ± 0.01	0.01 ± 0.00
1600	30%	0.05 ± 0.01	5.47 ± 1.67	0.13 ± 0.00
	60%	0.04 ± 0.01	3.27 ± 0.06	0.10 ± 0.00
	90%	0.01 ± 0.00	1.30 ± 0.03	0.04 ± 0.00
	98%	0.01 ± 0.00	0.60 ± 0.07	0.02 ± 0.00
3600	30%	0.21 ± 0.01	23.47 ± 0.40	0.62 ± 0.02
	60%	0.15 ± 0.01	16.20 ± 1.13	0.40 ± 0.01
	90%	0.06 ± 0.01	5.53 ± 0.54	0.16 ± 0.01
	98%	0.02 ± 0.00	3.59 ± 1.08	0.05 ± 0.00

Fonte: Autor (2019).

Tabela 4.14: Redução do tempo de processamento para a linguagem JULIA

Ordem	IE	CSR	CSC	CSV
900	30%	92%	-929%	93%
	60%	92%	-1117%	93%
	90%	90%	-578%	93%
	98%	95%	-301%	95%
1600	30%	90%	-1064%	93%
	60%	90%	-793%	93%
	90%	91%	-747%	92%
	98%	94%	-531%	92%
3600	30%	91%	-887%	94%
	60%	90%	-953%	94%
	90%	90%	-798%	93%
	98%	93%	-1156%	94%

Fonte: Autor (2019).

Com relação ao aumento de tempo de processamento apresentado pelo CSC, não se encontrou uma justificativa coerente para tais resultados, uma vez que os algoritmos foram implementados da mesma forma tanto no aplicativo Matlab quanto na liguagem Julia. Desponta-se que mais estudos são necessários para evitar qualquer afirmação prematura, principalmente ao que tange a escrita de códigos eficientes para computação numérica e especificidades de cada linguagem.

Nota-se que o Gradiente Conjugado usando o método CSR apresenta-se como o mais viável para utilização, sendo o que apresentou menor tempo de processamento. Com relação à redução no tempo de processamento, observa-se que o CSR e o CSV apresentaram elevada diminuição para todos os casos abordados, ressaltando a vantagem de utilização da linguagem científica JULIA.

5 CONCLUSÃO

À luz dos resultados obtidos no presente trabalho, as seguintes conclusões foram obtidas:

- com relação ao consumo de memória, conclui-se que todos os formatos de armazenamento provêm economia, sendo que para uma mesma ordem e grau de esparsidade todos os modelos apresentam resultados relativamente próximos. Os resultados apontaram para uma economia de memória de 9% para a menor esparsidade de 30%, chegando a 98% para o maior grau de esparsidade 98%;

- quanto ao tempo demandado para operar a compressão, o esquema CSC demonstrou ser o mais rápido, apresentando resultados entre $1{,}28 \cdot 10^{-2}$ e $7{,}2 \cdot 10^{-1}$ segundos. Com relação aos demais métodos, todos os casos testados apresentaram tempo de compressão entre $1{,}28 \cdot 10^{-2}$ e 21,4 segundos;

- com relação aos métodos de solução de sistemas lineares verificou-se que todos eles determinaram a solução dos sistemas, independente do formato de compressão utilizado. Ou seja, os métodos com e sem compressão apresentaram convergência para a solução do problema;

- quanto à relação entre esquema de compressão e método iterativo para resolução de sistemas lineares, tem-se que para o método de Jacobi os modelos CSR e CSC apresentaram melhor desempenho, com redução no tempo de processamento variando entre 44% e 97%. Para o Gauss-Seidel o formato que apresentou melhor desempenho foi o CSR, com a economia no tempo de processamento variando entre 42 e 97%. Por fim, no caso do Gradiente Conjugado ambos os esquemas apresentaram elevados ganhos computacionais, sendo o CSC e o CSR os que resultaram em menor tempo de processamento, com redução no tempo de processamento acima de 98% para todos os casos testados.

- fazendo um comparativo entre os métodos iterativos, tem-se que o Gradiente Conjugado foi o que proveu o melhor desempenho, apresentando redução no tempo de processamento superior a 90% em todos os casos testados.

- com relação à linguagem de programação científica JULIA, conclui-se que ela pode prover elevada economia no tempo de processamento dos métodos. Observou-se que para os métodos CSR e CSV a redução do tempo de processamento para o Gradiente Conjugado foi superior a 90% em todos os casos abordados.

REFERÊNCIAS

ALMEIDA, V.S. **Análise da Interação Solo Não-Homogêneo/Estrutura Via Acoplamento MEC/MEF**. 2003. Tese (Doutorado em Engenharia de Estruturas) –Universidade de São Paulo, São Carlos, 2003.

BATHE, K.-J.; WILSON, E. L. 1976. **Numerical Methods in Finite Element Analysis**. Prentice-Hall, Englewood Cliffs, New Jersey.

BENZANSON, J.; KARPINSKI, S.; SHAH, V. EDELMAN, A. et al. **JULIA Language Documentation**. Jan. 2017

BRAMELLER, A.; ALLAN, R.N.; HAMAM Y.M., **Sparsity**, Pitman, New York, 1976.

COELHO, M. A. O. 2014. **Métodos Iterativos para Resolver Sistemas de Equações Algébricas Lineares em Estruturas Esparsas**. Dissertação de Mestrado, Universidade Federal Fluminense.

COLAÇO, A.; COSTA, P. A.; LOPES, P. **Análise numérica da alteracão do estado de tensão geomecânico induzida pelo tráfego ferroviário**: Elsevier España, Catalunya, v. 31, n. 2, p. 120-131, jul. 2014.

CONNOR, D. 1984. **An introduction to sparse matrices** The irish mathematical Society, 15.

CUNHA, M. C. C. 2000. **Métodos Numéricos**. Campinas, Editora Unicamp.

DUFF, I.S. 1997. **A survey of Sparse Matriz Research**, Proc. IEE, v 65, p. 500-535

FARZANEH, A.; KHEIRI H.; SHAHMERSI M. A. 2009. **An Efficient Storage Format for Large Saparse Matrices. Commun**. Fac. Sci. Uni. Ank. Series, v. 58, n. 2, p. 1-10.

FIORITE, L.A.; RIOS, N.I.; LONGARAY, D.O.; MACIEL, G.C.F.; RAZEIRA, M. 2012. **Desenvolvimento de um Algoritmo em MATLAB para o Cálculo do Periodograma de Lomb-Scargle de Séries de Dados Estratigráficos, com Vetorização e Implementaçaõ em GPU.** Anais do Salão Internacional de Ensino, Pesquisa e Extensão.

FREITAS, S. R. **Métodos Numéricos**. 1ª Ed. Mato Grosso do Sul, 2000.

GEORGE, R., 2007. **Robs algorithm**, Applied Mathematics and Computation, 189, 314–325.

GUTERRES, M. X.; PERALTA, C. B. da L.; TEIXERA, L. G. M. de F. **Aproximação de uma solução para a Equação de Richards-2D pelo Método de Volumes Finitos com o auxílio dos algoritmos de Picard-Krylov**, Águas Subterrâneas, 31(2):89-108. 2013.

MALISKA, C. R. **Transferência de Calor e Mecânica dos Fluidos Computacional**. 2ª Ed. Rio de Janeiro, 2017.

MATTOS, L.E.G.; FERNANDES, M.A.; SANTIAGO, C.D. **Implementação de um código computacional para simular infiltração em solos não saturados**. In: SEMINÁRIO DE INICIAÇÃO CIENTIFICA E TECNOLÓGICA, 8., 2018, Apucarana. Anais Apucarana: UTFPR, 2018.

MCGARRITY S.; Programming Patterns: Maximizing Code Performance by Optimizing Memory Access, MathWorks, 2019, Disponível em: https://nl.mathworks.com/company/newsletters/articles/programming-patterns-maximizing-code-performance-by-optimizing-memory-access.html. Acesso em 22 de out. de 2019.

MENDES, T. R.; STRÖHER, G.R.; ROMEIRO, N. M. L. Avaliação do formato de armazenamento Compressed Sparse Row para resolução de sistemas de equações lineares esparsos. **ENGEVISTA**, V. 19, n.4, p. 1095-1109, Outubro 2017.

PEREIRA, J.M.; SIQUEIRA, M.B.B. 2016. **Linguagem de Programação Julia: Uma Alternativa Open Source e de Alto Desempenho ao MATLAB**, XLIV Congresso Brasileiro de Educação em Engenharia.

RIZWAN, B. 2007. **Introduction to Numerical Analysis Using MATLAB**, Massachusetts, Jones and Bartlett.

ROSE, D.J.; WILLOUGHBY, R.N. 1972. **Sparse Matrix and its Applications**, Plenum Pres, New York.

RUGGIERO, M.A.G.; LOPES, V.L.R. **Cálculo Numérico Aspectos Teóricos e Computacionais**. 2ª Ed. Local, São Paulo: Pearson, 1996.

SAAD, Y. 1994. **A Basic Tool Kit for Sparse Matrix Computations**.

SAAD, Y. 2003. **Iterative Methods for Sparse Linear Systems: Second Edition**, Society for Industrial and Applied Mathematics Philadelphia, PA, USA.

SINAIE, S.; NGUYEN, V.P.; NGUYEN C.T.; BORDAS, S. **Programming the Material Point Method in Julia**. Advances in Engineering Software. v. 105, p. 17-29, mar. 2017.

STERN, J.M. **Esparsidade, Estrutura, Estabilidade e Escalamento em Álgebra Linear Computacional**. 1ª Ed. São Paulo, 1994.

VALIENTE, E.S.P. **Aplicação de Sistemas Lineares e Determinante na Engenharia Civil**. 2015. Dissertação (Mestrado em Matemática em Rede Nacional) – Universidade Federal do Mato Grosso do Sul, Campo Grande, 2015.

VALLE, M. E. Métodos de Jacobi e Gauss-Seidel. **Unicamp**, Campinas. Disponível em: <https://www.ime.unicamp.br/~valle/Teaching/2015/MS211/Aula9.pdf>. Acesso em: 15 mai. 2019.

VALLE, M. E. Normas de Vetores e Matrizes e Condicionamento de uma Matriz. **Unicamp**, Campinas. Disponível em: <https://www.ime.unicamp.br/~valle/Teaching/MS211/Aula05.pdf>. Acesso em: 15 mai. 2019.

VARGA, R.S. 1962. **Matrix iterative analysis**, Prentice Hall, Englewood Cliffs.

VERSTEEG, H.K.; MALALASEKERA. 2007. **An Introduction to Computacional Fluid Dynamics: The Finite Volume Method**, Second Edition, Prentice Hall, England.

WANDRESEN, R. **Métodos Iterativos Para a Solução de Sistemas de Equações Normais**. 1980. Dissertação (Mestrado em Ciências) – Universidade Federal do Paraná, Curitiba, 1980.

World Community Grid. Cds sparse structure, 1995. Disponível em <http://netlib.org/utk/papers/templates/node94.html#SECTION00931400000000000000>. Acesso em: 15 maio 2019.