

Universidade Federal Fluminense

LEONARDO HILÁRIO DA SILVA

Métodos Iterativos para Sistemas Lineares:
Pré-condicionadores, Estruturas de Dados e Outras
Técnicas

VOLTA REDONDA

2021

LEONARDO HILÁRIO DA SILVA

Métodos Iterativos para Sistemas Lineares:
Pré-condicionadores, Estruturas de Dados e Outras Técnicas

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. Ricardo Silveira Sousa

Coorientador:

Prof. D.Sc. Thiago Jordem Pereira

Universidade Federal Fluminense

VOLTA REDONDA

2021

Ficha catalográfica automática - SDC/BINF
Gerada com informações fornecidas pelo autor

S586m Silva, Leonardo Hilário da
Métodos Iterativos para Sistemas Lineares : Pré-
condicionadores, Estruturas de Dados e Outras Técnicas /
Leonardo Hilário da Silva ; Ricardo Silveira Sousa,
orientador ; Thiago Jordem Pereira, coorientador. Santo
Antônio de Pádua, 2021.
162 f. : il.

Dissertação (mestrado)-Universidade Federal Fluminense,
Volta Redonda, 2021.

DOI: <http://dx.doi.org/10.22409/PPG-MCCT.2021.m.13625316773>

1. Métodos Iterativos. 2. Estruturas de Dados. 3. Pré-
condicionadores. 4. Refinamento. 5. Produção intelectual. I.
Sousa, Ricardo Silveira, orientador. II. Pereira, Thiago
Jordem, coorientador. III. Universidade Federal Fluminense.
Instituto do Noroeste Fluminense de Educação Superior. IV.
Título.

CDD -

LEONARDO HILÁRIO DA SILVA

Métodos Iterativos para Sistemas Lineares:
Pré-condicionadores, Estruturas de Dados e Outras Técnicas

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.

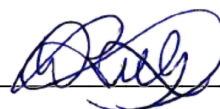
Aprovada por:



Prof. Ricardo Silveira Sousa, D.Sc.
INFES-UFF (Presidente)



Prof. Diego Brandão, Ph.D.
CEFET-RJ



Prof. Wagner Rambaldi Telles, D.Sc.
INFES-UFF

Agradecimentos

Primeiramente agradeço a Deus, por estar ao meu lado a cada etapa do desenvolvimento desse trabalho. Por ter me fortalecido nos momentos mais difíceis em que pensei que não conseguiria. Por não me deixar abalar pelos acontecimentos e pelas dificuldades da vida.

Aos meus pais, por sempre me apoiarem, me incentivarem e por me ensinarem os valores da vida. Agradeço a Deus por ter me concedido a benção de ser seu filho.

Ao meu professor orientador, pela sua paciência e compreensão, pelas suas instruções e direcionamentos, pelas correções e orientação durante todos esses anos.

Ao meu professor coorientador, pelas suas orientações e instruções nesse projeto.

A cada professor que desde a graduação temos convivido, e que, em cada aula, seminário e projeto, contribuíram para a minha formação.

Aos funcionários do INFES, que muitas vezes, em momentos difíceis e complicados desse período, me ajudavam com nossas conversas e brincadeiras.

Ao corpo administrativo do MCCT pelas orientações e auxílios sempre que foi necessário.

Aos meus amigos da escola, da graduação, do programa e da igreja, pelas novas e velhas amizades formadas e que jamais serão esquecidas. Pelas trocas de ideias e momentos preciosos.

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

Resumo

A resolução de sistemas lineares esparsos é de grande interesse em diversas áreas da ciência. Logo, é natural que vários métodos tenham sido desenvolvidos a fim de facilitar a realização dessa tarefa. Para resolver sistemas lineares esparsos os métodos iterativos podem ser mais eficientes que os métodos diretos, como é o caso do Método do Resíduo Mínimo Generalizado (GMRES). Este método tem a vantagem sobre outros métodos iterativos, pois converge para sistemas lineares em que as matrizes não são definidas positivas ou simétricas. Uma maneira eficiente de implementar um método de resolução de sistemas lineares de grande porte e esparsos é armazenar apenas os elementos não nulos utilizando estrutura de dados apropriada, para que possa economizar tempo e memória. Assim, neste trabalho foram pesquisados e implementados os métodos Jacobi, Gauss-Seidel, Gradiente Conjugado e o GMRES reinicializado com a estrutura de dados *Compressed Sparse Row* (CSR), e o GMRES do software Octave para fins comparativos. Foi realizado também um experimento com uma técnica para soma de vetores esparsos. Incorporou-se diferentes pré-condicionadores, com algumas variações, para melhorar a qualidade da solução e tornar a convergência mais rápida. Por último, utilizou-se também a técnica de refinamento de soluções a fim de obter solução mais próximas da exata. Os resultados obtidos mostram as vantagens do uso de uma estrutura de dados adequada, assim como a importância do uso de pré-condicionadores para sistemas mal condicionados. Além disso, a técnica de refinamento de solução mostrou ser promissora para obtenção de melhores soluções.

Abstract

The resolution of sparse linear systems is of great interest in several areas of science. Therefore, it is natural that divers methods have been developed in order to facilitate this task. To solve sparse linear systems, iterative methods can be more efficient than direct methods, such as the Generalized Minimum Residue Method (GMRES). This method has the advantage over other iterative methods, as it converges to linear systems in which the matrices are not positively defined or symmetrical. An efficient way to implement a method of solving large and sparse linear systems is to store only the non-null elements using appropriate data structure, so you can save time and memory. Thus, in this work the Jacobi, Gauss-Seidel, Conjugated Gradient and the GMRES reset with the Compressed Sparse Row (CSR) data structure and the GMRES of the Octave software for comparative purposes were researched and implemented. An experiment was also carried out with a technique for the sum of sparse vectors. Different pre-conditioners were incorporated, with some variations, to improve the quality of the solution and make the convergence faster. Finally, the solution refinement technique was also used in order to obtain a solution closer to the exact one. The results obtained show the advantages of using an adequate data structure, as well as the importance of using pre-conditioners for poorly conditioned systems. Furthermore, the solution refinement technique showed to be promising for obtaining better solutions.

Palavras-chave

1. Métodos Iterativos.
2. GMRES(m).
3. Estrutura de Dados.
4. Pré-condicionadores.
5. Refinamento.

Sumário

Lista de Figuras	10
Lista de Tabelas	11
1. Introdução	16
1.1. Justificativa	16
1.2. Objetivos	18
1.3. Organização do Texto	18
2. Sistemas Lineares e Métodos Iterativos	20
2.1. Sistemas Lineares: Importância e Aplicações	20
2.1.1. Métodos Iterativos e Diretos	21
2.1.2. Erros Comparativos e Condicionamento	21
2.2. Métodos Iterativos	25
2.2.1. Método de Jacobi	26
2.2.2. Método de Gauss-Seidel	29
2.2.3. Método do Gradiente Conjugado	31
2.2.4. Método do Resíduo Mínimo Generalizado Reinicializado (GMRES(m))	35
3. Esparsidade e Estruturas de Dados	40
3.1. Esparsidade e Densidade	40
3.2. Estruturas de Dados Compactas	42
3.2.1. Linhas Esparsas Compactadas (<i>Compressed Sparse Row</i> – CSR)	42
3.2.2. Colunas Esparsas Compactadas (<i>Compressed Sparse Column</i> – CSC)	44
3.3. Manipulação de Vetores Esparsos	45
3.3.1. Vetores Esparsos no GMRES(m)	48
3.3.2. Complexidade de Vetores Esparsos no GMRES(m)	52
4. Pré-condicionadores e Refinamento de Solução	55
4.1. Pré-condicionamento	55
4.2. Pré-condicionadores	58
4.2.1. Pré-condicionador Lumped Modificado	58
4.2.2. Pré-condicionador Escala Simétrica	59
4.2.2.1. Pré-condicionador Escala Simétrica – Variante 1	61
4.2.2.2. Pré-condicionador Escala Simétrica – Variante 2	62
4.2.3. Pré-condicionador ILU(0)	63
4.2.4. Pré-condicionador Diagonal/Escala Simétrica	67
4.2.5. Pré-condicionador Diagonal/ILU(0)	71
4.2.6. Pré-condicionador P	72
4.3. Refinamento	74
5. Problemas Tratados	79

5.1.	Problemas dos Repositórios	79
5.2.	Gerador de Problemas Aleatórios	80
6.	Resultados	84
6.1.	Especificações e Parâmetros	84
6.2.	Conversão de Arquivos de Entrada	85
6.3.	Arquivos de Saída	86
6.4.	Validação dos Arquivos de Entrada	88
6.5.	Resultados	89
6.5.1.	Jacobi com Estrutura de Dados Indexada e CSR	90
6.5.1.1.	Problemas dos Repositórios	90
6.5.1.2.	Problemas do Gerador Aleatório	91
6.5.2.	Gauss-Seidel com Estrutura de Dados Indexada e CSR	93
6.5.2.1.	Problemas dos Repositórios	93
6.5.2.2.	Problemas do Gerador Aleatório	94
6.5.3.	Gradiente Conjugado com Estrutura de Dados Indexada e CSR	96
6.5.3.1.	Problemas dos Repositórios	96
6.5.3.2.	Problemas do Gerador Aleatório	97
6.5.4.	GMRES(m) com Estrutura de Dados CSR	99
6.5.4.1.	Problemas dos Repositórios	99
6.5.4.2.	Problemas do Gerador Aleatório	100
6.5.5.	GMRES(m) com Soma de Vetores Esparsos	103
6.5.5.1.	Problemas dos Repositórios	104
6.5.5.2.	Problemas do Gerador Aleatório	105
6.5.6.	GMRES(m) com Estrutura de Dados CSR e Pré-condicionador Lumped Modificado	107
6.5.6.1.	Problemas dos Repositórios	107
6.5.6.2.	Problemas do Gerador Aleatório	109
6.5.7.	GMRES(m) com Estrutura de Dados CSR e Pré-condicionador Escala Simétrica	112
6.5.7.1.	Problemas dos Repositórios	112
6.5.7.2.	Problemas do Gerador Aleatório	114
6.5.8.	GMRES(m) com Estrutura de Dados CSR e Pré-condicionador Escala Simétrica – Variante 1	117
6.5.8.1.	Problemas dos Repositórios	117
6.5.8.2.	Problemas do Gerador Aleatório	118
6.5.9.	GMRES(m) com Estrutura de Dados CSR e Pré-condicionador Escala Simétrica – Variante 2	121
6.5.9.1.	Problemas dos Repositórios	121
6.5.9.2.	Problemas do Gerador Aleatório	123
6.5.10.	GMRES(m) com Estrutura de Dados CSR e Pré-condicionador ILU(0)	126
6.5.10.1.	Problemas dos Repositórios	126
6.5.10.2.	Problemas do Gerador Aleatório	127

6.5.11.	GMRES(m) com Estrutura de Dados CSR e Pré-condicionador Diagonal/Escala Simétrica	131
6.5.11.1.	Problemas dos Repositórios	131
6.5.11.2.	Problemas do Gerador Aleatório	132
6.5.12.	GMRES(m) com Estrutura de Dados CSR e Pré-condicionador Diagonal/ILU(0)	135
6.5.12.1.	Problemas dos Repositórios	135
6.5.12.2.	Problemas do Gerador Aleatório	137
6.5.13.	GMRES(m) com Estrutura de Dados CSR e Pré-condicionador P	139
6.5.13.1.	Problemas dos Repositórios	139
6.5.13.2.	Problemas do Gerador Aleatório	141
6.5.14.	Refinamento de Solução Utilizando GMRES(m)	144
6.5.14.1.	Problemas dos Repositórios	144
6.5.14.2.	Problemas do Gerador Aleatório	145
6.5.15.	Refinamento de Solução Utilizando ILU(0)	145
6.5.15.1.	Problemas dos Repositórios	145
6.5.15.2.	Problemas do Gerador Aleatório	147
6.5.16.	GMRES do Octave	148
6.5.16.1.	Problemas dos Repositórios	148
6.5.16.2.	Problemas do Gerador Aleatório	150
6.6.	Análise de Resultados	150
7.	Conclusões e Trabalhos Futuros	156
	Referências	159

Lista de Figuras

2.1.	Problema bem condicionado	23
2.2.	Problema mal condicionado	23
2.3.	Problema singular	24
3.1.	Representação da estrutura CSR	43
3.2.	Representação da estrutura CSC	44
3.3.	Soma de dois vetores esparsos	46
6.1.	Arquivo de entrada	85
6.2.	Arquivo de entrada após conversão	86
6.3.	Arquivo de saída do GMRES(m)	87
6.4.	Arquivo de saída do Gradiente Conjugado	88
6.5.	Média de iterações entre as estruturas indexada e CSR	151
6.6.	Tempo médio entre as estruturas indexada e CSR	152
6.7.	Média de iterações entre os métodos sem pré-condicionamento	153
6.8.	Tempo médio entre os métodos sem pré-condicionamento	154
6.9.	Média de iterações com e sem pré-condicionamento para o método GMRES(m)	154
6.10.	Tempo médio com e sem pré-condicionamento para o método GMRES(m)	155

Lista de Tabelas

Tabela 1 - Propriedades dos problemas dos repositórios	79
Tabela 2 - Jacobi com estrutura indexada e CSR – Repositórios.	90
Tabela 3 - Jacobi com estrutura indexada e CSR – Gerador aleatório – Matriz cheia.	91
Tabela 4 - Jacobi com estrutura indexada e CSR – Gerador aleatório – Matriz triangular.	92
Tabela 5 - Jacobi com estrutura indexada e CSR – Gerador aleatório – Matriz tridiagonal.	92
Tabela 6 - Gauss-Seidel com estrutura indexada e CSR - Repositórios.	93
Tabela 7 - Gauss-Seidel com estrutura indexada e CSR – Gerador aleatório – Matriz cheia.	94
Tabela 8 - Gauss-Seidel com estrutura indexada e CSR – Gerador aleatório – Matriz triangular.	95
Tabela 9 - Gauss-Seidel com estrutura indexada e CSR – Gerador aleatório – Matriz tridiagonal.	95
Tabela 10 - Gradiente Conjugado com estrutura indexada e CSR - Repositórios.	96
Tabela 11 - Gradiente Conjugado com estrutura indexada e CSR – Gerador aleatório – Matriz cheia.	97
Tabela 12 - Gradiente Conjugado com estrutura indexada e CSR – Gerador aleatório – Matriz triangular.	98
Tabela 13 - Gradiente Conjugado com estrutura indexada e CSR – Gerador aleatório – Matriz tridiagonal.	98
Tabela 14 - GMRES(m) com estrutura CSR - Repositórios.	99
Tabela 15 - GMRES(m) com estrutura CSR - Repositórios.	100
Tabela 16 – GMRES(m) com estrutura CSR – Gerador aleatório – Matriz cheia.	101
Tabela 17 – GMRES(m) com estrutura CSR – Gerador aleatório – Matriz triangular.	101
Tabela 18 – GMRES(m) com estrutura CSR – Gerador aleatório – Matriz tridiagonal.	102
Tabela 19 – GMRES(m) com estrutura CSR – Gerador aleatório – Matriz cheia.	102
Tabela 20 – GMRES(m) com estrutura CSR – Gerador aleatório – Matriz triangular.	103
Tabela 21 – GMRES(m) com estrutura CSR – Gerador aleatório – Matriz tridiagonal.	103
Tabela 22 – GMRES(m) com soma de vetores esparsos – Repositórios.	104
Tabela 23 – GMRES(m) com soma de vetores esparsos - Repositórios.	104
Tabela 24 – GMRES(m) com soma de vetores esparsos – Gerador aleatório – Matriz cheia.	105
Tabela 25 – GMRES(m) com soma de vetores esparsos – Gerador aleatório – Matriz triangular.	105
Tabela 26 – GMRES(m) com soma de vetores esparsos – Gerador aleatório – Matriz tridiagonal.	105
Tabela 27 – GMRES(m) com soma de vetores esparsos – Gerador aleatório – Matriz cheia.	106

Tabela 28 – GMRES(m) com soma de vetores esparsos – Gerador aleatório – Matriz triangular.	106
Tabela 29 – GMRES(m) com soma de vetores esparsos – Gerador aleatório – Matriz tridiagonal.	106
Tabela 30 – GMRES(m) com estrutura CSR e pré-condicionador Lumped – Repositórios.	107
Tabela 31 – GMRES(m) com estrutura CSR e pré-condicionador Lumped – Repositórios.	108
Tabela 32 – GMRES(m) com estrutura CSR e pré-condicionador Lumped – Gerador aleatório – Matriz cheia.	109
Tabela 33 – GMRES(m) com estrutura CSR e pré-condicionador Lumped – Gerador aleatório – Matriz triangular.	110
Tabela 34 – GMRES(m) com estrutura CSR e pré-condicionador Lumped – Gerador aleatório – Matriz tridiagonal.	110
Tabela 35 – GMRES(m) com estrutura CSR e pré-condicionador Lumped – Gerador aleatório – Matriz cheia.	111
Tabela 36 – GMRES(m) com estrutura CSR e pré-condicionador Lumped – Gerador aleatório – Matriz triangular.	111
Tabela 37 – GMRES(m) com estrutura CSR e pré-condicionador Lumped – Gerador aleatório – Matriz tridiagonal.	111
Tabela 38 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica – Repositórios.	112
Tabela 39 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica – Repositórios.	113
Tabela 40 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica – Gerador aleatório – Matriz cheia.	114
Tabela 41 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica – Gerador aleatório – Matriz triangular.	114
Tabela 42 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica – Gerador aleatório – Matriz tridiagonal.	115
Tabela 43 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica – Gerador aleatório – Matriz cheia.	115
Tabela 44 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica – Gerador aleatório – Matriz triangular.	116
Tabela 45 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica – Gerador aleatório – Matriz tridiagonal.	116
Tabela 46 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 1 - Repositórios.	117
Tabela 47 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 1 - Repositórios.	118
Tabela 48 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 1 – Gerador aleatório – Matriz cheia.	118
Tabela 49 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 1 – Gerador aleatório – Matriz triangular.	119

Tabela 50 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 1 – Gerador aleatório – Matriz tridiagonal.	119
Tabela 51 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variação 1 – Gerador aleatório – Matriz cheia.	120
Tabela 52 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variação 1 – Gerador aleatório – Matriz triangular.	120
Tabela 53 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variação 1 – Gerador aleatório – Matriz tridiagonal.	121
Tabela 54 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 2 - Repositórios.	122
Tabela 55 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 2 - Repositórios.	122
Tabela 56 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 2 – Gerador aleatório – Matriz cheia.	123
Tabela 57 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 2 – Gerador aleatório – Matriz triangular.	123
Tabela 58 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 2 – Gerador aleatório – Matriz tridiagonal.	124
Tabela 59 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 2 – Gerador aleatório – Matriz cheia.	124
Tabela 60 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 2 – Gerador aleatório – Matriz triangular.	125
Tabela 61 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 2 – Gerador aleatório – Matriz tridiagonal.	125
Tabela 62 – GMRES(m) com estrutura CSR e pré-condicionador ILU(0) – Repositórios.	126
Tabela 63 – GMRES(m) com estrutura CSR e pré-condicionador ILU(0) – Repositórios.	127
Tabela 64 – GMRES(m) com estrutura CSR e pré-condicionador ILU(0) – Gerador aleatório – Matriz cheia.	128
Tabela 65 – GMRES(m) com estrutura CSR e pré-condicionador ILU(0) – Gerador aleatório – Matriz triangular.	128
Tabela 66 – GMRES(m) com estrutura CSR e pré-condicionador ILU(0) – Gerador aleatório – Matriz tridiagonal.	128
Tabela 67 – GMRES(m) com estrutura CSR e pré-condicionador ILU(0) – Gerador aleatório – Matriz cheia.	129
Tabela 68 – GMRES(m) com estrutura CSR e pré-condicionador ILU(0) – Gerador aleatório – Matriz triangular.	129
Tabela 69 – GMRES(m) com estrutura CSR e pré-condicionador ILU(0) – Gerador aleatório – Matriz tridiagonal.	130
Tabela 70 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/Escala Simétrica - Repositórios.	131
Tabela 71 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/Escala Simétrica - Repositórios.	132

Tabela 72 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/Escala Simétrica – Gerador aleatório – Matriz cheia.	132
Tabela 73 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/Escala Simétrica – Gerador aleatório – Matriz triangular.	133
Tabela 74 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/Escala Simétrica – Gerador aleatório – Matriz tridiaogonal.	133
Tabela 75 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/Escala Simétrica – Gerador aleatório – Matriz cheia.	134
Tabela 76 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/Escala Simétrica – Gerador aleatório – Matriz triangular.	134
Tabela 77 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/Escala Simétrica – Gerador aleatório – Matriz tridiagonal.	135
Tabela 78 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/ILU(0) – Repositórios.	136
Tabela 79 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/ILU(0) – Repositórios.	136
Tabela 80 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/ILU(0) – Gerador aleatório – Matriz cheia.	137
Tabela 81 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/ILU(0) – Gerador aleatório – Matriz triangular.	137
Tabela 82 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/ILU(0) – Gerador aleatório – Matriz tridiagonal.	137
Tabela 83 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/ILU(0) – Gerador aleatório – Matriz cheia.	138
Tabela 84 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/ILU(0) – Gerador aleatório – Matriz triangular.	138
Tabela 85 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/ILU(0) – Gerador aleatório – Matriz tridiagonal.	139
Tabela 86 – GMRES(m) com estrutura CSR e pré-condicionador P - Repositórios.	140
Tabela 87 – GMRES(m) com estrutura CSR e pré-condicionador P - Repositórios.	140
Tabela 88 – GMRES(m) com estrutura CSR e pré-condicionador P – Gerador aleatório – Matriz cheia.	141
Tabela 89 – GMRES(m) com estrutura CSR e pré-condicionador P – Gerador aleatório – Matriz triangular.	141
Tabela 90 – GMRES(m) com estrutura CSR e pré-condicionador P – Gerador aleatório – Matriz tridiagonal.	141
Tabela 91 – GMRES(m) com estrutura CSR e pré-condicionador P – Gerador aleatório – Matriz cheia.	142
Tabela 92 – GMRES(m) com estrutura CSR e pré-condicionador P – Gerador aleatório – Matriz triangular.	143
Tabela 93 – GMRES(m) com estrutura CSR e pré-condicionador P – Gerador aleatório – Matriz tridiagonal.	143
Tabela 94 – Refinamento de solução utilizando o método GMRES(m) - Repositórios.	144

Tabela 95 – Refinamento de solução utilizando o método GMRES(m) – Gerador aleatório.	145
Tabela 96 – Refinamento de solução utilizando a fatoração ILU(0) - Repositórios.	146
Tabela 97 – Refinamento de solução utilizando a fatoração ILU(0) – Gerador aleatório.	147
Tabela 98 – GMRES do Octave - Repositórios.	149
Tabela 99 – GMRES do Octave – Gerador aleatório.	150

Capítulo 1

Introdução

Em diversas áreas, a necessidade de se resolver problemas que envolvem, em algum momento, sistemas lineares, tem ganhado alto destaque. Ao trabalhar com problemas reais, é comum que sua matriz principal seja de grande porte, como também possuam grande parte de seus elementos nulos.

Assim, métodos que consigam resolver sistemas lineares de grande porte têm sido alvos de estudos constantes. É possível separá-los em dois grupos: dos diretos para aqueles que obtêm a solução exata do problema desconsiderando os erros na aritmética do ponto flutuante, mas que não é adequado para problemas maiores e, um segundo grupo, dos iterativos, com aqueles que alcançam uma aproximação da solução e lidam bem com esses problemas de grande porte.

Para obter melhores resultados é feito o uso de técnicas (pré-condicionamento) e estratégias (refinamento de solução). E ainda, com o uso de uma estrutura de dados adequada é possível diminuir muito o tempo de resolução, ou até mesmo fazer determinado método resolver um problema que antes não era possível devido ao problema da falta de memória.

1.1 Justificativas

Destaca-se que em muitos problemas das mais diversas áreas, como na matemática, física, engenharia, computação, etc., há a necessidade de, em algum momento resolver um subproblema que consiste exatamente de um ou mais sistemas lineares.

Um sistema linear pode ser definido através da seguinte equação:

$$\mathbf{Ax} = \mathbf{b} \tag{1.1}$$

em que $\mathbf{A}$ representa a matriz principal do problema contendo cada um dos coeficientes do sistema de dimensão $n \times n$, $\mathbf{b}$ é o vetor independente e $\mathbf{x}$ é o vetor solução.

Desde muito tempo sistemas lineares têm sido alvos de estudos, os quais necessitam de métodos para resolvê-los, buscando a melhor solução possível. Atualmente, pelo fato de sistemas lineares estarem presentes em muitos problemas reais, desenvolve-se também a preocupação com o tempo necessário para resolvê-los.

Em Sousa e Silva (2016) foi feito um estudo sobre o método iterativo do Resíduo Mínimo Generalizado (GMRES) para resolução de sistemas lineares apresentado por Saad e Schultz (1986), mais especificamente na versão reinicializada, chamada GMRES(m). Este método tem ganhado destaque em pesquisas de diversas áreas e se tornou alvo de vários estudos, os quais buscam melhorar sua eficiência com uso de diversas técnicas e estratégias. Silva et. al. (2017) analisou também o efeito que uma estrutura de dados adequada provocava no tempo computacional e como o uso de pré-condicionadores influenciaria o resultado final.

A estrutura de dados utilizada no trabalho de Silva et. al. (2017), foi a *Compressed Sparse Row* – CSR que permite armazenar somente os elementos não-nulos da matriz. A estrutura foi usada no armazenamento da matriz principal A do sistema linear já que, em problemas reais, ela apresenta como característica principal a sua alta esparsidade. Os resultados obtidos revelaram um grande desempenho desta estrutura ao ser utilizada no método GMRES(m).

Ressalta-se que incorporar pré-condicionadores em métodos iterativos pode melhorar o desempenho e ainda conseguir a convergência de problemas, que sem o uso desta técnica não é possível obter a solução.

Vale lembrar que um bom pré-condicionador deve ser aplicado num sistema de tal forma que se tenha um novo sistema onde a nova matriz principal A seja próxima da original. Nos trabalhos de Silva e Sousa (2018) e Silva e Sousa (2019) foram testados quatro pré-condicionadores: dois pré-condicionadores simples tanto na construção como na aplicação (Diagonal e Lumped), um pré-condicionador com construção simples, mas aplicação diferente dos dois anteriores (Escala Simétrica), e um pré-condicionador que pede mais esforço para ser construído e que na aplicação surgem dois novos sistemas simples para se resolver (Fatoração LU Incompleta sem preenchimento, ou apenas $ILU(0)$). Cada um possui suas vantagens e desvantagens, dependendo principalmente das características do problema tratado. Vale destacar ainda que três destes pré-condicionadores que tiveram um melhor desempenho (levando em conta a qualidade da solução, o tempo de resolução e a quantidade de ocorrências de convergência): Diagonal (Silva e Sousa, 2018), Escala Simétrica e $ILU(0)$ (Silva e Sousa, 2019).

1.2 Objetivos

Este trabalho tem como objetivo geral a implementação e observação da estrutura de dados CSR para alguns métodos iterativos, assim como a pesquisa de novas técnicas e pré-condicionadores que podem ser aplicados ao GMRES(m) a fim de melhorar o desempenho dele. Assim, espera-se que este texto contribua para o estudo e pesquisa de métodos iterativos, estrutura de dados adequada e técnicas de pré-condicionamento e refinamento para sistemas lineares.

Como objetivos específicos, pode-se citar:

- Implementar os métodos iterativos Jacobi, Gauss-Seidel e Gradiente Conjugado, e comparar suas versões com estrutura de dados indexada e CSR;
- Implementar o método GMRES(m) com a estrutura de dados CSR, e compará-lo com os demais métodos utilizando a mesma estrutura;
- Estudar e desenvolver um gerador aleatório de sistemas lineares;
- Estudar e implementar a técnica de soma de vetores esparsos no método GMRES(m);
- Estudar, propor e implementar variações e novas combinações de pré-condicionadores para o método GMRES(m);
- Implementar a técnica de refinamento de solução e aplicá-la em diversas situações de melhora de resultados;
- Verificar o desempenho dos métodos iterativos implementados com o GMRES do Octave.

1.3 Organização do Texto

O presente trabalho encontra-se dividido em sete capítulos com os seguintes conteúdos:

No Capítulo 2, inicialmente é feita uma breve introdução aos métodos iterativos e sobre erros computacionais e condicionamento seguida por um estudo rápido sobre os quatro métodos iterativos usados no presente trabalho: Jacobi, Gauss-Seidel, Gradiente Conjugado e GMRES(m).

Por outro lado, no Capítulo 3 tem-se a discussão acerca das estruturas de dados, das quais duas (CSR e Vetor Esperso) serão abordadas com maior detalhe, apontando as considerações de implementações necessárias para ambas.

Já o Capítulo 4 traz, principalmente, os pré-condicionadores utilizados no presente trabalho, onde alguns se tratam de novas abordagens e adaptações dos utilizados por Silva (2017). Também será visto aqui uma nova estratégia ao utilizar dois pré-condicionadores consecutivos para resolver um determinado problema. Este capítulo também tratará, ainda, sobre a técnica de refinamento de solução.

No Capítulo 5 são apresentados os problemas utilizados que foram retirados de repositórios online. Também se aborda um gerador de problemas aleatório que foi implementado.

Na sequência, o Capítulo 6 é dedicado para a apresentação dos resultados obtidos, assim como uma discussão sobre eles.

Por último, tem-se, no Capítulo 7, as conclusões finais, assim como propostas apontadas para trabalhos futuros.

Capítulo 2

Sistemas Lineares e Métodos Iterativos

Como mencionado anteriormente, métodos para resolução de sistema lineares podem ser divididos em dois grupos distintos: métodos diretos e métodos iterativos.

Além disso, quando se trata de resolver sistemas lineares (principalmente quando se lida com métodos iterativos), entender o que são os erros e porque eles ocorrem também se torna necessário para uma boa análise dos dados obtidos.

Assim, quatro métodos iterativos diferentes serão estudados aqui: Jacobi e Gauss-Seidel, dos métodos iterativos estacionários; Gradiente Conjugado e GMRES(m), dos métodos iterativos não-estacionários.

2.1 Sistemas Lineares: Importância e Aplicações

Em diversas áreas, como a engenharia, economia, administração, física, química e eletrônica, por exemplo, ao estudar determinado fenômeno, é necessário resolver determinados sistemas lineares.

Durante as elaborações de projetos de estruturas com vigas metálicas (Pescador, Possamai e Possamai, 2011), assim como estudos em sistemas de energia envolvendo, por exemplo, o fluxo de potência e a estabilidade transitória (Paz, 2012), necessitam da aplicação de determinado método numérico para resolução de um sistema linear.

Pescador, Possamai e Possamai (2011) mostraram, ainda, que em circuitos elétricos, quando deseja-se encontrar as correntes elétricas presente no mesmo e, na química, ao se tentar balancear equações, como a que representa a reação da combustão da amônia em oxigênio gerando nitrogênio e água, também trazem a necessidade de se resolver sistemas lineares.

Porém, para esses problemas reais, tem-se como características, dimensões de grande porte e sistemas mais difíceis de resolver, tornando inviável que seja usado simplesmente o esforço humano para resolvê-los.

Com essas dificuldades, aliado ao avanço da tecnologia (principalmente quando se trata da capacidade de processamento dos computadores), foram surgindo métodos para auxiliar na resolução dos sistemas lineares. Dentre esses métodos tem aqueles mais simples como a Eliminação de Gauss e a Decomposição LU e, até mesmo, outros mais complexos como o Gradiente Conjugado e, alvo principal deste trabalho, o GMRES(m).

2.1.1 Métodos Iterativos e Diretos

Dentre os métodos de resolução para sistemas lineares é possível separá-los em duas classes distintas: os métodos diretos e os métodos iterativos.

Os métodos diretos são aqueles que realizam operações na matriz de coeficientes do sistema, ou seja, eles modificam diretamente a matriz principal do problema original. E o fato deles realizarem essa modificação pode interferir na sua capacidade de resolver os problemas, principalmente quando a dimensão do mesmo cresce muito e/ou quando a matriz $\mathbf{A}$ se torna muito esparsa (grande parte de elementos são nulos), uma vez que podem gerar preenchimento, dificultando a resolução (Silva, 2017). Por último, esses métodos possuem garantia de convergência, ou seja, garantia de encontrar a solução mais próxima da exata. Exemplos clássicos de métodos diretos são: Eliminação de Gauss e suas variações e métodos de fatoração (como LU e Cholesky).

Por outro lado, os métodos iterativos, não modificam a matriz principal do problema, mas sim utilizam a mesma somente em operações de multiplicações de matriz por vetor. Com isso, eles não perdem tanta eficiência ao tratar sistemas grandes e esparsos. Porém, em contra partida, eles perdem a garantia de convergência que os métodos diretos possuem e passam a depender de determinadas condições para obter a uma boa aproximação da solução. Dentre os métodos iterativos pode-se destacar: Gauss-Seidel, Jacobi, Gradiente Conjugado e o GMRES(m).

2.1.2 Erros Comparativos e Condicionamento

Antes de apresentar os métodos iterativos é preciso fazer pequenas considerações acerca de fatores importantes quando se trata da resolução de sistemas lineares: erros comparativos e condicionamento.

De acordo com Bortoli et al. (2003), toda solução de sistemas lineares deve ser considerada como uma aproximação da ideal, por causa de erros de arredondamentos ocorridos principalmente em métodos numéricos. Uma forma de verificar a dimensão desse erro é através do resíduo. Desde que cada componente do mesmo seja suficientemente pequeno (para isso é definido uma tolerância), é possível dizer que tal solução é uma boa aproximação da exata. O resíduo pode ser obtido da seguinte maneira:

$$\mathbf{r} = \mathbf{A}\bar{\mathbf{x}} - \mathbf{b} \quad (2.1)$$

Porém, o resíduo não consegue dimensionar esse erro de maneira adequada para todos os sistemas. Tais sistemas são conhecidos como mal condicionados, os quais quando sujeitos a pequenas alterações resultam em mudanças consideráveis na solução. Bortoli et al. (2003) mostraram isso por meio de um exemplo.

Considere o seguinte sistema:

$$\begin{cases} x_1 + x_2 = 2 \\ 1,01x_1 + x_2 = 2,01 \end{cases}$$

Em que sua solução exata é: $\mathbf{x}^T = [1 \ 1]$. Porém, os autores mostram que, com uma solução obtida com uma diferença significativa na segunda componente, $x_1 = 0$ e $x_2 = 2,005$, ainda retornará um resíduo que aparentemente seria uma boa solução, $\mathbf{r}^T = [-0,005 \ 0,005]$, porém, obviamente está muito distante da solução exata.

Os autores ainda exemplificam outra situação possível: quando ocorre uma mudança nos dados do problema. Suponha que determinados valores do sistema anterior sofram uma pequena alteração. Para este exemplo, de acordo com Bortoli et al. (2003), tem-se:

$$\begin{cases} x_1 + x_2 = 2 \\ 1,0001x_1 + x_2 = 2,007 \end{cases}$$

Note que, com apenas essa ligeira modificação de coeficientes, é possível obter outros valores de x , $x_1 = 100$ e $x_2 = -98$, no qual o resíduo fique pequeno, $\mathbf{r}^T = [0 \ -0,003]$.

Diante disso, fica agora a necessidade de identificar quais problemas são mal condicionados. Para problemas bidimensionais (duas variáveis e duas equações) é fácil. Basta fazer um gráfico plotando as retas de cada equação do sistema e analisar a disposição delas. Na Figura 2.1 é apresentada a primeira situação que pode ser encontrada. Quando as retas das equações se encontram em um único ponto esse gráfico representará um problema bem condicionado:

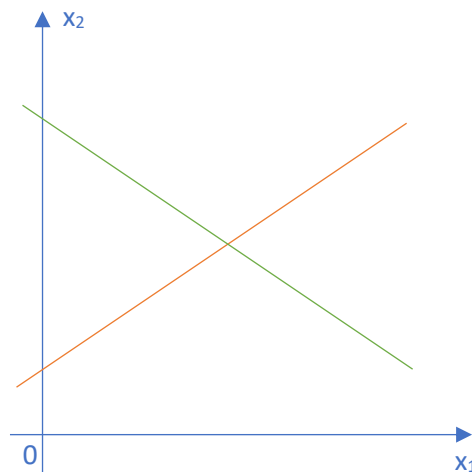


Figura 2.1: Problema bem condicionado.

Na Figura 2.2 é apresentada outra situação possível. Quando há mais de um ponto de interseção demonstra um problema mal condicionado:

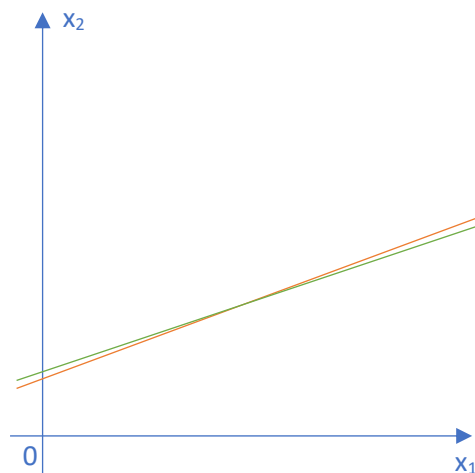


Figura 2.2: Problema mal condicionado.

Por último veja a Figura 2.3 para a última situação apresentada por Bortolli et al. (2003). Neste caso as duas retas nunca se cruzam. Quando isso ocorre é dado o nome de sistema singular:

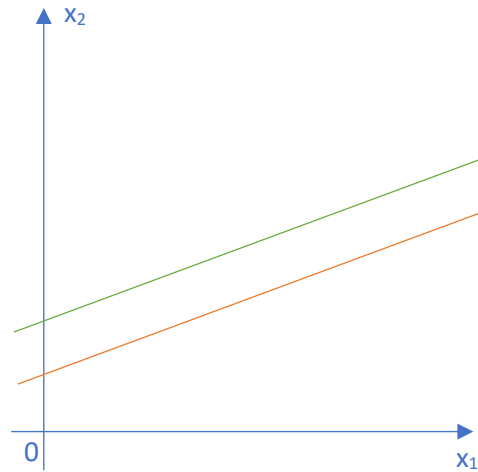


Figura 2.3: Sistema singular.

Porém, conforme a ordem aumenta já não é possível visualizar facilmente se o problema é mal condicionado com um gráfico. Assim, Bortoli et al. (2003) sugeriram outra forma de analisar a condição de matrizes maiores: Determinante Normalizado da Matriz de Coeficientes.

Para tal, normaliza-se a matriz A pela raiz da soma dos quadrados dos elementos da linha correspondente. Depois basta calcular a determinante dessa matriz normalizada. Em outros termos:

$$\text{norm}|A| = \begin{vmatrix} \frac{a_{11}}{\alpha_1} & \frac{a_{12}}{\alpha_1} & \dots & \frac{a_{1n}}{\alpha_1} \\ \frac{a_{21}}{\alpha_2} & \frac{a_{22}}{\alpha_2} & \dots & \frac{a_{2n}}{\alpha_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{a_{n1}}{\alpha_n} & \frac{a_{n2}}{\alpha_n} & \dots & \frac{a_{nn}}{\alpha_n} \end{vmatrix} \quad (2.2)$$

Ou seja,

$$\text{norm}|A| = \frac{|A|}{\alpha_1 \alpha_2 \dots \alpha_n} \quad (2.3)$$

onde

$$\alpha_i = \sqrt{a_{i1}^2 + a_{i2}^2 + \dots + a_{in}^2} \quad (2.4)$$

De maneira geral, quando esse número for pequeno, em relação a unidade (Bortoli et al., 2003), esse sistema é dito mal condicionado.

2.2 Métodos Iterativos

Como citado anteriormente, métodos iterativos são indicados para sistemas de grande porte e esparsos, já que não alteram diretamente a matriz dos coeficientes, e, portanto, não haverá preenchimentos de posições nulas. Por outro lado, esses métodos possuem restrições quanto a garantia de convergência, sendo que cada método depende de determinadas propriedades da matriz principal.

Purcina (2010) define os métodos iterativos como aqueles que buscam uma solução para o problema a partir de uma solução inicial aproximada. Esses métodos convertem o problema original representado pela Equação (1.1) para um processo iterativo

$$\mathbf{x}^{k+1} = \mathbf{C}\mathbf{x}^k + \mathbf{g} \quad (2.5)$$

onde $\mathbf{x}^{k+1}$ é a nova solução a ser encontrada, $\mathbf{C}$ é a matriz de iteração, $\mathbf{x}^k$ é a solução atual e $\mathbf{g}$ o vetor de iteração (Silva, 2017).

Os métodos iterativos podem ainda ser divididos em duas categorias: os estacionários e os não-estacionários. Os estacionários são aqueles que realizam operações matriciais constantes sobre $\mathbf{x}^k$ a fim de encontrar $\mathbf{x}^{k+1}$. Os métodos de Jacobi, Gauss-Seidel e do Relaxamento Sucessivo (do inglês *Successive Over Relaxation* - SOR) são alguns dos exemplos mais conhecidos desta categoria.

Já nos métodos iterativos não-estacionários, as operações matriciais não são constantes. Essa categoria se baseia na otimização de problemas. Para isso, ao invés de resolver diretamente o problema em sua forma original (1.1), ele o trata como um problema de minimizar a norma do resíduo com a solução atual, ou seja, $\min \|\mathbf{b} - \mathbf{A}\mathbf{x}^k\|$. Os métodos dessa classe são relativamente recentes (principalmente se comparados aos estacionários) tendendo a serem mais complexos, porém se destacando por serem altamente eficientes. Para mais detalhes

sobre métodos iterativos estacionários e não-estacionários consulte Saad (2003) e Purcina (2010).

Assim, nos próximos tópicos serão apresentados os quatros métodos iterativos utilizados no presente trabalho, sendo o último deles o foco principal.

2.2.1 Método de Jacobi

Segundo Asano e Colli (2009), o método de Jacobi é um método iterativo que possui a vantagem de ser mais simples de implementar que o método de Escalonamento (um dos métodos diretos) além de apresentar menos erros por arredondamento. Os autores destacam também que uma desvantagem desse método é que o mesmo não é adequado para todos tipos de problemas.

Dado um sistema linear do tipo (1.1) é possível escrevê-lo também da seguinte forma generalizada

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n = b_n \end{cases} \quad (2.6)$$

onde $a_{ii} \neq 0$ para $i = 1, 2, \dots, n$ e lembrando que n é a dimensão do problema. A partir dessa forma geral, isola-se as n -ésimas variáveis das n -ésimas equações, obtendo:

$$\begin{aligned} x_1 &= (b_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n)/a_{11} \\ x_2 &= (b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n)/a_{22} \\ &\vdots \\ x_n &= (b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{nn-1}x_{n-1})/a_{nn} \end{aligned} \quad (2.7)$$

Com isso em mãos, se existir um vetor solução inicial $\mathbf{x}^0$ é possível obter os próximos vetores soluções $\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, \dots$, até que a condição de parada seja satisfeita. Para facilitar o entendimento, adiciona-se um novo índice acima das incógnitas para indicar se a mesma pertence a solução atual ou a nova solução a ser descoberta (k é a solução atual e $k+1$ é a nova solução):

$$\begin{aligned}
x_1^{k+1} &= (b_1 - a_{12}x_2^k - a_{13}x_3^k - \dots - a_{1n}x_n^k) / a_{11} \\
x_2^{k+1} &= (b_2 - a_{21}x_1^k - a_{23}x_3^k - \dots - a_{2n}x_n^k) / a_{22} \\
&\vdots \\
x_n^{k+1} &= (b_n - a_{n1}x_1^k - a_{n2}x_2^k - \dots - a_{nn-1}x_{n-1}^k) / a_{nn}
\end{aligned} \tag{2.8}$$

Asano e Colli (2009) apresentam ainda uma condição de garantia de convergência deste método. Tal condição é chamada de Critério de Linhas. Resumidamente, esse critério diz que o valor (absoluto) do elemento da diagonal de determinada linha deverá ser maior que a soma dos demais valores (também absoluto) desta linha, ou de forma geral (Asano e Colli, 2009):

$$\sum_{\substack{j=1 \\ j \neq i}}^n |a_{ij}| < |a_{ii}|, \forall i = 1, \dots, n \tag{2.9}$$

Assim, quando essa condição é atendida para todas as linhas da matriz principal do problema testado, há a garantia que o método irá convergir. Se dividir ambos os lados da condição pelo elemento da diagonal, torna-se possível simplificá-la de forma a ficar:

$$\sum_{\substack{j=1 \\ j \neq i}}^n \left(\frac{|a_{ij}|}{|a_{ii}|} \right) < \left(\frac{|a_{ii}|}{|a_{ii}|} \right), \forall i = 1, \dots, n \tag{2.10}$$

ou ainda

$$\sum_{\substack{j=1 \\ j \neq i}}^n \left(\frac{|a_{ij}|}{|a_{ii}|} \right) < 1, \forall i = 1, \dots, n \tag{2.11}$$

Destaca-se ainda dois pontos importantes (Asano e Colli, 2009): Primeiro é que sempre que o critério das linhas é atendido, há a garantia de que o método irá convergir. Porém não é certo afirmar o oposto, pois é possível que um determinado problema não atenda ao critério, mas ainda assim o método consiga convergir.

O segundo ponto a ser levado em consideração, diz que a troca da ordem de linhas da matriz principal influência no comportamento do critério. Suponha um problema que originalmente atenda ao critério: caso seja feita uma mudança na ordem das linhas pode acontecer que o mesmo não atenda mais ao critério da linha. O inverso também é válido: caso haja um problema que não atenda ao critério e seja feita uma mudança cuidadosa (ou bem planejada) na ordem das linhas é possível que consiga atender ao critério.

Por último, como condição de parada do método, os autores usam o erro relativo entre a nova solução obtida e a solução atual:

$$\frac{\|\mathbf{x}^{k+1} - \mathbf{x}^k\|_{\infty}}{\|\mathbf{x}^{k+1}\|_{\infty}} \quad (2.12)$$

Segue abaixo o algoritmo do método de Jacobi.

Algoritmo 2.1 – Jacobi

Entrada: Matriz de coeficientes $\mathbf{A}$ e vetor independente $\mathbf{b}$

Saída: Vetor solução $\mathbf{x}$

Inicialização: $\mathbf{x}^0 \leftarrow 0$

```

1: Início
2:    $pare \leftarrow \text{falso}$ 
3:   Enquanto NÃO  $pare$ 
4:     Para  $i \leftarrow 1$  Até  $n$  Faça
5:        $x_i^{(k+1)} \leftarrow \left( b_i - \sum_{j=1}^{(i-1)} a_{ij}x_j^k - \sum_{j=i+1}^n a_{ij}x_j^k \right) / a_{ii}$ 
6:     Fim_Para
7:     Se  $\frac{\|\mathbf{x}^{k+1} - \mathbf{x}^k\|}{\|\mathbf{x}^{k+1}\|} \leq \varepsilon$  Então  $pare \leftarrow \text{verdade}$ 
8:     Senão
9:        $k \leftarrow k + 1$ 
10:    Fim_Se
10:  Fim_Enquanto
11: Fim

```

No Algoritmo 2.1 ε representa a tolerância para o erro relativo calculado conforme mostrado acima. Em relação a variável $pare$, ela serve para controle do *loop*, de forma que, enquanto ela tiver o valor falso, o algoritmo irá encontrar uma nova solução, e quando uma solução adequada (que atenda a condição de parada) for obtida ele receberá verdade e então o algoritmo terminará. Por último, k serve somente para contar as iterações realizadas.

2.2.2 Método de Gauss-Seidel

Apesar do método de Jacobi ser simples e apresentar poucos erros de arredondamento, para alguns problemas (Asano e Colli, 2009), somente o critério das linhas não é suficiente para garantir a convergência. Por isso será apresentada uma variação desse método que possui uma condição de convergência menos restrita que a anterior afim de conseguir a garantia de convergir para mais alguns problemas. Esta variação é conhecida como método de Gauss-Seidel.

Retornando a Jacobi, sabe-se que, de forma geral, para descobrir a nova solução a partir da anterior é feito uso da Equação (2.8), a qual mostra que o mesmo encontra todos os novos elementos (x^{k+1}) utilizando os elementos da iteração anterior (x^k). Porém, no método de Gauss-Seidel os elementos novos (x^{k+1}) já encontrados também são utilizados para calcular os demais da iteração atual. Com isso:

$$\begin{aligned}
 x_1^{k+1} &= (b_1 - a_{12}x_2^k - a_{13}x_3^k - \dots - a_{1n}x_n^k) / a_{11} \\
 x_2^{k+1} &= (b_2 - a_{21}x_1^{k+1} - a_{23}x_3^k - \dots - a_{2n}x_n^k) / a_{22} \\
 &\vdots \\
 x_n^{k+1} &= (b_n - a_{n1}x_1^{k+1} - a_{n2}x_2^{k+1} \dots - a_{nn-1}x_{n-1}^{k+1}) / a_{nn}
 \end{aligned} \tag{2.13}$$

Note que a cada novo elemento encontrado, o mesmo será usado para encontrar o próximo elemento. Dessa forma quando chegar ao último elemento ele usará somente os novos elementos encontrados.

Para analisar as condições de convergência tem-se o critério de Sassenfeld que pode ser descrito pelas seguintes equações (Asano e Colli, 2009):

$$\begin{aligned}
\phi_1 &= \frac{|a_{12}| + |a_{13}| + \dots + |a_{1n}|}{|a_{11}|} \\
\phi_2 &= \frac{\phi_1 |a_{21}| + |a_{23}| + \dots + |a_{2n}|}{|a_{22}|} \\
\phi_3 &= \frac{\phi_1 |a_{31}| + \phi_2 |a_{32}| + \dots + |a_{3n}|}{|a_{33}|} \\
&\vdots \\
\phi_i &= \frac{\phi_1 |a_{i1}| + \phi_2 |a_{i2}| + \dots + \phi_{i-1} |a_{ii-1}| + \dots + |a_{in}|}{|a_{ii}|} \\
&\vdots \\
\phi_n &= \frac{\phi_1 |a_{n1}| + \phi_2 |a_{n2}| + \dots + \phi_{n-1} |a_{(n-1)n}|}{|a_{nn}|}
\end{aligned} \tag{2.14}$$

Há a garantia de convergência quando cada uma dessas equações, ou seja, quando cada um dos ϕ forem estritamente menores que 1. Generalizando (2.14) fica:

$$\phi_i = \frac{1}{|a_{ii}|} \left(\phi_j \sum_{j=1}^{i-1} |a_{ij}| + \sum_{j=i+1}^n |a_{ij}| \right) < 1, \quad \forall i = 1, \dots, n \tag{2.15}$$

Note acima que, quando o primeiro ϕ é calculado pela equação acima, o termo do primeiro somatório se anula restando o somente o segundo. Em seguida, veja também que, quando o último ϕ é calculado, o termo do segundo somatório se anula restando somente o primeiro, correspondendo com a forma aberta apresentada anteriormente.

Mais uma vez, será utilizado como condição de parada o mesmo erro relativo (2.12) mostrado no método de Jacobi. Segue abaixo o algoritmo do método de Gauss-Seidel:

Algoritmo 2.2 – Gauss-Seidel

Entrada: Matriz de coeficientes $\mathbf{A}$ e vetor independente $\mathbf{b}$

Saída: Vetor solução $\mathbf{x}$

Inicialização: $\mathbf{x}^0 \leftarrow 0$

```

1: Início
2:    $pare \leftarrow \text{falso}$ 
3:   Enquanto NÃO  $pare$ 
4:     Para  $i \leftarrow 1$  Até  $n$  Faça
5:        $x_i^{(k+1)} \leftarrow \left( b_i - \sum_{j=1}^{(i-1)} a_{ij}x_j^{k+1} - \sum_{j=i+1}^n a_{ij}x_j^k \right) / a_{ii}$ 
6:     Fim Para
7:     Se  $\frac{\|\mathbf{x}^{k+1} - \mathbf{x}^k\|}{\|\mathbf{x}^{k+1}\|} \leq \varepsilon$  Então  $pare \leftarrow \text{Verdade}$ 
8:     Senão
9:        $k \leftarrow k + 1$ 
10:    Fim Se
11:  Fim Enquanto
12: Fim

```

Observe que o método de Gauss-Seidel segue praticamente os passos descritos para o método de Jacobi, com exceção apenas do cálculo de $\mathbf{x}^{k+1}$, onde, agora, no primeiro somatório, ele utiliza os elementos da solução já calculados na iteração atual. Perceba também que esse cálculo possui o mesmo comportamento do cálculo dos ϕ da condição de convergência: para primeira linha, anula o primeiro somatório, então usa somente os elementos da iteração anterior e, para a última linha, ele anula o segundo somatório, sendo utilizado apenas os elementos da iteração atual.

2.2.3 Método do Gradiente Conjugado

Segundo Cunha (2000), desde a década de 50, vários estudos envolvendo o método do Gradiente Conjugado foram realizados, voltados principalmente para solução de sistemas obtidos pela discretização de Equações Diferenciais Parciais (EDP's). Para trabalhar com este método, primeiro é necessário assumir que a matriz de coeficientes do problema seja simétrica e positiva definida ou seja:

$$\begin{cases} \mathbf{A}^T = \mathbf{A} \\ \mathbf{x}^T \mathbf{A} \mathbf{x} > 0 \end{cases} \quad (2.16)$$

Baseado no método do gradiente, ele busca encontrar a solução do problema minimizando a função quadrática:

$$F(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} - \mathbf{b}^T \mathbf{x} \quad (2.17)$$

O método do gradiente conjugado considera n direções, as quais serão conjugadas linearmente independentes, $\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_n$, onde $\mathbf{d}^k$ é o vetor das n direções, minimizando a função anterior na forma (Chung, 2015):

$$F(\mathbf{x}^k + \alpha \mathbf{d}^k) = \frac{1}{2} (\mathbf{x}^k + \alpha \mathbf{d}^k)^T \mathbf{A} (\mathbf{x}^k + \alpha \mathbf{d}^k) - \mathbf{b}^T (\mathbf{x}^k + \alpha \mathbf{d}^k) \quad (2.18)$$

Na equação (2.18) $\mathbf{x}^k$ representa o vetor solução anterior, e α é tamanho do passo dado pelo método nas direções obtidas. Vale lembrar também que dois vetores $\mathbf{y}_1$ e $\mathbf{y}_2$ são conjugados quando obedecem a seguinte regra:

$$\mathbf{y}_1^T \mathbf{A} \mathbf{y}_2 = \mathbf{y}_2^T \mathbf{A} \mathbf{y}_1 = 0 \quad (2.19)$$

Agora, é necessário encontrar os valores das variáveis importantes para minimizar o valor da função $F(\mathbf{x} + \alpha \mathbf{d})$. Chung (2015) demonstra um passo a passo para obtê-las. Primeiro em relação a α (tamanho do passo), o qual pode ser dito como o fator que minimizará a função F :

$$\begin{aligned} \frac{\partial}{\partial \alpha} F(\mathbf{x}^k + \alpha \mathbf{d}^k) &= \frac{1}{2} \mathbf{d}^k \cdot \mathbf{A} (\mathbf{x}^k + \alpha \mathbf{d}^k) + \frac{1}{2} (\mathbf{x}^k + \alpha \mathbf{d}^k) \cdot \mathbf{A} \mathbf{d}^k - \mathbf{b} \cdot \mathbf{d}^k \\ \frac{\partial}{\partial \alpha} F(\mathbf{x}^k + \alpha \mathbf{d}^k) &= \frac{1}{2} \mathbf{A} (\mathbf{x}^k + \alpha \mathbf{d}^k) \cdot \mathbf{d}^k + \frac{1}{2} (\mathbf{x}^k + \alpha \mathbf{d}^k) \cdot \mathbf{A} \mathbf{d}^k - \mathbf{b} \cdot \mathbf{d}^k \\ \frac{\partial}{\partial \alpha} F(\mathbf{x}^k + \alpha \mathbf{d}^k) &= \frac{1}{2} (\mathbf{x}^k + \alpha \mathbf{d}^k) \cdot \mathbf{A}^T \mathbf{d}^k + \frac{1}{2} (\mathbf{x}^k + \alpha \mathbf{d}^k) \cdot \mathbf{A} \mathbf{d}^k - \mathbf{b} \cdot \mathbf{d}^k \quad (2.20) \\ \frac{\partial}{\partial \alpha} F(\mathbf{x}^k + \alpha \mathbf{d}^k) &= (\mathbf{x}^k + \alpha \mathbf{d}^k) \cdot \mathbf{A} \mathbf{d}^k - \mathbf{b} \cdot \mathbf{d}^k \\ \frac{\partial}{\partial \alpha} F(\mathbf{x}^k + \alpha \mathbf{d}^k) &= \alpha \mathbf{d}^k \cdot \mathbf{A} \mathbf{d}^k + \mathbf{A} \mathbf{x}^k \cdot \mathbf{d}^k - \mathbf{b} \cdot \mathbf{d}^k \end{aligned}$$

Isolando $\mathbf{d}^k$ nos segundo e terceiro termo:

$$\frac{\partial}{\partial \alpha} F(\mathbf{x}^k + \alpha \mathbf{d}^k) = \alpha \mathbf{d}^k \cdot \mathbf{A} \mathbf{d}^k + (\mathbf{A} \mathbf{x}^k - \mathbf{b}) \cdot \mathbf{d}^k \quad (2.21)$$

Por último, o termo entre parênteses representa exatamente o resíduo descrito pela equação (2.1), logo:

$$\frac{\partial}{\partial \alpha} F(\mathbf{x}^k + \alpha \mathbf{d}^k) = \alpha \mathbf{d}^k \cdot \mathbf{A} \mathbf{d}^k - \mathbf{r}^k \cdot \mathbf{d}^k \quad (2.22)$$

O próximo passo é, lembrando que ao realizar a minimização de uma função a derivada da mesma será nula, isolar α :

$$\alpha = \frac{\mathbf{r}^k \cdot \mathbf{d}^k}{\mathbf{d}^k \cdot \mathbf{A} \mathbf{d}^k} \quad (2.23)$$

Assumindo que o novo vetor solução é calculado levando em conta a direção vezes o tamanho do passo, tem-se: $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha \mathbf{d}^k$. Com isso, e levando em conta que o resíduo foi definido pela Equação (2.1), é definido o novo vetor $\mathbf{r}$ para o método gradiente conjugado a partir de:

$$\begin{aligned} \mathbf{r}^{k+1} &= \mathbf{b} - \mathbf{A} \mathbf{x}^{k+1} \\ \mathbf{r}^{k+1} &= \mathbf{b} - \mathbf{A} (\mathbf{x}^k + \alpha \mathbf{d}^k) \\ \mathbf{r}^{k+1} &= \mathbf{b} - \mathbf{A} \mathbf{x}^k - \alpha \mathbf{A} \mathbf{d}^k \\ \mathbf{r}^{k+1} &= \mathbf{r}^k - \alpha \mathbf{A} \mathbf{d}^k \end{aligned} \quad (2.24)$$

Por fim, basta atualizar os valores do vetor de direção, visando minimizar o vetor resíduo. Semelhante a atualização da solução, as novas direções serão a soma do novo resíduo obtido mais as direções atuais multiplicado por um coeficiente φ , ou seja: $\mathbf{d}^{k+1} = \mathbf{r}^{k+1} + \varphi \mathbf{d}^k$.

O φ acima, possui um papel de minimização parecido com o do α . Porém, este agora irá atualizar as direções de forma a minimizar o resíduo. Este escalar pode ser definido partindo do conceito de que as direções atuais e as novas devem ser conjugadas:

$$\begin{aligned}
 \mathbf{d}^k \cdot \mathbf{A}\mathbf{d}^{k+1} &= 0 \\
 \mathbf{d}^k \cdot \mathbf{A}(\mathbf{r}^{k+1} + \varphi \mathbf{d}^k) &= 0 \\
 \mathbf{d}^k \cdot \mathbf{A}\mathbf{r}^{k+1} + \varphi(\mathbf{d}^k \cdot \mathbf{A}\mathbf{d}^k) &= 0 \\
 \mathbf{A}^\top \mathbf{d}^k \cdot \mathbf{r}^{k+1} + \varphi(\mathbf{d}^k \cdot \mathbf{A}\mathbf{d}^k) &= 0
 \end{aligned} \tag{2.25}$$

E, assim, isolando φ : $\varphi = -\frac{\mathbf{r}^{k+1} \cdot \mathbf{A}\mathbf{d}^k}{\mathbf{d}^k \cdot \mathbf{A}\mathbf{d}^k}$

Com esses passos definidos, é possível, agora, desenvolver um algoritmo do método do Gradiente Conjugado:

Algoritmo 2.3 – Gradiente Conjugado

Entrada: Matriz de coeficientes $\mathbf{A}$ e vetor independente $\mathbf{b}$

Saída: Vetor solução $\mathbf{x}$

Inicialização: $\mathbf{d}^0 = \mathbf{r}^0 = \mathbf{b} - \mathbf{A}\mathbf{x}^0$

```

1: Início
2:   Para  $k \leftarrow 1$  Até  $2n$  Faça
3:      $\alpha = \frac{\mathbf{r}^k \cdot \mathbf{d}^k}{\mathbf{d}^k \cdot \mathbf{A}\mathbf{d}^k}$  // 2ª Etapa: Calcular tamanho do passo
4:      $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha \mathbf{d}^k$  // 3ª Etapa: Atualizar solução
5:      $\mathbf{r}^{k+1} = \mathbf{r}^k - \alpha \mathbf{A}\mathbf{d}^k$  // 4ª Etapa: Atualizar resíduo
6:      $\varphi = -\frac{\mathbf{r}^{k+1} \cdot \mathbf{A}\mathbf{d}^k}{\mathbf{d}^k \cdot \mathbf{A}\mathbf{d}^k}$  // 5ª Etapa: Encontrar coeficiente  $\varphi$ 
7:      $\mathbf{d}^{k+1} = \mathbf{r}^{k+1} + \varphi \mathbf{d}^k$  // 6ª Etapa: Atualizar direção
8:     Se  $(|\mathbf{x}^{k+1} - \mathbf{x}^k| < \varepsilon)$  Então
9:       Retorne  $\mathbf{x}^{k+1}$ 
10:    Senão
11:      Se  $(k=2n)$ , então não convergiu
12:    Fim_Se
13:  Fim_Para
14: Fim

```

Note que o algoritmo do Gradiente Conjugado realiza mais passos que os métodos anteriores. Perceba também que na inicialização o vetor de direções é configurado para ser

correspondente ao resíduo do sistema linear, considerando a solução inicial também preparada na inicialização.

Com os dados iniciais em mãos, basta agora repetir os passos do método a cada iteração: primeiro calcula-se o tamanho do passo (α) que será dado. Depois, com ele, deve-se calcular a nova solução levando em conta também a direção. Depois o resíduo é atualizado e, com esse novo resíduo, é possível encontrar o coeficiente (φ) que ajudará a obter a nova direção que o minimize. Por fim, basta atualizar a direção e verificar se as condições de paradas foram satisfeitas. Se sim, o algoritmo é finalizado, se não ele segue para a próxima iteração, repetindo os mesmos passos com os dados atualizados. Caso atinja o limite máximo de iterações e não satisfaça as condições de paradas, o algoritmo também é finalizado, mas considerando que método não convergiu para a solução deste problema. Vale lembrar que, para problemas cuja matriz principal seja simétrica definida positiva, este método possui a garantia de que sempre irá convergir.

2.2.4 Método do Resíduo Mínimo Generalizado Reinicializado (GMRES(m))

O método do Resíduo Mínimo Generalizado (*Generalized Minimal Residual Method* – GMRES) de Saad e Schultz (1986) é uma generalização de outro método, o Resíduo Mínimo (do inglês *Minimal Residual Method* – MINRES, para mais detalhes sobre esse método leia Paige e Saunders (1975)), variante do gradiente conjugado que consegue resolver sistemas em que sua matriz de coeficientes é simétrica e indefinida.

O GMRES foi desenvolvido afim de resolver uma classe de problemas que seu antecessor não conseguia: sistema lineares que não eram simétricos. Para isso, seu objetivo era minimizar a norma do resíduo do problema, utilizando um subespaço conhecido como subespaço de Krylov.

Tendo como referência o trabalho de Bocanegra (2005) o subespaço de Krilov de dimensão m , representado por $K_m(A, r^0)$ é definido pelo espaço gerado pelas iterações de uma matriz A a partir de um vetor r^0 dado por:

$$K_m(A, r^0) = r^0, Ar^0, A^2r^0, A^3r^0, \dots, A^m r^0 \quad (2.26)$$

Note que $\mathbf{r}^0$ é o resíduo da solução inicial, ou seja, $\mathbf{r}^0 = \mathbf{b} - \mathbf{A}\mathbf{x}^0$. Outro ponto importante a citar é que a solução aproximada é obtida por $\mathbf{x}^m = \mathbf{x}^0 + \mathbf{K}_m$ e ela só é adotada (apropriada) quando o seu resíduo for ortogonal ao subespaço de Krylov:

$$\mathbf{b} - \mathbf{A}\mathbf{x}^m \perp \mathbf{K}_m \quad (2.27)$$

O GMRES possui, por natureza dos métodos que utilizam o subespaço de Krylov, uma garantia de convergência em até n iterações, correspondente a dimensão do problema. Porém, possui uma desvantagem: sua complexidade é muito alta. Só para armazenamento e processamento sua complexidade seria na ordem de $O(n^2)$ e $O(n^3)$, respectivamente (Gonzalez, 2005).

Assim, Saad e Schultz (1986) também propuseram outra versão de seu método, a qual é utilizada neste trabalho: o GMRES Reinicializado ou apenas GMRES(m). Neste, ao gerar o subespaço de Krylov, a dimensão do mesmo é limitada (reduzida) pelo parâmetro m , que geralmente é bem menor que a dimensão do problema.

Com isso, conseguiu-se reduzir a ordem da complexidade do método, levando em conta armazenamento e processamento, para $O(m^2)$ e $O(mn)$, respectivamente (Gonzalez, 2005). Em contra partida, ele perde a garantia de convergência em até n iterações, ou a mesma pode ser muito lenta, sem contar da chance de ocorrer estagnação.

Limitando esse subespaço, projeta-se a matriz de coeficiente $\mathbf{A}$ nesse subespaço, obtêm-se uma nova matriz conhecida como matriz superior de Hessemberg ($\mathbf{H}$) de dimensão m . Essa matriz tem por característica ser uma quase triangular superior, por possuir uma subdiagonal.

Para obter $\mathbf{H}$ foi utilizado o método de Arnoldi (para mais detalhes leia Silva (2017) e Arnoldi (1951)). Outra característica importante dessa matriz é que seus auto valores são bem próximos aos da matriz original (Saad, 2003; Bocanegra, 2005), o que facilita na hora de resolver sistemas de grande porte.

É importante citar, ainda, que Arnoldi não retorna exatamente uma matriz de Hessemberg mas, sim, uma quase Hessemberg. Essa matriz possuirá uma linha a mais, a qual é ignorada ao ser utilizada no GMRES. Além de $\mathbf{H}$, o processo de Arnoldi também retorna a matriz $\mathbf{V}$, de dimensão $n \times m$, na qual seus vetores colunas representam uma base ortonormal no subespaço de Krylov.

Considerando a relaxação de Arnoldi tem-se que $AV_m = V_{m+1}\tilde{H}_m$. É possível também reescrever $\mathbf{x}^m = \mathbf{x}^0 + \mathbf{K}_m$ como $\mathbf{x}^m = \mathbf{x}^0 + V_m \mathbf{y}$. Por último, definindo $\beta = \|\mathbf{r}^0\|_2$ e sabendo que o primeiro vetor da base V será gerado por $\mathbf{v}_1 = \mathbf{r}^0 / \beta$, isolando $\mathbf{r}^0$ obtêm-se $\mathbf{r}^0 = \beta \mathbf{v}_1$, porém, levando em conta o vetor canônico $\mathbf{e}^T = [1, 0, 0, \dots, 0]$ então $\mathbf{r}^0 = \beta V_m \mathbf{e}^1$. Agora, a partir do resíduo para a solução $\mathbf{x}^m$ pode-se escrever:

$$\begin{aligned}
 \mathbf{r}^m &= \mathbf{b} - A\mathbf{x}^m \\
 \mathbf{r}^m &= \mathbf{b} - A(\mathbf{x}^0 + V_m \mathbf{y}) \\
 \mathbf{r}^m &= \mathbf{b} - A\mathbf{x}^0 - AV_m \mathbf{y} \\
 \mathbf{r}^m &= \mathbf{r}^0 - AV_m \mathbf{y} \\
 \mathbf{r}^m &= \beta V_{m+1} \mathbf{e} - V_{m+1} \tilde{H}_m \mathbf{y} \\
 \mathbf{r}^m &= V_{m+1} (\beta \mathbf{e} - \tilde{H}_m \mathbf{y})
 \end{aligned} \tag{2.28}$$

Agora, calculando a norma deste resíduo conforme foi feito em Silva (2017), lembrando que V_{m+1} é um vetor ortogonal:

$$\|\mathbf{r}^m\|_2 = \|\beta \mathbf{e} - \tilde{H}_m \mathbf{y}\|_2 \tag{2.29}$$

Com isso é possível ver que, para satisfazer o método, minimizar a norma do resíduo, deve-se resolver um problema de mínimos quadrados $\min_y \|\beta \mathbf{e} - \tilde{H}_m \mathbf{y}\|_2$. Para resolvê-lo, foi utilizado outro método: a fatoração QR (para mais detalhes leia Silva (2017)), que retornará o valor de $\mathbf{y}$.

Com isso, basta atualizar a solução anterior fazendo $\mathbf{x}_m = \mathbf{x}_0 + V_m \mathbf{y}_m$ e avaliar se a norma do resíduo atual satisfaz a tolerância estipulada. Segue agora abaixo o algoritmo do GMRES(m).

Algoritmo 2.4 – GMRES(m)

Entrada: Matriz de coeficientes $\mathbf{A}$ e vetor independente $\mathbf{b}$

Saída: Vetor solução $\mathbf{x}$

Inicialização: Escolha $\mathbf{x}^0 \leftarrow \mathbf{0}$, $it \leftarrow 0$ e a dimensão de m
 Calcule $\mathbf{r}^0$, β e $\mathbf{v}_1$

```

1: Início
2:   Faça
3:      $it \leftarrow it + 1$ 
4:     Para  $j \leftarrow 1$  Até  $m$  Faça      // Processo de Arnoldi
5:       Calcule  $\mathbf{w} \leftarrow \mathbf{A}\mathbf{v}_j$ 
6:       Para  $i \leftarrow 1$  Até  $j$  Faça
7:          $h_{i,j} \leftarrow \langle \mathbf{w}, \mathbf{v}_i \rangle$       // Produto interno
8:          $\mathbf{w} \leftarrow \mathbf{w} - h_{i,j}\mathbf{v}_i$ 
9:       Fim_Para
10:       $h_{j+1,j} \leftarrow \|\mathbf{w}\|_2$ 
11:      Se ( $h_{j+1,j} < tol1$ ) Então
12:        Faça  $m \leftarrow j$  e saia do laço
13:      Fim_Se
14:       $\mathbf{v}_{j+1} \leftarrow \mathbf{w} / h_{j+1,j}$ 
15:    Fim_Para
16:    // Decomposição QR
17:    Calcule  $\mathbf{y}_m$  em que  $\mathbf{y}_m \leftarrow \min_{\mathbf{y}} \|\beta \mathbf{e}_1 - \mathbf{H}_m \mathbf{y}\|_2$ ,  $\mathbf{e}_1 \leftarrow [1, 0, \dots, 0]^T$ 
18:    // Atualização da Solução
19:    Calcule  $\mathbf{x}_m \leftarrow \mathbf{x}_0 + \mathbf{V}_m \mathbf{y}_m$ , em que  $\mathbf{V}_m \leftarrow [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m]$ 
20:    Se ( $\beta = \|\mathbf{r}\|_2 < tol2$ ) Então // Calcular e verificar novo resíduo
21:      Pare
22:    Senão
23:       $\mathbf{v}_1 = \mathbf{r}_1 / \beta$ 
24:    Fim_Se
25:  Enquanto ( $it < MAX$ )
26:  Fim

```

Veja ainda que na inicialização é necessário definir o valor do parâmetro m . Tal definição é bem complexa, como pode ser visto em Sousa e Silva (2016). Por padrão, m é considerado como uma porcentagem da dimensão n do problema. Porém, pode ocorrer, caso o valor desse parâmetro seja pequeno ou muito grande, que método não convirja. Saad e Schultz (1986) propuseram um teorema para a escolha do valor desse parâmetro, mas como discutido em Silva (2017), não é possível dizer que este seja totalmente adequado.

Foram determinadas também algumas constantes. $tol1$ representa a tolerância para o os elementos de $\mathbf{H}$ calculado dentro do método de Arnoldi, desde que tendem a ser muito próximos de zero. $tol2$ é a tolerância para a norma do resíduo atual, sendo assim, um dos critérios de parada. Por último, MAX representa o máximo de iterações que o algoritmo irá realizar, definida como sendo igual a dimensão n do problema.

Para um estudo mais aprofundado deste método, recomenda-se as seguintes leituras: Saad e Schultz (1986), Strikwerda e Stodder (1995), Gonzalez (2005), Motta (2010).

Capítulo 3

Esparsidade e Estruturas de Dados

Como mencionado anteriormente, em muitos problemas reais em que se necessita resolver um sistema linear, a matriz dos coeficientes A é normalmente de grande porte (milhares de variáveis) e muito esparsa (poucos elementos não nulos). Estas características possuem grande impacto em como a matriz deve ser armazenada e como os cálculos com os elementos devem ser realizados. Portanto, neste capítulo é dialogado sobre esparsidade e densidade assim como são apresentadas estruturas de dados e técnicas que foram utilizadas para tornar os métodos mais eficientes na resolução do sistema linear considerado.

3.1 Esparsidade e Densidade

Matrizes de sistemas lineares relacionadas a problemas reais, obtidos através de repositórios on-line como o Matrix Market (<https://math.nist.gov/MatrixMarket/>) e o SuiteSparse Matrix Collection (<https://sparse.tamu.edu/>), tendem a serem grandes e com poucos elementos não nulos.

Ao tratar estas matrizes de grande porte, é necessário considerar dois aspectos principais: a quantidade de memória que será necessária para armazenar os elementos e o tempo para o processamento das principais operações realizadas pelo método em questão.

Obviamente, a quantidade de memória requerida está diretamente relacionada com a dimensão do problema. Logo, quanto maior a dimensão do problema, maior a quantidade de elementos a serem armazenados (se n é a dimensão da matriz, o total de elementos é n^2). Mas, em problemas esparsos não é necessário armazenar e manipular este total de elementos.

O tempo de processamento das operações está relacionado com a quantidade de elementos nulos que são manipulados. Logo, um armazenamento inadequado poderia guardar e realizar cálculos com uma quantidade grande elementos nulos, o que acarreta em grandes desperdícios de memória e tempo de processamento.

Considerando que a principal operação do GMRES(m) e dos outros métodos trabalhados é a multiplicação de uma matriz por um vetor, tem-se diversas operações (n vezes para cada linha tratada) que envolve a multiplicação de um elemento da matriz por um componente do vetor. Logo, nesta situação haverá um custo computacional muito elevado para multiplicar e somar elementos nulos.

Os conceitos de esparsidade e de densidade de uma matriz são correlacionados. A esparsidade trata da proporção de elementos nulos de uma matriz em relação ao total de elementos, enquanto a densidade mostra a proporção de elementos não-nulos em relação a quantidade total. Com isso, pode-se definir:

$$Esparsidade(\%): \frac{nz}{n^2} \times 100 \quad (3.1)$$

$$Densidade(\%): \frac{nnz}{n^2} \times 100 \quad (3.2)$$

em que nz representa a quantidade de zeros e nnz a quantidade de números não nulos (ou não zeros). Com esses dois valores é possível verificar a esparsidade e densidade de uma determinada matriz. Por exemplo, considere a matriz **E1** a seguir.

$$\mathbf{E1} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0,45 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0,1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0,45 \\ -0,04 & 0 & 0 & 0 & 1 & 0 & 0 \\ -0,02 & 0 & 0 & 0 & 0 & 1 & 0 \\ -0,01 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

A dimensão é 7, logo a quantidade total de elementos que ela possui é $7 \times 7 = 49$. Observe que sua densidade é de aproximadamente 31%, ou seja, 15 elementos não nulos de um total de 49, o que pode ser considerada uma densidade razoável; sua esparsidade é aproximadamente 69%. Sendo assim, tem-se uma matriz razoavelmente esparsa com a maioria de seus elementos nulos. Note ainda a complementaridade entre estes dois conceitos.

Portanto, não é preciso trabalhar com os dois ao mesmo tempo, por isso no presente trabalho será observada somente a densidade das matrizes.

Destaca-se ainda que é extremamente importante aproveitar a característica de esparsidade das matrizes para trabalhar com estruturas de dados que não realiza cálculos nulos, evitando assim o desperdício de memória e tempo de processamento. A seguir, serão apresentadas as estruturas de dados utilizadas nos métodos iterativos para resolver computacionalmente os diversos sistemas lineares tratados.

3.2 Estruturas de Dados Compactas

Considerando uma matriz A , uma forma comum para armazená-la é conhecida por estrutura indexada (ou coordenadas) em que cada elemento é guardado em posições subsequentes de memória no computador e sua representação é igual ao tratamento algébrico dado por $A[i][j]$. Neste caso, o acesso a cada elemento é direto tendo como informação a linha i e coluna j . Porém, este tipo de estrutura se torna totalmente inadequada quando se trabalha com matrizes esparsas e de grande porte.

Portanto, a estrutura indexada armazena todos os elementos (nulos e não-nulos) da matriz do sistema linear. Em problemas pequenos, com dezenas de variáveis, não é um impedimento. No entanto, quando se tem problemas na casa de centenas e milhares de variáveis, o armazenamento e as operações com os elementos nulos é um desperdício de memória e tempo, sendo que na maioria dos casos, com dimensões maiores, é impraticável implementar um método iterativo para resolver o problema.

Assim, para se ter uma implementação viável computacionalmente em termos de memória e tempo de processamento, é fundamental o uso de estruturas de dados eficientes para estes dois aspectos. Logo, as estruturas de dados de Linhas Esparsas Compactadas (*Compressed Sparse Row - CSR*) e a de Colunas Esparsas Compactadas (*Compressed Sparse Column - CSC*), são adequadas para armazenar matrizes com muitos elementos nulos e realizar operações que envolvem a multiplicação de uma matriz por um vetor. A seguir, ambas serão apresentadas.

3.2.1 Linhas Esparsas Compactadas (*Compressed Sparse Row - CSR*)

A estrutura de dados CSR (Rodrigues, 2011), trabalha de forma a guardar os elementos não-nulos ordenados por linha. Para isso, ela utiliza três vetores, evitando assim o desperdício de memória. Vale mencionar que já existe na literatura (Lugon, 2013) rotinas prontas de

multiplicação de matriz por vetor, utilizando esta estrutura, permitindo evitar assim o armazenamento de elementos nulos e os cálculos que os envolvem.

Considerando que n representa a dimensão da matriz e nnz representa quantidade de elementos não nulos, a estrutura CSR é constituída por 3 vetores da seguinte forma:

- Primeiro vetor: **val** armazena os valores dos elementos não nulos linha por linha na ordem que aparecem na matriz do sistema original;
- Segundo vetor: **cind** é usado para armazenar o índice da coluna de cada elemento não nulo;
- Terceiro vetor: **rptr** é utilizado para guardar a posição do início de cada linha no vetor de valores (**val**).

Os vetores **val** e **cind** têm dimensão nnz e o vetor **rptr** tem dimensão $n + 1$.

Para ilustrar melhor a forma de armazenamento considere a matriz **E1** apresentada anteriormente. Assim, os três vetores utilizados para a estrutura CSR da referida matriz são apresentados na Figura 3.1.

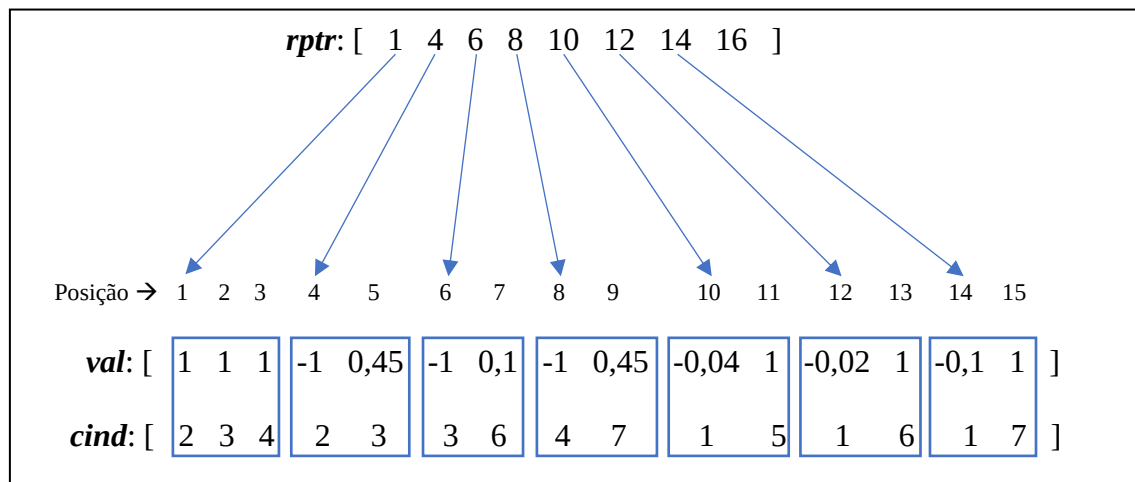


Figura 3.1 – Representação da estrutura CSR.

Vale observar que, se uma variável real ocupa 8 bytes e uma do tipo inteira ocupa 5 bytes na memória do computador, então na forma coordenada o armazenamento de uma matriz de ordem n ocupa $8n^2$ bytes e no formato CSR ocupa $8nnz + 4nnz + 4(n+1)$ bytes.

Para observar esta diferença, uma matriz de médio porte de dimensão 1500 e 2% de esparsidade, a forma coordenada requer aproximadamente 18 Megabytes de armazenamento,

ao passo que o formato CSR necessita de apenas 0,5 Megabyte. Isto é, uma considerável vantagem somente em termos de memória.

3.2.2 Colunas Esparsas Compactadas (*Compressed Sparse Column - CSC*)

Trabalhando de maneira análoga a CSR, a estrutura de dados CSC armazena somente os elementos não nulos em ordem por coluna. Logo, é também composta por 3 vetores como a estrutura anterior.

Na Figura 3.2 são apresentados os três vetores utilizados para o armazenamento da matriz da **E1** no formato CSC.

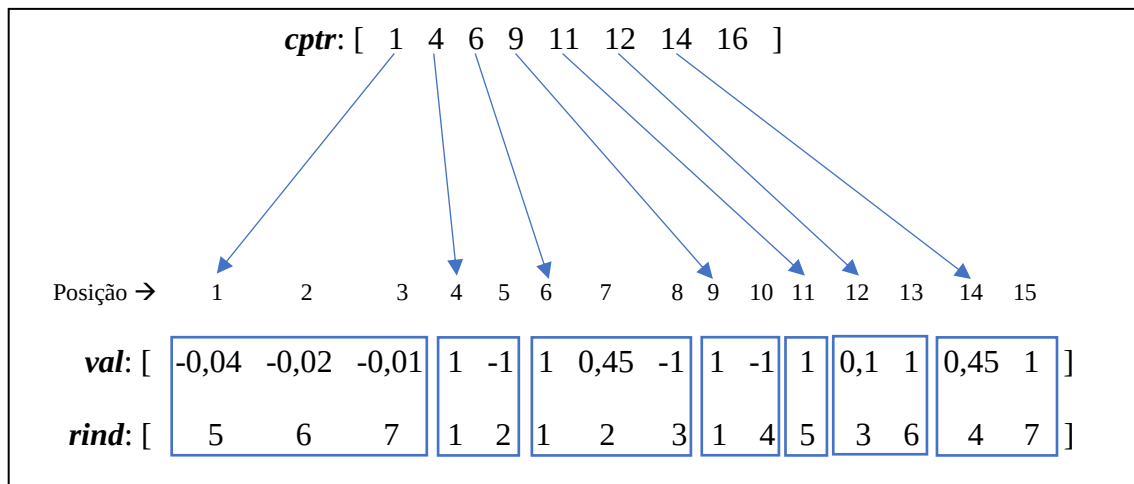


Figura 3.2 – Representação da Estrutura CSC.

Semelhante à estrutura CSR, esta forma de armazenar uma matriz esparsa, precisa de apenas $8nnz + 4nnz + 4(n+1)$ bytes.

A escolha de qual estrutura utilizar deve-se levar em conta as principais operações realizadas pelo método que esteja trabalhando. Neste trabalho será utilizada apenas a estrutura CSR, pois para todos os métodos tratados, a principal operação é a multiplicação de uma matriz por vetor. Destaca-se que o trabalho de Santos (2020) se utiliza as estruturas CSC e CSR, na implementação do método Dual Simplex Canalizado para as quais se teve um ótimo desempenho computacional.

Vale destacar, ainda, que existem outras estruturas que também são adequadas para tratar matrizes esparsas como Armazenamento Diagonal Compactado (*Compressed Diagonal Storage - CDS*) e a *Skyline* (Pachas, 2009; Rodrigues, 2011; Coelho, 2014).

3.3 Manipulação de Vetores Esparsos

O trabalho de Santos (2020) apresenta três formas de representar um vetor esparsos: a primeira seria a forma densa, onde ele armazena todos os valores, mesmo os nulos, e em operações básicas todos são utilizados. A segunda forma, chamada de indexada, armazena os elementos nulos, mas também guarda os índices dos não nulos, evitando o desperdício de operações com valores nulos.

Por último, tem-se a estrutura compacta, a qual armazena apenas os valores não nulos e seus índices correspondentes a sua posição em um vetor de referência, como se fosse um ponteiro. Desta maneira, além de evitar operações desnecessárias consegue-se também uma economia de memória ao não armazenar zeros. Assim, nesta seção será vista a utilização desta estrutura para vetores esparsos.

Considere uma soma de dois vetores dada pela seguinte expressão:

$$\mathbf{v}_2 = \mathbf{v}_2 + \lambda \mathbf{v}_1 \quad (3.3)$$

em que $\lambda \in \mathbb{R}$ é um escalar qualquer.

Para exemplificar a forma compacta de armazenamento, considere os seguintes vetores esparsos:

$$\mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ -3 \\ 1 \end{bmatrix} \quad e \quad \mathbf{v}_2 = \begin{bmatrix} 4 \\ 0 \\ 7 \\ 2 \\ 0 \end{bmatrix} \quad (3.4)$$

Agora, suponha $\lambda = 1$. Note que em ambos os vetores existem posições nulas. Para não armazenar elementos nulos é necessário uma variável para guardar a quantidade total de elementos não nulos de cada vetor (neste caso $nzv1$ e $nzv2$), um vetor para armazenar os valores não nulos na mesma ordem que aparecem nos vetores esparsos (neste caso $valv1$ e $valv2$) e por último, outro vetor para guardar as posições de cada elemento não nulo, também em ordem ($indv1$ e $indv2$).

Assim, os vetores esparsos $\mathbf{V}_1$ e $\mathbf{V}_2$ assumirão a seguinte forma:

$$\begin{aligned}
 \mathbf{v}_1 &= \begin{cases} nzv1 = 3 \\ \mathbf{valv1}^T = [1 \quad -3 \quad 1] \\ \mathbf{indv1}^T = [1 \quad 4 \quad 5] \end{cases} \\
 \mathbf{v}_2 &= \begin{cases} nzv2 = 3 \\ \mathbf{valv2}^T = [4 \quad 7 \quad 2] \\ \mathbf{indv2}^T = [1 \quad 3 \quad 4] \end{cases}
 \end{aligned} \tag{3.5}$$

Na Figura 3.3 é demonstrado como será a soma dos vetores acima, utilizando esta estrutura compacta.

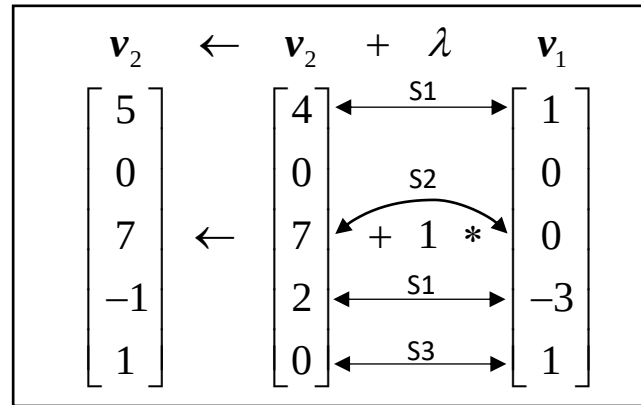


Figura 3.3 – Soma de dois vetores esparsos.

Observe que a soma não será realizada em posições que sejam nulas em ambos os vetores. S1 representa as somas que são realmente efetuadas. S2 demonstra quando a posição do segundo vetor é nula enquanto a do primeiro (correspondente ao que receberá o resultado final da soma) é não nula, por isso nenhuma ação é tomada. Por último S3 destaca quando no vetor que recebe a soma há uma posição nula e no outro não. Neste ocorrerá um preenchimento, logo, uma mudança na representação da estrutura.

No Algoritmo 3.1 é mostrado como a soma de dois vetores é realizada com essa estrutura:

Algoritmo 3.1 – Soma de Dois Vetores Esparsos

Entrada: Vetores de valores **valv1** e **valv2**, vetores de índices correspondentes **indv1** e **indv2** e tamanhos **nzv1** e **nzv2**

Saída: Vetores **valv2** e **indv2** após a soma e o novo tamanho **nzv2**

```

1: Início
2:   Para  $j \leftarrow 1$  Até  $nzv1$  Faça
3:      $wt[indv1[j]] \leftarrow valv1[j]$ 
4:   Fim_Para

5:   Para  $j \leftarrow 1$  Até  $nzv2$  Faça
6:      $i \leftarrow indv2[j]$ 
7:     Se ( $wt[i] \neq 0$ ) Então
8:        $valv2[j] \leftarrow valv2[j] + \lambda * wt[i]$ 
9:        $wt[i] \leftarrow 0$ 
10:    Fim_Se
11:  Fim_Para

12:  Para  $j \leftarrow 1$  Até  $nzv1$  Faça
13:     $i \leftarrow indv1[j]$ 
14:    Se ( $wt[i] \neq 0$ ) Então
15:       $nzv2 \leftarrow nzv2 + 1$ 
16:       $valv2[nzv2] \leftarrow \lambda * wt[i]$ 
17:       $indv2[nzv2] \leftarrow i$ 
18:       $wt[i] \leftarrow 0$ 
19:    Fim_Se
20:  Fim_Para
21: Fim

```

O algoritmo da soma pode ser separado em três partes. A primeira parte, a partir da linha 2, inicializa o vetor **wt**, chamado aqui de vetor de ‘trabalho’, controla quando será tomada alguma ação (atualização de elementos ou preenchimento de posição) para as posições não nulas de v_1 . Na segunda parte, a partir da linha 6, serão atualizadas as posições não nulas de v_1 que também são não nulas em v_2 . A última parte, linha 15, realiza o chamado preenchimento, que ocorre quando será feita uma atualização em uma posição que era originalmente nula em v_2 . Esse preenchimento é feito na posição $nzv2 + 1$, aumentando o tamanho dos vetores **valv2** e **indv2**.

Cabe ainda destacar a importância do vetor **w** nas linhas 12 e 23. A cada atualização ou preenchimento é importante zerar a posição correspondente em **wt**, pois assim o algoritmo sabe que já realizou os cálculos nas mesmas pulando para próxima.

Agora, o próximo passo será utilizar essa estrutura no método GMRES(m), especificamente num vetor que seja adequado para tal estrutura, ou seja, um vetor que seja esparso e ainda trabalhe com operações de soma de vetores.

3.3.1 Vetores Esparsos no GMRES(m)

No método do GMRES(m) há o processo de Arnoldi no qual o vetor w é constantemente utilizado em algumas operações. Este vetor tende a ser esparso (não será em todos os casos, mas no geral, principalmente em problemas grandes haverá ocorrências de esparsidade), tornando este vetor adequado para o uso dessa estrutura em vetores. No Algoritmo 3.2 a seguir encontra-se em destaque o método de Arnoldi utilizado no Algoritmo 2.4.

Algoritmo 3.2 – Método de Arnoldi

Entrada: Matriz de coeficientes A , vetor coluna v_j , parâmetro m

Saída: Matriz V atualizada e matriz de Hessenberg H

```

1: Início
2:   Para  $j \leftarrow 1$  Até  $m$  Faça
3:     Calcule  $w \leftarrow Av_j$  // Multiplicação de matriz por vetor
4:     Para  $i \leftarrow 1$  Até  $j$  Faça
5:        $h_{i,j} \leftarrow |w, v_i|$  // Produto interno entre dois vetores
6:        $w \leftarrow w - h_{i,j}v_i$  // Subtração de vetores
7:     Fim_Para
8:      $h_{j+1,j} \leftarrow \|w\|_2$  // Norma de um vetor
9:     Se ( $h_{j+1,j} < tol$ ) Então
10:      Faça  $m \leftarrow j$  e saia do laço
11:     Fim_se
13:      $v_{j+1} \leftarrow w/h_{j+1,j}$  // Divisão de um vetor por um escalar
14:   Fim_Para
15: Fim

```

Perceba que o vetor w é utilizado em 5 momentos. Assim, é necessário revisar o cálculo feito em cada um deles a fim de aproveitar as características da nova estrutura utilizada. O primeiro, na linha 4 do Algoritmo 3.2, trata-se da construção de w , realizada através da multiplicação de uma matriz por um vetor. Note ainda que este vetor é, na verdade, um vetor coluna da matriz V como pode ser observado no Algoritmo 3.3.

Algoritmo 3.3 – Cálculo do vetor w

Entrada: Matriz de coeficientes A (val , $cind$, $rp\text{tr}$), Matriz de vetores
colunas ortonormais V (Subespaço de Krylov), índice do vetor
coluna a ser utilizado

Saída: Vetor auxiliar w

```

1: Início
2:    $nzw \leftarrow 0$ 
3:   Para  $j \leftarrow 1$  Ate  $n$  Faça
4:      $k1 \leftarrow rp\text{tr}[j]$ 
5:      $k2 \leftarrow rp\text{tr}[j+1]-1$ 
6:     Para  $i \leftarrow k1$  Ate  $k2$  Faça
7:        $aux \leftarrow aux + (val[i] * v[cind[i]][indice])$ 
8:     Fim_Para
9:     Se ( $aux \neq 0$ ) Então
10:       $nzw \leftarrow nzw + 1$ 
11:       $valw[nzw] \leftarrow aux$ 
12:       $indw[nzw] \leftarrow j$ 
13:    Fim_Se
14:  Fim_Para
15: Fim

```

Este algoritmo possui dois parâmetros. O primeiro é uma matriz onde suas colunas são as bases do subespaço atualizado até o momento. O outro parâmetro é um valor inteiro que indica qual vetor coluna de V será utilizado para construir w na iteração atual do Arnoldi.

O tamanho do vetor w (nzw) foi iniciado como zero e feitas as multiplicações referentes ao primeiro elemento. O primeiro passo é verificar se o resultado dessa multiplicação é zero. Se for, não armazenará e continuará os próximos elementos. Caso contrário, primeiro incrementa o tamanho do vetor w e aloca um novo espaço da estrutura ($valw$ e $indw$, que são, respectivamente, os vetores para os valores não nulos de w e para seus índices). Por fim, armazena os valores devidos na estrutura. Com isso o vetor w está agora armazenado numa nova estrutura onde não se armazena valores nulos.

No próximo passo w , na linha 7 do Algoritmo 3.2, é utilizado no produto interno entre dois vetores. Observe no Algoritmo 3.4 a seguir como este cálculo fica determinado ao utilizar a estrutura para vetores esparsos.

Algoritmo 3.4 – Produto Interno Entre Dois Vetores

Entrada: Vetor $\mathbf{w}$ ($\mathbf{valw}$ e $\mathbf{indw}$), matriz de vetores colunas ortonormais $\mathbf{V}$ e índice do vetor coluna a ser utilizado

Saída: Produto interno entre o vetor $\mathbf{w}$ e o vetor coluna de $\mathbf{V}$

```

1: Início
2:    $aux \leftarrow 0$ 
3:   Para  $t \leftarrow 1$  Até  $nzw$  Faça
4:      $aux \leftarrow aux + (\mathbf{valw}[t] * \mathbf{V}[\mathbf{indw}[t]][\mathbf{indice}])$ 
5:   Fim_Para
6:   Retorne  $aux$ 
7: Fim

```

No cálculo deste produto interno surgem, novamente, como parâmetros de entrada uma matriz, que no caso do método de Arnoldi será a matriz de vetores bases, e uma constante que indicará qual vetor coluna da matriz $\mathbf{V}$ será utilizado neste produto.

É importante lembrar que, como é utilizado a estrutura de vetores esparsos para $\mathbf{w}$, seus índices ($\mathbf{indw}$) podem não estar necessariamente em ordem, por causa de algum preenchimento. Assim, é importante utilizar, na multiplicação (linha 5), o vetor $\mathbf{indw}$ para localizar os elementos do vetor coluna de $\mathbf{V}$ na ordem correta.

Ainda sobre o Algoritmo 3.2 do método de Arnoldi, na linha 8 há uma subtração de vetores. Tal subtração também pode ser entendida como uma soma de vetores (como visto no algoritmo 3.1) onde o seu λ é -1.

Nessa subtração serão necessários dois passos: primeiro é feita a multiplicação de um vetor por um escalar, a qual é armazenada em uma estrutura de vetor esparsa temporária, e depois, vem a subtração propriamente dita. Veja o primeiro passo no Algoritmo 3.5 que retrata da multiplicação.

Algoritmo 3.5 – Multiplicação

Entrada: Matriz de Hessemberg $\mathbf{H}$ atual, matriz de vetores colunas ortonormais $\mathbf{V}$ e índices $\mathbf{ii}$ e $\mathbf{kk}$ do elemento de $\mathbf{H}$ a ser utilizado

Saída: Vetor auxiliar resultante desta multiplicação ($\mathbf{valaux}$ e $\mathbf{indaux}$)

```

1: Início
2:    $nzaux \leftarrow 0$ 
3:   Para  $j \leftarrow 1$  Ate  $n$  Faça
4:      $aux \leftarrow \mathbf{H}[\mathbf{ii}][\mathbf{kk}] * \mathbf{V}[j][\mathbf{ii}]$ 
5:     Se ( $aux \neq 0$ ) Então
6:        $nzaux \leftarrow nzaux + 1$ 
7:        $\mathbf{valaux}[nzaux] \leftarrow aux$ 
8:        $\mathbf{indaux}[nzaux] \leftarrow j$ 
9:     Fim_Se
10:  Fim_Para
11: Fim

```

Nessa multiplicação existem dois parâmetros (ii e kk) que são índices de um elemento da matriz H que será como escalar para multiplicar com um vetor coluna da matriz V (linha 5). Com esse vetor auxiliar (**valaux**) calculado é possível agora realizar a subtração. Baseado no Algoritmo 3.1 pode-se realizar a subtração conforme o Algoritmo 3.6.

Algoritmo 3.6 – Subtração

Entrada: Vetores de valores **valaux** e **valw**, vetores de índices correspondentes **indaux** e **indw** e tamanhos **nzaux** e **nzw**

Saída: Vetores **valw** e **indw** após a soma e o novo tamanho **nzw**

```

1: Início
2:   Para  $j \leftarrow 1$  Até  $nzaux$  Faça
3:      $wt[indaux[j]] \leftarrow -valaux[j]$ 
4:   Fim_Para

5:   Para  $j \leftarrow 1$  Até  $nzw$  Faça
6:      $i \leftarrow indw[j]$ 
7:     Se ( $wt[i] \neq 0$ ) Então
8:        $valw[j] \leftarrow valw[j] + wt[i]$ 
9:        $wt[i] \leftarrow 0$ 
10:    Fim_Se
11:  Fim_Para

12:  Para  $j \leftarrow 1$  Até  $nzaux$  Faça
13:     $i \leftarrow indaux[j]$ 
14:    Se ( $wt[i] \neq 0$ ) Então
15:       $nzw \leftarrow nzw + 1$ 
16:       $valw[nzw] \leftarrow wt[i]$ 
17:       $indw[nzw] \leftarrow i$ 
18:       $wt[i] \leftarrow 0$ 
19:    Fim_Se
20:  Fim_Para
21: Fim

```

$\left. \begin{array}{l} A1 \\ A2 \end{array} \right\} A$

$\left. \begin{array}{l} B1 \\ B2 \end{array} \right\} B$

A diferença desse algoritmo para o 3.1, além da mudança de variáveis, se encontra na linha 4. Para fazer a subtração basta inverter o sinal na construção de **wt**. Assim, com o sinal invertido, quando o algoritmo for realizar as somas, o valor do vetor de trabalho será retirado de **w** ao invés de ser acrescentado.

Voltando ao algoritmo do método de Arnoldi (Algoritmo 3.2) o próximo uso do vetor **w** está agora na linha 10, onde é calculada sua norma. Observe o Algoritmo 3.7.

Algoritmo 3.7 – Norma de Um Vetor

Entrada: Vetor w ($valw$)
Saída: Norma do vetor w

```

1: Início
2:    $aux \leftarrow 0$ 
3:   Para  $t \leftarrow 1$  Até  $nzw$  Faça
4:      $aux \leftarrow aux + (valw[t] * valw[t])$ 
5:   Fim_Para
6:    $aux \leftarrow \sqrt{aux}$ 
7:   Retorne  $aux$ 
8: Fim

```

Na norma há um laço, indo de 1 até nzw (equivalente a última posição do vetor), onde os quadrados dos elementos de w ($valw$) são somados para depois ser feita a raiz dessa soma.

Assim, o vetor w já está armazenado na estrutura de dados adequada e também as rotinas que utilizam o mesmo modificadas para utilizar das melhores características dessa estrutura.

3.3.2 Complexidade de Vetores Esparsos no GMRES(m)

Para analisar a complexidade envolvida no uso de tal estrutura, foi escolhido seu cálculo mais custoso: a subtração de dois vetores. Observe que, no Algoritmo 3.6, blocos de comandos foram divididos da seguinte maneira: o bloco A referente ao laço da linha 6, que faz as atualizações das posições não nulas, e o bloco B referente ao laço da linha 15, que realiza o preenchimento de posições nulas. Além disso, cada bloco também está subdividido em outros blocos (A1, A2, B1 e B2). Assim, primeiro será analisada a complexidade em cada sub-bloco desse algoritmo.

Nos blocos A1 e B1 ocorrem as mesmas operações. Em ambos, há uma atribuição e uma comparação sendo feitas. Lembrando que os mesmos estão dentro de laços que em A1 se repete nzw vezes e B1 $nzaux$ vezes. Logo, suas complexidades são dadas da seguinte forma:

$$\begin{aligned}
 T(n)_{A1} &= \underset{\text{atribuições}}{nzw} + \underset{\text{comparações}}{nzw} = 2nzw \\
 T(n)_{B1} &= \underset{\text{atribuições}}{nzaux} + \underset{\text{comparações}}{nzaux} = 2nzaux
 \end{aligned} \tag{3.6}$$

Observe agora no bloco A2 a presença de 3 operações: duas atribuições e uma soma, sendo que estas também estão envolvidas no mesmo laço de A1. Para que essas operações sejam

realizadas, a condicional da linha 9 deve ser verdadeira. Neste caso serão analisadas as situações de melhor e pior caso.

Para o melhor caso, todos os elementos de **valw** serão nulos nas posições em que em **valaux** são diferentes de zeros, ou seja, a condicional ser falsa e, portanto, não realizará o bloco A2. Assim $T(n)_{A2_{melhor}} = 0$.

No pior caso, todos esses elementos de **valw** seriam não nulos e, portanto, essas operações seriam realizadas nzw vezes. Ou seja:

$$T(n)_{A2_{pior}} = \underbrace{2nzw}_{atribuições} + \underbrace{nzw}_{soma} = 3nzw \quad (3.7)$$

Por último há o bloco B2. Nele são realizadas 5 operações básicas: 4 atribuições e uma soma. Note que novamente há a presença de uma condicional (linha 18).

No melhor caso, seria a situação em que todas as posições não-nulas em **valaux** também sejam não nulas em **valw**, de forma que não haja preenchimento, ou seja, é o caso onde o bloco B2 não será executado nenhuma vez. Logo, $T(n)_{B2_{melhor}} = 0$.

No pior caso, todas as posições em **valw** que antes eram nulas agora seriam atualizadas, gerando o maior preenchimento possível, que pode ser descrito por $n - nzw$. Portanto, a complexidade deste bloco pode ser descrita por

$$T(n)_{B2_{pior}} = \underbrace{4(n - nzw)}_{atribuições} + \underbrace{(n - nzw)}_{soma} = 5(n - nzw) \quad (3.8)$$

Com a complexidade de cada bloco já definida pode-se agora analisar a complexidade geral desse algoritmo. Lembrando que, como nos blocos A2 e B2 acontecem as situações de melhor e pior caso, então a complexidade geral também deverá ser analisada em situações de melhor e pior caso.

Para o melhor caso, os blocos A2 e B2 não serão executados, então basta somar as complexidades de A1 e B1:

$$\begin{aligned}
T(n)_{\text{melhor}} &= T(n)_{A1} + T(n)_{B1} + T(n)_{A2_{\text{melhor}}} + T(n)_{B2_{\text{melhor}}} \\
T(n)_{\text{melhor}} &= 2nzw + \underbrace{2nzaux}_{B1} + \underbrace{0}_{A2} + \underbrace{0}_{B2} \\
T(n)_{\text{melhor}} &= 2(nzw + nzaux)
\end{aligned} \tag{3.9}$$

Olhando agora para o pior caso, tem-se:

$$\begin{aligned}
T(n)_{\text{pior}} &= T(n)_{A1} + T(n)_{B1} + T(n)_{A2_{\text{pior}}} + T(n)_{B2_{\text{pior}}} \\
T(n)_{\text{pior}} &= 2nzw + \underbrace{2nzaux}_{B1} + \underbrace{3nzw}_{A2} + \underbrace{5(n - nzw)}_{B2} \\
T(n)_{\text{pior}} &= \underbrace{5nzw - 5nzw}_0 + 2nzaux + 5n \\
T(n)_{\text{pior}} &= 2nzaux + 5n
\end{aligned} \tag{3.10}$$

Levando em conta que numa estrutura tradicional (indexada) a complexidade $T(n)$ da subtração de vetores, tanto no melhor quanto no pior caso, é definida por $T(n) = 2n$, é possível tirar algumas conclusões.

Para o pior caso, a estrutura tradicional oferece um melhor desempenho para a subtração, já que a estrutura de vetores esparsos realiza operações a mais. Porém, no melhor caso, desde que a soma de nzw e $nzaux$ não seja maior que n , a nova estrutura será mais eficiente que a tradicional. Vale citar também que, no pior caso o consumo de memória será menor para esta estrutura em relação a indexada.

Após compreender como se dá a complexidade dessa estrutura foi observado, no Capítulo 06, os resultados mostrando na prática como foi o desempenho desta técnica para os problemas dos repositórios e dos geradores.

Capítulo 4

Pré-condicionadores e Refinamento de Solução

Neste capítulo serão vistas duas diferentes técnicas: pré-condicionamento e refinamento de solução, que podem ajudar na convergência do método e na melhora da qualidade da solução encontrada.

Com o pré-condicionamento é possível melhorar o número de condição da matriz e consequentemente obter uma solução mais próxima da exata e algumas vezes até conseguir a convergência da solução do problema conforme foi observado por Silva (2017).

4.1 Pré-condicionamento

Pode-se considerar o pré-condicionamento como qualquer modificação realizada no sistema original de modo que o novo sistema seja equivalente ao anterior, com mesma solução, porém mais fácil de se resolver.

Algumas propriedades de uma matriz como: simetria, definida positiva, número de condicionamento e dominância diagonal, tem grande relevância na convergência e no desempenho do método utilizado para resolver o sistema associado.

Portanto, quando se pré-condiciona a matriz original através de uma matriz de transformação ou de pré-condicionamento, estas propriedades podem surgir no sistema linear transformado, conforme discutido por Carvalho, Fortes e Mangiavacchi (2008), Paz (2012), Saad (2003) e Medeiros (2014).

Assim, o uso de técnicas de pré-condicionamento pode tornar a convergência do método bem mais rápida com um número muito baixo de iterações em função da dimensão do sistema linear.

Um ponto importante a ser observado é na escolha do pré-condicionador a ser utilizado. Pessanha et al. (2012) relatam que os pré-condicionadores podem ser separados em dois grupos distintos: o primeiro são os pré-condicionadores mais específicos, feitos sob medida para uma classe de problema em particular. Porém, ele só será necessário para aqueles

problemas e são sensíveis a qualquer variação. O outro grupo é composto pelos pré-condicionadores feitos para uso geral. Podem abranger problemas de diversas naturezas, porém, se será eficiente ou não, dependerá de diversos fatores.

A ideia do pré-condicionamento é muito simples, no entanto a escolha deve ser feita de forma que o sistema pré-condicionado seja fácil de ser resolvido e o pré-condicionador seja também fácil de construir e aplicar (Benzi, 2002). Ou seja, após o sistema ser pré-condicionado, o método deve ser rápido ao convergir e cada iteração não deve possuir um alto custo computacional.

O pré-condicionamento implica em alterar a matriz dos coeficientes do sistema, fazendo com que os autovalores da matriz estejam mais próximos e, por consequência, o sistema mais estável, reduzindo o número de iterações necessárias para resolver o problema.

No geral, pré-condicionadores mais sofisticados e robustos apresentam melhores resultados em termos convergência e qualidade da solução.

Existem três formas possíveis de realizar o pré-condicionamento (Saad, 2003): o pré-condicionamento pela esquerda ($\mathbf{M}^{-1}\mathbf{Ax} = \mathbf{M}^{-1}\mathbf{b}$), pré-condicionamento pela direita ($\mathbf{AM}^{-1}\mathbf{u} = \mathbf{b}$, $\mathbf{x} \equiv \mathbf{M}^{-1}\mathbf{u}$) e também a combinação dos dois anteriores.

Vale destacar que ambas as formas nos permitem obter sistemas equivalentes e desempenho semelhantes quanto ao método, porém, dentre eles o pré-condicionamento pela esquerda é o mais usado por ser simples e também por ser facilmente integrado em métodos de resolução de sistemas lineares (Poma, 2010).

No presente trabalho observou-se o comportamento do método GMRES(m) após a aplicação de alguns pré-condicionadores. No Algoritmo 4.1 é apresentada a versão do método GMRES reinicializado utilizando-se o pré-condicionamento a esquerda.

Algoritmo 4.1 – GMRES(m) Pré-condicionado

Entrada: Matriz de coeficientes $\mathbf{A}$, vetor independente $\mathbf{b}$ e matriz pré-condicionadora $\mathbf{M}$

Saída: Vetor solução $\mathbf{x}$

Inicialização: Escolha $\mathbf{x}^0 \leftarrow 0$, $it \leftarrow 0$ e a dimensão de m
 Calcule $\mathbf{r}^0$, β e $\mathbf{v}_1$

```

1: Início
2:   Faça
3:      $it \leftarrow it + 1$ 
4:     Para  $j \leftarrow 1$  Até  $m$  Faça //Passo 3: Processo de Arnoldi();
5:       Calcule  $\mathbf{z}^j \leftarrow \mathbf{M}^{-1} \mathbf{v}^j$ 
6:       Calcule  $\mathbf{w} \leftarrow \mathbf{A} \mathbf{z}^j$ 
7:       Para  $i \leftarrow 1$  Até  $j$  Faça
8:          $h_{i,j} \leftarrow \langle \mathbf{w}, \mathbf{v}^i \rangle$ 
9:          $\mathbf{w} \leftarrow \mathbf{w} - h_{i,j} \mathbf{v}^i$ 
10:      Fim_Para
11:       $h_{j+1,j} \leftarrow \|\mathbf{w}\|_2$ 
12:      Se ( $h_{j+1,j} < \text{tol1}$ ) Então
13:        faça  $m = j$  e vá ao Passo 4
14:      Senão
15:         $\mathbf{v}^{j+1} \leftarrow \mathbf{w} / h_{j+1,j}$ 
16:      Fim_Se
17:    Fim_Para
18:    //Passo 4: Decomposicao_QR();
19:    Calcule  $\mathbf{y}_m$  em que  $\mathbf{y}_m \leftarrow \min_y \|\beta \mathbf{e}_1 - \mathbf{H}_m \mathbf{y}\|_2$ ,  $\mathbf{e}_1 \leftarrow [1, 0, \dots, 0]^T$ 
20:    //Passo 5: Atualizacao();
21:    Calcule  $\mathbf{x}^m \leftarrow \mathbf{x}^0 + \mathbf{M}^{-1} \mathbf{V}_m \mathbf{y}_m$ , em que  $\mathbf{V}_m \leftarrow [\mathbf{v}^1, \mathbf{v}^2, \dots, \mathbf{v}^m]$ 
22:    //Passo 6: Calcula Resíduo();
23:    Se ( $\beta \leftarrow \|\mathbf{r}\|_2 < \text{tol2}$ ) Então
24:      Pare.
25:    Senão
26:       $\mathbf{v}^i \leftarrow \mathbf{r} / \beta$ 
27:    Fim_Se
28:  Enquanto ( $it < \text{MAX}$ )
29: Fim

```

Neste algoritmo o pré-condicionamento modifica somente duas operações em relação ao Algoritmo 2.4. Primeiro no Processo de Arnoldi, utiliza-se a inversa da matriz pré-condicionadora no cálculo de $\mathbf{w}$: $\mathbf{w} = \mathbf{A} \mathbf{M}^{-1} \mathbf{v}^j$. A outra modificação é na atualização da solução com o uso novamente da matriz $\mathbf{M}$ invertida: $\mathbf{x}^m = \mathbf{x}^0 + \mathbf{M}^{-1} \mathbf{V}_m \mathbf{y}_m$. Para mais detalhes sobre o método GMRES(m) com pré-condicionamento consulte Silva (2017).

Na próxima seção, os pré-condicionadores abordados neste trabalho serão apresentados. Alguns deles já foram usados em Silva (2017), enquanto outros tratam-se de novos pré-condicionadores que foram propostos na tentativa de obter melhores resultados.

4.2 Pré-condicionadores

A seguir serão apresentados os pré-condicionadores que foram utilizados no presente trabalho, destacando-se que alguns deles foram propostos e outros foram modificados a partir de sua versão original. Por último será apresentada também a técnica de refinamento de solução com a qual tentou-se melhorar a qualidade da solução de alguns problemas.

4.2.1 Pré-condicionador Lumped Modificado

Silva e Sousa (2018) trabalharam com o pré-condicionador Lumped que foi apresentado também por Carvalho, Fortes e Mangiavacchi (2008). Este método de pré-condicionamento modificado é uma variante do Lumped original e semelhante ao pré-condicionador Diagonal (Silva, 2017; Silva e Sousa, 2018; Carvalho, Fortes e Mangiavacchi, 2008), sendo simples de construir e de aplicar.

Originalmente no pré-condicionador Lumped, a matriz $\mathbf{M}$ (pré-condicionadora) é uma matriz diagonal em que cada posição não nula é exatamente a soma dos elementos da linha correspondente à matriz original. Logo, os elementos de $\mathbf{M}$ são dados da seguinte forma:

$$m_{ii} = \sum_{j=1}^n a_{ij}, \quad i = 1, 2, \dots, n \quad (4.1)$$

Para os casos onde a matriz principal $\mathbf{A}$ é simétrica, o somatório dos elementos da linha corresponde também ao somatório dos elementos das colunas.

Assim, uma variante simples para este pré-condicionador é utilizar a soma das colunas, dado da seguinte maneira:

$$m_{jj} = \sum_{i=1}^n a_{ij}, \quad j = 1, 2, \dots, n \quad (4.2)$$

Uma dificuldade que pode surgir é quando a soma é nula. Como o pré-condicionador utiliza a inversa da matriz $\mathbf{M}$, o fato de se ter um elemento nulo na diagonal de $\mathbf{M}$, torna impossível de encontrar a sua inversa. Para contornar este problema quando esta situação ocorre, foi proposto a soma utilizando o módulo dos elementos.

Com essa alteração é possível ampliar a quantidade de problemas nos quais este pré-condicionador pode ser utilizado, e possivelmente obter melhores resultados do que em alguns casos com a versão original.

4.2.2 Pré-condicionador Escala Simétrica

Campos (1999) e Pachás (2009) utilizaram o pré-condicionador Escala Simétrica proposto por Jennings e Malik, em 1978, em que o sistema de equações lineares original $\mathbf{Ax} = \mathbf{b}$ é transformado num sistema equivalente para o qual se aplica o método de resolução e, em seguida, é feita a substituição para encontrar a solução do sistema original.

No trabalho de Silva e Sousa (2019) observa-se que o pré-condicionador Escala Simétrica reduziu o número de iterações para o GMRES(m) em um tempo computacional bem razoável, principalmente para os problemas maiores com baixo número de iterações.

O primeiro passo para construir esse novo sistema é construir uma matriz diagonal $\mathbf{D}$, descrita por:

$$d_{ii} = \frac{1}{\sqrt{a_{ii}}}, \quad i = 1, 2, \dots, n \quad (4.3)$$

Com esta matriz é possível reformular o novo sistema seguindo os passos abaixo:

$$\mathbf{A}' \mathbf{x}' = \mathbf{b}' \quad (4.4)$$

onde

$$\mathbf{A}' = \mathbf{DAD} \quad (4.5)$$

$$\mathbf{x}' = \mathbf{D}^{-1} \mathbf{x} \quad (4.6)$$

$$\mathbf{b}' = \mathbf{Db} \quad (4.7)$$

o que equivale a multiplicar a matriz $\mathbf{D}$ à esquerda do sistema original. Fazendo as multiplicações (4.5) e (4.7) de forma algébrica:

$$\mathbf{A}' = \begin{bmatrix} (d_1)^2 a_{11} & d_1 d_2 a_{12} & \cdots & d_1 d_n a_{1n} \\ d_2 d_1 a_{21} & (d_2)^2 a_{22} & \cdots & d_2 d_n a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_n d_1 a_{n1} & d_n d_2 a_{n2} & \cdots & (d_n)^2 a_{nn} \end{bmatrix} \quad (4.8)$$

$$\mathbf{b}' = \begin{bmatrix} b_1 d_1 \\ b_2 d_2 \\ \vdots \\ b_n d_n \end{bmatrix} \quad (4.9)$$

Primeiro, veja que, se o elemento da diagonal da matriz original for nulo, não será possível construir o novo sistema e, conseqüentemente não se pode utilizar esse pré-condicionador para um problema com esta situação. No Algoritmo 4.2 é retratado o processo de construção desse novo sistema:

Algoritmo 4.2 – Construção do Novo Sistema

Entrada: Matriz de coeficientes $\mathbf{A}$ e vetor independente $\mathbf{b}$

Saída: Matriz $\mathbf{A}$ e vetor $\mathbf{b}$ transformados

```

1: Início
2:   Para  $i \leftarrow 1$  Até  $n$  Faça
3:      $d[i] \leftarrow \frac{1}{\sqrt{a[i][i]}}$  // caso  $a_{ii}$  seja zero, impossível de
                               // aplicar
4:   Fim_Para
5:   Para  $i \leftarrow 1$  Até  $n$  Faça
6:     Para  $j \leftarrow 1$  Até  $n$  Faça
7:        $a[i][j] \leftarrow d[i] * d[j] * a[i][j]$ 
8:     Fim_Para
9:      $b[i] \leftarrow b[i] * d[i]$ 
10:  Fim_Para
11: Fim

```

Perceba os dois laços externo: o primeiro dedicado a construir somente a matriz $\mathbf{D}$. Já o segundo, fazendo uso de $\mathbf{D}$, após ser totalmente construída, para transformar o sistema original no novo sistema.

Note, ainda, que apesar de $\mathbf{D}$ ser uma matriz, nos dois algoritmos acima ela foi representada por um vetor ($\mathbf{d}$). Desde que ela possua a característica de ser diagonal pode-se armazená-la em um vetor, economizando memória (mesmo se fosse usado a estrutura CSR ainda teria um custo/gasto maior de memória do que usar um simples vetor).

Por último vale lembrar que a solução obtida será correspondente a $\mathbf{x}'$. Para obter $\mathbf{x}$ isso é necessária uma rápida e simples transformação da equação (4.6) descrita por:

$$\mathbf{x} = \mathbf{D}\mathbf{x}' \quad (4.10)$$

Um problema muito evidente presente no pré-condicionador Escala Simétrica é o fato de possuir uma forte dependência dos elementos da diagonal da matriz $\mathbf{A}$. Como visto, esse pré-condicionador transforma o sistema original e para isso ele precisa construir a matriz $\mathbf{D}$ seguindo a Equação (4.3).

Observe que há duas situações que impedem o uso do pré-condicionador Escala Simétrica. A primeira refere-se ao sinal negativo dos elementos da diagonal. O segundo impedimento é quando se tem elementos nulos na diagonal. Na construção de $\mathbf{D}$ tem uma divisão, e justamente o seu denominador é o elemento da diagonal de $\mathbf{A}$. Logo, neste caso não se pode utilizar também o referido pré-condicionador.

Portanto, diante destes dois cenários, foram propostas modificações muito simples para este pré-condicionador que serão apresentadas nas próximas duas seções.

4.2.2.1 Pré-condicionador Escala Simétrica – Variante 1

Considerando a primeira situação que impede a utilização do Escala Simétrica, uma alternativa muito simples é a não utilização da raiz quadrada no cálculo dos elementos da diagonal para a matriz $\mathbf{D}$, chamada de variante 1 do Escala Simétrica, dado por:

$$d_{ii} = \frac{1}{a_{ii}}, i = 1, 2, \dots, n \quad (4.11)$$

Cabe ressaltar que, desta forma, há uma maior aplicabilidade do pré-condicionador, pois, como relatado em Silva (2017), existem diversos problemas do repositório MatrixMarket com elementos negativos na diagonal. Além disso, tem-se uma redução de n operações envolvendo raiz quadrada na construção da matriz diagonal do pré-condicionador.

4.2.2.2 Pré-condicionador Escala Simétrica – Variante 2

Com relação a segunda situação (elementos nulos na diagonal) propôs-se uma adaptação simples no algoritmo original.

A modificação consiste em somar os módulos dos elementos da linha correspondente e em seguida aplicar a raiz quadrada. Logo, o cálculo de d_i é dado por:

$$d = \begin{bmatrix} \frac{1}{\sqrt{(|a_{1,i}| + |a_{1,i+1}| + \dots + |a_{1,n}|)}} \\ \frac{1}{\sqrt{(|a_{2,i}| + |a_{2,i+1}| + \dots + |a_{2,n}|)}} \\ \vdots \\ \frac{1}{\sqrt{(|a_{n,i}| + |a_{n,i+1}| + \dots + |a_{n,n}|)}} \end{bmatrix}, i = 1, 2, \dots, n \quad (4.12)$$

O Algoritmo 4.3, a seguir, aborda esta alteração e faz a construção do novo sistema.

Algoritmo 4.3 – Construção do Novo Sistema: Utilizando Soma das Linhas

Entrada: Matriz de coeficientes **A** (**val**, **cind** e **rptra**) e vetor independente **b**

Saída: Matriz **A** (**val**, **cind** e **rptra**) e vetor **b** transformados

```

1: Início
2:   Para  $j \leftarrow 1$  Até  $n$  Faça
3:      $k1 \leftarrow rptra[j]$ 
4:      $k2 \leftarrow rptra[j+1]-1$ 
5:      $soma \leftarrow 0$ 
6:     Para  $i \leftarrow k1$  Até  $k2$  Faça
7:        $soma \leftarrow soma + fabs(val[i])$ 
8:     Fim_Para
9:      $d[j] \leftarrow \frac{1}{\sqrt{soma}}$ 
10:  Fim_Para
11:  Para  $j \leftarrow 1$  Até  $n$  Faça
12:     $k1 \leftarrow rptra[j]$ 
13:     $k2 \leftarrow rptra[j+1]-1$ 
14:    Para  $i \leftarrow k1$  Até  $k2$  Faça
15:       $val2[i] \leftarrow d[j] * d[cind[i]] * val[i]$ 
16:    Fim_Para
17:     $b2[j] \leftarrow b[j] * d[j]$ 
18:  Fim_Para
19: Fim

```

Note que agora foi feito um somatório para cada linha com o módulo de todos os seus elementos.

4.2.3 Pré-condicionador ILU(0)

A partir de métodos de fatoração de matrizes é possível obter pré-condicionadores eficazes, definindo parâmetros para esse processo. Assim, estabelecendo a intensidade do preenchimento de elementos não nulos em posições que antes eram nulas (*fill-in*) como um parâmetro é possível obter diferentes variações de um mesmo pré-condicionador. Dentre esses métodos, a Fatoração LU é uma decomposição que consegue tornar um pré-condicionador eficiente para determinados problemas (Paz, 2012).

A Fatoração Incompleta LU com zero de preenchimento (ou seja, aquela em que todas as posições nulas na matriz original são mantidas durante a fatoração), ou apenas ILU(0), implementada por Silva (2017), apresenta pontos que necessitam de mais atenção, dos quais dois se destacam: a separação da matriz $\mathbf{M}$ (matriz resultante dessa fatoração) em $\mathbf{L}$ e $\mathbf{U}$, e uso excessivo da rotina de busca de elementos na estrutura CSR.

Na rotina implementada por Silva (2017), baseada em Medeiros (2014) e Pires (2014), depois de gerar a matriz $\mathbf{M}$ (os elementos da diagonal e acima dela pertencem a $\mathbf{U}$ e abaixo da diagonal fazem parte da matriz $\mathbf{L}$) as matrizes triangulares $\mathbf{L}$ e $\mathbf{U}$ eram separadas em duas estruturas CSR diferentes, ocasionando em um desperdício de processamento para separá-las (em problemas pequenos não é tão visível, mas ao tratar problemas de grande porte essa separação faz uma diferença considerável).

Além disso, tanto na fatoração como nos processos da substituição progressiva e regressiva (métodos de resolução de sistemas triangulares superiores e inferiores) usou-se excessivamente uma função de busca de elementos em uma estrutura CSR, de tal forma que a característica principal da estrutura (eficiência ao reduzir cálculos com elementos nulos) foi desperdiçada. Assim, o que deveria ser mais eficiente com o uso dessa estrutura de dados, tinha um desempenho muito próximo de uma estrutura indexada tradicional.

Com isso foram estabelecidos dois passos: Primeiro, ao invés de dividir a matriz $\mathbf{M}$ em duas estruturas separadas, será utilizada diretamente a matriz $\mathbf{M}$ para os cálculos. Depois, deve-se revisar as rotinas que utilizam a busca em estrutura CSR para evitar o uso excessivo da mesma.

Lugon (2013) alcançou estes objetivos ao utilizar apenas uma estrutura de dados compacta para armazenar e usar apenas a matriz $\mathbf{M}$. Portanto, tendo como referência o trabalho de Lugon (2013), propõe-se aqui a Fatoração ILU(0) utilizando-se da estrutura de dados CSR armazenada como somente uma matriz e está representada no Algoritmo 4.4 a seguir.

Algoritmo 4.4 – ILU(0)

Entrada: Matriz de coeficientes $\mathbf{A}$ (*val*, *cind* e *rp**tr*)

Saída: Matriz fatorada (*valilu*)

```

1: Início
2:   valilu ← val
3:   Para i ← 2 Até n Faça
4:     Para k ← rptr[i] Até (i! = cind[k]) Faça
5:       valilu[k] ←  $\frac{\text{valilu}[k]}{\text{valilu}[\text{busca}(k,k)]}$ 
6:       Para j ← k+1 Até (rptr[i+1]−1) Faça
7:         valilu[j] ← valilu[j] − (valilu[k] * valilu[busca(cind[k], cind[j])])
8:       Fim_Para
9:     Fim_Para
10:  Fim_Para
11: Fim

```

Observa-se que não é mais preciso declarar duas estruturas CSR separadas (uma para $\mathbf{L}$ e outra para a matriz $\mathbf{U}$). Com apenas um vetor (*valilu*) é possível, agora, armazenar $\mathbf{M}$ no formato CSR, aproveitando os vetores *cind* e *rp**tr* que permanecem os mesmos do sistema original. Observe agora como se dá o consumo de memória em três situações distintas: estrutura principal sem pré-condicionamento, utilizando o ILU(0) com duas estruturas CSR e utilizando o ILU(0) com apenas uma estrutura CSR.

Conforme visto no Capítulo 3, considere que uma variável real ocupe 8 bytes e uma inteira ocupe 4 bytes. Quando não há o pré-condicionamento a estrutura principal consiste dos três vetores para CSR, assim tem-se: o vetor *val* utilizará $8nnz$ bytes (*nnz* é o número de elementos não nulos da matriz $\mathbf{A}$), o vetor *cind* utilizará $4nnz$ bytes e o vetor *rp**tr* utilizará $4(n + 1)$ bytes (*n* é a dimensão da matriz $\mathbf{A}$). No total serão $(12nnz + 4n + 4)$ bytes. Supondo um problema com uma dimensão de 1500 e com 2% de densidade, será aproximadamente um gasto de 0,54 Megabytes.

A versão da fatoração ILU(0) utilizada em Silva (2017), além dos três vetores citados anteriormente, também haverá mais duas estruturas CSR, uma para os elementos de $\mathbf{L}$ e outra para os elementos de $\mathbf{U}$: o vetor *val2*, que armazena os elementos não nulos de $\mathbf{M}$ (resultado da fatoração), que necessita de $8nnz$ bytes; *valu*, que armazena os elementos não nulos de $\mathbf{U}$, precisa de $8nnzu$ bytes (*nnzu* é o número de elementos não nulos de $\mathbf{U}$); *vall*, que armazena os

elementos não nulos de L , precisa de $8nnzl$ bytes ($nnzl$ é o número de elementos não nulos de L); **cindu**, que armazena os índices de coluna dos elementos de U , precisa de $4nnzu$ bytes; **cindl**, que armazena os índices de coluna dos elementos de L , precisa de $4nnzl$ bytes; **rptru** e **rptrl**, que armazenam, respectivamente, os ponteiros de linha de U e L , precisam de $4(n + 1)$ bytes cada. No total serão $(20nnz + 12n + 12nnzu + 12nnzl + 12)$ bytes.

Supondo novamente um problema com uma dimensão de 1500, 2% de densidade e supondo que 60% dos elementos não nulos pertencem a U e os outros 40% a L , tem-se aproximadamente um gasto de 1,45 Megabytes, que representa um aumento de quase 2,7 vezes em relação a situação anterior, mostrando assim o tamanho do desperdício de memória.

Por fim, após a alteração na fatoração com base no trabalho de Lugon (2013), houve a necessidade de, além dos três vetores da estrutura CSR referente ao sistema original, somente mais um novo vetor: o vetor **valilu**, que armazena os elementos não nulos de M , que consome $8nnz$ bytes de memória. No total tem-se $(20nnz + 4n + 4)$ bytes. Mais uma vez considerando um problema com uma dimensão de 1500 e 2% de densidade, teve-se aproximadamente um gasto de 0,91 Megabytes. Assim, é possível perceber que a nova mudança trouxe uma economia considerável na quantidade de memória utilizada.

Note ainda que, no Algoritmo 4.4, a rotina de busca para estrutura CSR foi utilizada somente em dois momentos, tornando-o mais eficiente computacionalmente, ao contrário de Silva (2017), onde todos elementos eram localizados pela busca.

Destaca-se também que Lugon (2013) apresentou uma rotina para a ILU(p) (Fatoração Incompleta LU que permite diferentes níveis de preenchimentos), porém, como ele mesmo coloca, é de difícil implementação e com código extenso, além de não ser adequado para usar com a estrutura de dados CSR, já que necessitaria de atualiza-la sempre que houvesse preenchimento, elevando o custo computacional de forma muito intensa.

A seguir são apresentados os algoritmos para as substituições progressiva e regressiva (ou sucessiva e retroativa).

Algoritmo 4.5 – Substituição Progressiva

Entrada: Matriz A fatorada ($valilu$, $cind$ e $rptra$) e vetor independente b

Saída: Vetor solução x

```

1: Início
2:    $x[1] \leftarrow \frac{b[1]}{valilu[1]}$ 
3:   Para  $i \leftarrow 2$  Até  $n$  Faça
4:      $soma \leftarrow b[i]$ 
5:      $k1 \leftarrow rptra[i]$ 
6:      $k2 \leftarrow rptra[j+1]-1$ 
7:     Para  $j \leftarrow k1$  Até  $k2$  Faça
8:       Se ( $cind[j] < i$ ) Então
9:          $soma \leftarrow soma - (valilu[j] * x[cind[j]])$ 
10:      Fim_Se
11:    Fim_Para
12:     $x[i] \leftarrow soma$ 
13:  Fim_Para
14: Fim

```

Algoritmo 4.6 – Substituição Regressiva

Entrada: Matriz A fatorada ($valilu$, $cind$ e $rptra$) e vetor independente b

Saída: Vetor solução x

```

1: Início
2:    $x[n] \leftarrow \frac{b[n]}{valilu[nz]}$ 
3:   Para  $i \leftarrow (n-1)$  Até 1 Faça
4:      $soma \leftarrow b[i]$ 
5:      $k1 \leftarrow rptra[i]$ 
6:      $k2 \leftarrow rptra[j+1]-1$ 
7:     Para  $j \leftarrow k1$  Até  $k2$  Faça
8:       Se ( $cind[j] > i$ ) Então
9:          $soma \leftarrow soma - (valilu[j] * x[cind[j]])$ 
10:      Fim_Se
11:    Fim_Para
12:     $x[i] \leftarrow soma$ 
13:  Fim_Para
14: Fim

```

A principal vantagem desses algoritmos é que eles permitem que as substituições sejam realizadas sem que separe a matriz M nas matrizes L e U , podendo, assim, usar diretamente a matriz M (saída do algoritmo ILU(0)).

Para isso, as substituições progressiva e regressiva utilizam de uma condição para identificar a qual matriz (L ou U) o elemento m_{ij} pertence. Se o índice de coluna do elemento

for menor que o índice de linha, quer dizer que ele pertence a matriz L e, portanto, será utilizado na substituição progressiva. Caso contrário, se o índice de coluna for maior que o de linha, então ele pertence a matriz U , sendo utilizado na substituição regressiva.

Outro ponto positivo dessas rotinas foi o aproveitamento direto da estrutura de dados CSR, evitando cálculos desnecessários reduzindo, portanto, o custo computacional.

4.2.4 Pré-condicionador Diagonal/Escala Simétrica

Os trabalhos de Silva e Sousa (2018) e Silva e Sousa (2019) abordam 4 pré-condicionadores: dois mais simples, Diagonal e Lumped, e dois mais complexos, Escala Simétrica e ILU(0). Propõem-se, aqui, então, combinar os pré-condicionadores Diagonal e Escala Simétrica.

De acordo com Silva e Sousa (2019) não há um pré-condicionador que tenha um ótimo desempenho para todos os problemas. Assim, o que pode ser feito é selecionar aqueles que tiveram vantagens em pontos que são relevantes para um estudo.

Inicialmente, o pré-condicionador Diagonal foi escolhido, pois sua construção é simples e apresenta bons resultados (Silva e Sousa, 2018), com a restrição de não ser aplicável em alguns casos.

O pré-condicionador Diagonal (Pini, G. e Gambolati, 1990), conhecido também por Jacobi, é considerado um dos mais simples de implementar com um custo muito baixo. Silva e Sousa (2018) destacam a eficiência dele em vários problemas, conseguindo convergências com uma boa qualidade do resíduo.

Para o desenvolvimento do pré-condicionador proposto neste trabalho, considere que o pré-condicionador Diagonal é uma matriz M diagonal definida por:

$$m_{ij} = \begin{cases} a_{ij}, & \text{se } i = j, \\ 0, & \text{se } i \neq j. \end{cases} \quad (4.13)$$

O outro pré-condicionador considerado é o Escala Simétrica, já descrito na Seção 4.2.2. Ele utiliza uma matriz diagonal, definida por (4.3) para transformar o sistema original. Ele também mostrou ótimos resultados (Silva e Sousa, 2019), principalmente para problemas grandes.

Perceba que ambos os pré-condicionadores dependem dos elementos da diagonal da matriz do sistema original.

A matriz referente ao pré-condicionador Diagonal, construída pela equação (4.13), será chamada de $\mathbf{M}$, e para a matriz utilizada na transformação do sistema original referente ao pré-condicionador Escala Simétrica, será denotada por $\mathbf{D}$.

Primeiro é preciso fazer o pré-condicionamento no sistema original ($\mathbf{Ax} = \mathbf{b}$) à direita com a matriz $\mathbf{M}$. Logo, tem-se:

$$\mathbf{A}(\mathbf{M})^{-1}(\mathbf{M})\mathbf{x} = \mathbf{b} \quad (4.14)$$

fazendo $\mathbf{x}' = \mathbf{M}\mathbf{x}$ vem que

$$\mathbf{A}(\mathbf{M})^{-1}\mathbf{x}' = \mathbf{b} \quad (4.15)$$

Considerando que $\mathbf{A}' = \mathbf{A}(\mathbf{M})^{-1}$, pode-se reescrever a equação (4.15) de forma a ter:

$$\mathbf{A}'\mathbf{x}' = \mathbf{b} \quad (4.16)$$

Agora, com o pré-condicionador Diagonal aplicado no sistema original, utiliza-se o pré-condicionador Escala Simétrica para transformar o sistema (4.16) antes de aplicar o método de resolução GMRES(m), seguindo os mesmos passos visto na Seção 4.2.2 deste trabalho.

Considerando então o sistema (4.16) e utilizando a matriz $\mathbf{D}$ no pré-condicionamento à esquerda, ocorre que:

$$\mathbf{DA}'\mathbf{DD}^{-1}\mathbf{x}' = \mathbf{Db} \quad (4.17)$$

Fazendo,

$$\mathbf{A}'' = \mathbf{DA}'\mathbf{D}, \quad (4.18)$$

$$\mathbf{x}'' = \mathbf{D}^{-1}\mathbf{x}' \quad (4.19)$$

$$e \mathbf{b}' = \mathbf{D}\mathbf{b} \quad (4.20)$$

Assim,

$$\mathbf{A}''\mathbf{x}'' = \mathbf{b}' \quad (4.21)$$

Neste caso, os elementos da matriz $\mathbf{D}$ são dados por:

$$d_i = \frac{1}{\sqrt{a'_{ii}}}, i = 1, 2, \dots, n \quad (4.22)$$

Observe que o Escala Simétrica está sendo aplicado no sistema obtido em (4.16), o qual já foi aplicado o pré-condicionador Diagonal ao sistema original. Por isso, os elementos utilizados nessa transformação são aqueles obtidos desse novo sistema, e consequentemente, os elementos da diagonal de $\mathbf{D}$ são também do sistema transformado. Neste ponto, aplica-se o método de resolução, no caso o GMRES(m).

Portanto, para obter a solução do sistema original, deve-se passar por duas etapas, partindo de $\mathbf{x}''$ até chegar em $\mathbf{x}$. Assim, de posse da solução $\mathbf{x}''$ em (4.21), utiliza-se (4.19) para obter $\mathbf{x}'$:

$$\mathbf{x}'' = \mathbf{D}^{-1}\mathbf{x}' \Rightarrow \mathbf{x}' = \mathbf{D}\mathbf{x}'' \quad (4.23)$$

Note que, como $\mathbf{D}$ é uma matriz diagonal, é possível armazená-la como um vetor, evitando posições nulas, e agilizando cálculos de (4.23), que passa a ser uma multiplicação entre vetores.

Agora, sabendo que $\mathbf{x}' = \mathbf{M}\mathbf{x}$, é possível definir $\mathbf{x}$ fazendo:

$$\mathbf{x} = (\mathbf{M})^{-1} \mathbf{x}'. \quad (4.24)$$

Logo, obtêm-se a solução para o sistema original a partir de uma solução obtida após todo o processo de pré-condicionamento duplo. Observe que, mais uma vez, há uma matriz

diagonal (que pode ser vista como um vetor) multiplicando-se por um vetor, ou seja, esse cálculo é bem rápido.

Com relação aos algoritmos utilizados, o pré-condicionador Escala Simétrica continua da mesma forma. Modifica-se apenas as variáveis referente ao sistema linear, já que não foi utilizado o original, e sim o sistema modificado com o pré-condicionador Diagonal.

Para aplicação do pré-condicionador Diagonal faz-se previamente a multiplicação de $\mathbf{A}$ por $(\mathbf{M})^{-1}$ (veja 4.15). Note que, apesar de $\mathbf{M}$ ser uma matriz diagonal, é possível representá-la como um vetor, e assim, além de poupar memória, também facilita a multiplicação, pois o que de fato ocorre é a multiplicação de uma matriz por um vetor. No Algoritmo 4.7 é retratada essa multiplicação.

Algoritmo 4.7 – Aplicação do Pré-condicionador Diagonal

Entrada: Matriz de coeficientes $\mathbf{A}$ (**val** e **cind**) e matriz pré-condicionadora $\mathbf{M}$

Saída: Matriz $\mathbf{A}$ pré-condicionada (**val**)

```

1: Início
2:   Para  $i \leftarrow 1$  Até  $nz$  Faça
3:      $\mathbf{val}[i] \leftarrow \mathbf{val}[i] * \mathbf{M}[\mathbf{cind}[i]]$ 
4:   Fim_Para
5: Fim

```

A multiplicação apresentada no Algoritmo 4.7 pode ser descrita por:

$$\mathbf{A}(\mathbf{M})^{-1} = \begin{bmatrix} a_{11}/(m)_1 & a_{12}/(m)_2 & \dots & a_{1n}/(m)_n \\ a_{21}/(m)_1 & a_{22}/(m)_2 & \dots & a_{2n}/(m)_n \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}/(m)_1 & a_{n2}/(m)_2 & \dots & a_{nn}/(m)_n \end{bmatrix} \quad (4.25)$$

Porém, ao usar a estrutura de dados CSR, é necessário o cuidado de saber qual coluna determinado elemento de **val** pertence para identificar o elemento do pré-condicionador Diagonal será multiplicado. Isso é facilmente feito com o vetor **cind** da estrutura compactada, que contém os índices de colunas dos valores de **val**. Assim, tem-se que:

$$\begin{bmatrix} \frac{val[1]}{M[cind[1]]} \\ \frac{val[2]}{M[cind[2]]} \\ \vdots \\ \frac{val[n]}{M[cind[n]]} \end{bmatrix} \quad (4.26)$$

4.2.5 Pré-condicionador Diagonal/ILU(0)

Propõe-se aqui, novamente, um novo pré-condicionador que é a integração de outros dois já existentes. Neste caso, foi utilizado também o pré-condicionador Diagonal juntamente com o ILU(0). Pois este último conseguiu obter soluções de ótima qualidade principalmente para os problemas maiores, além de possuir, na maioria dos casos estudados, um tempo de resolução baixo (Silva e Sousa, 2019).

A Fatoração ILU(0) (comentada na Seção 4.2.3) consiste em manter todas as posições originais com elementos nulos tanto na matriz triangular inferior L quanto na matriz triangular superior U que são geradas após uma fatoração incompleta.

Inicialmente, de maneira semelhante à técnica utilizada na seção anterior, o pré-condicionador Diagonal é aplicado no sistema original de forma a obter o novo sistema como em (4.16). Em seguida, aplica-se o pré-condicionador ILU(0) neste novo sistema.

Como visto na Seção 4.2.3, o pré-condicionamento modifica duas rotinas do método GMRES(m). Na primeira rotina, antes de realizar o cálculo de $\mathbf{w}$ (linha 8 do Algoritmo 4.1), é preciso resolver dois sistemas lineares. O cálculo de $\mathbf{z}$ é realizado de maneira equivalente a uma resolução por decomposição LU tradicional, ou seja, resolve-se um sistema triangular inferior e em seguida um sistema triangular superior para se obter o vetor $\mathbf{z}$. Cabe lembrar que estas resoluções são substituições progressiva e regressiva, respectivamente. Isto é, são facilmente resolvidos.

A outra rotina modificada pelo pré-condicionamento foi a atualização da solução (linha 22 do Algoritmo 4.1) descrita por:

$$\mathbf{x}'_m = \mathbf{x}'_0 + \mathbf{M}^{-1}\mathbf{V}_m\mathbf{y}_m \quad (4.27)$$

Vale lembrar que $\mathbf{x}'$ é a solução pro sistema (4.16) já alterado pelo pré-condicionador Diagonal. Perceba que $\mathbf{M} = \mathbf{LU}$, ou seja, a Equação (4.27) representa um novo conjunto de sistemas linear que será resolvido. Para isso, é necessário, antes, manipular a Equação (4.27). Primeiro isola-se $\mathbf{x}'_m$ e $\mathbf{x}'_0$ de forma a ter:

$$\mathbf{x}'_m - \mathbf{x}'_0 = \mathbf{M}^{-1} \mathbf{V}_m \mathbf{y}_m \quad (4.28)$$

Simplificando, define-se $\mathbf{x}'' = \mathbf{x}'_m - \mathbf{x}'_0$ e assim reescrevendo (4.28), tem-se:

$$\mathbf{x}'' = \mathbf{M}^{-1} \mathbf{V}_m \mathbf{y}_m \quad (4.29)$$

Perceba que neste ponto já se tem um vetor de incógnitas ($\mathbf{x}''$), uma matriz ($\mathbf{M}$) e um vetor independente ($\mathbf{V}_m \mathbf{y}_m$), ou seja, mudando a matriz de lado na Equação (4.29) obtém-se o novo sistema a ser resolvido:

$$\mathbf{M} \mathbf{x}'' = \mathbf{V}_m \mathbf{y}_m \quad (4.30)$$

Por fim, sabendo que $\mathbf{M}$ é o resultado da fatoração, têm-se os seguintes sistemas para resolver:

$$\begin{cases} \mathbf{L} \mathbf{g} = \mathbf{V}_m \mathbf{y}_m \\ \mathbf{U} \mathbf{x}'' = \mathbf{g} \end{cases} \quad (4.31)$$

Porém, sabendo que $\mathbf{x}'' = \mathbf{x}'_m - \mathbf{x}'_0$, isolando $\mathbf{x}'_m$ obtém-se a solução para o Sistema (4.16). Com isso, de maneira análoga a seção anterior, aplica-se essa solução na Equação (4.24) a fim de obter então a solução para o sistema original.

4.2.6 Pré-condicionador P

Na Seção 4.2.2 foi discutido sobre o pré-condicionador Escala Simétrica, o qual transforma o sistema original, através de algumas de manipulações algébricas. O mesmo possui

algumas restrições quanto ao tipo de problema em que pode ser aplicado. Por isso foram apresentadas algumas variações para utilizar com esses problemas.

Tendo agora como ponto de partida o pré-condicionador Escala Simétrica, propõe-se neste trabalho, algumas modificações na versão original para tentar resolver novos problemas e obter um bom desempenho. Este novo pré-condicionador será chamado simplesmente de pré-condicionador **P**.

Partindo do sistema original $\mathbf{Ax} = \mathbf{b}$ define-se a matriz **P** como sendo a matriz diagonal de **A**.

Assim, tem-se:

$$\mathbf{AP}^{-1}\mathbf{Px} = \mathbf{b} \quad (4.32)$$

Fazendo a multiplicação à esquerda em (4.32) com a matriz **P**, vem que:

$$\mathbf{PAP}^{-1}\mathbf{Px} = \mathbf{Pb} \quad (4.33)$$

De maneira análoga ao pré-condicionador Escala Simétrica, fazendo-se

$$\begin{aligned} \mathbf{A}' &= \mathbf{PAP}^{-1} \\ \mathbf{x}' &= \mathbf{Px} \\ \mathbf{b}' &= \mathbf{Pb} \end{aligned} \quad (4.34)$$

o novo sistema será dado por:

$$\mathbf{A}'\mathbf{x}' = \mathbf{b}'. \quad (4.35)$$

Portanto, o método GMRES(*m*) é aplicado neste novo sistema. De posse da solução $\mathbf{x}'$, é necessário obter **x** para o sistema original:

$$\mathbf{Px} = \mathbf{x}' \quad (4.36)$$

Como **P** é invertível, vem que:

$$\mathbf{x} = \mathbf{P}^{-1} \mathbf{x}' \quad (4.37)$$

Com $\mathbf{P}$ sendo uma matriz diagonal é possível armazená-la como um vetor. Assim, o cálculo em (4.37) passa de uma multiplicação de matriz por vetor para uma multiplicação de vetores, o que torna esse processo mais rápido. Logo, a obtenção da solução do sistema original pode ser descrita por:

$$\mathbf{x} = \begin{bmatrix} x'_1 / p_1 \\ x'_2 / p_2 \\ \vdots \\ x'_n / p_n \end{bmatrix} \quad (4.38)$$

Cabe destacar aqui que, da mesma forma que o pré-condicionador Escala Simétrica, o pré-condicionador $\mathbf{P}$ possui a característica muito importante de preservar a esparsidade original. Esta característica se torna importante pois a estrutura de dados CSR, utilizada neste trabalho, se torna muito custosa quando há casos de preenchimento.

Portanto, este pré-condicionador se destaca pela sua fácil implementação e também aplicação em diversos problemas. Este pré-condicionador, assim como os derivados do Escala Simétrica, proporciona a economia de armazenamento envolvendo a matriz $\mathbf{P}$, assim como evita a realização de cálculos desnecessários com elementos nulos.

4.3 Refinamento

Quando se trabalha com métodos de resolução de sistemas lineares é comum, como observado em Silva (2017), casos em que não há a convergência do método ou que se obtenha um resíduo muito alto. Dependendo do problema tratado (principalmente os problemas reais) é necessário obter uma solução mais próxima da exata. Por isso, será apresentada aqui uma técnica que utiliza o resíduo obtido para tentar aproximar ainda mais a solução numérica da exata.

O refinamento de solução (Asano e Colli, 2009) trabalha principalmente com o resíduo calculado pela Equação (2.21). Suponha que se tenha aplicado o método GMRES(m) (vale citar

que o refinamento de solução pode ser aplicado com os resultados obtidos com qualquer método de resolução de sistemas, desde que se obtenha o resíduo no final) para determinado problema, com matriz principal $\mathbf{A}$ e vetor independente $\mathbf{b}$. Após a convergência do método obtém-se a solução final $\mathbf{x}^1$ e também o resíduo final $\mathbf{r}^1$.

Para conseguir um resultado melhor é possível tomar uma nova solução conhecida por $\mathbf{x}^2$ que representa uma melhora em relação à anterior ($\mathbf{x}^1$), descrita por

$$\mathbf{x}^2 = \mathbf{x}^1 + \boldsymbol{\delta}^1 \quad (4.39)$$

onde $\boldsymbol{\delta}^1$ (vetor de correção) representa o quanto $\mathbf{x}^1$ precisa ser acrescentado para alcançar $\mathbf{x}^2$. Com isso, aplicando essa nova solução ao sistema linear tem-se:

$$\mathbf{A}\mathbf{x}^2 = \mathbf{b} \quad (4.40)$$

Utilizando-se (4.39) vem que:

$$\mathbf{A}(\mathbf{x}^1 + \boldsymbol{\delta}^1) = \mathbf{b} \quad (4.41)$$

$$\mathbf{A}\mathbf{x}^1 + \mathbf{A}\boldsymbol{\delta}^1 = \mathbf{b} \quad (4.42)$$

$$\mathbf{A}\boldsymbol{\delta}^1 = \mathbf{b} - \mathbf{A}\mathbf{x}^1 \quad (4.43)$$

Observe que, no lado direito da equação acima, tem-se o resíduo encontrado anteriormente:

$$\mathbf{A}\boldsymbol{\delta}^1 = \mathbf{r}^1 \quad (4.44)$$

Com isso, obtém-se um novo sistema, onde a matriz principal continua a mesma do problema original, o vetor independente representa o resíduo encontrado anteriormente e o vetor de incógnitas é vetor de correção, que sendo utilizado em (4.39) encontra-se uma nova solução para o problema original.

Porém, é possível que um único refinamento não seja o suficiente para obter uma precisão desejada. Então, novamente, calcula-se o resíduo em relação a nova solução encontrada, para então aplicar o refinamento mais uma vez, repetindo esse processo de refinamento até alcançar a precisão esperada.

Para facilitar, pode-se generalizar as Equações (4.39) e (4.44) obtendo:

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \boldsymbol{\delta}^k \quad (4.45)$$

$$\mathbf{A}\boldsymbol{\delta}^k = \mathbf{r}^k \quad (4.46)$$

onde $k = 1, 2, \dots, ITERMAX$ representa as iterações realizadas pelo processo de refinamento.

Como um único refinamento pode não ser suficiente, há a necessidade de se fazer uso de alguns critérios para o término deste processo. O termo *ITERMAX* representa a quantidade máxima de refinamentos a serem feitos, prevenindo situações em que os valores do resíduo obtidos não possuem uma diferença significativa entre cada refinamento, causando estagnação. Os valores propostos neste trabalho variam de acordo com o tamanho do sistema: com problemas de 100 até 1000 variáveis, adota-se 1% de n , de 1000 até 10000 variáveis, será 0,5% de n . Além desses valores, utilizou-se também $n/2$, n , e $2n$ pensando-se em casos que necessitem de quantidades maiores de refinamentos.

Outro critério estabelecido foi em relação ao nível de precisão. No presente trabalho utilizou-se a norma do vetor de resíduo atual ($\mathbf{r}^k$) menor do que uma certa tolerância.

Portanto, estas restrições podem ser representadas por:

- (i) $k > ITERMAX$.
- (ii) $|\mathbf{r}^k| < \varepsilon$, onde $\varepsilon = 1\text{E}-12$ representa a tolerância do critério de parada utilizada;

Para resolver os sistemas gerados a cada iteração do refinamento, o método direto da decomposição LU é uma boa opção, pois a matriz $\mathbf{A}$ permanece a mesma em todas as iterações. Assim, basta realizar a fatoração uma única vez e reaproveitá-la cada iteração.

Porém, como no presente trabalho utilizou-se a estrutura de dados CSR e esta não é adequada para métodos que geram preenchimento, como é o caso da Fatoração LU, serão feitas duas abordagens diferentes: primeiro será utilizado o próprio GMRES(m) e, na segunda, a Fatoração ILU(0), variante da tradicional Fatoração LU.

A primeira forma de realizar o refinamento é utilizando-se o método GMRES(m) para resolver os sistemas gerados por esse processo, o qual está representado no Algoritmo 4.8 a seguir.

Algoritmo 4.8 – Refinamento com GMRES(m)

Entrada: Matriz de coeficientes A e vetor independente b

Saída: Vetor solução x

```

1: Início
2:    $b \leftarrow ro$ 
3:    $it \leftarrow 0$ 
4:   Faça
5:      $it \leftarrow it + 1$ 
6:      $gmres()$ 
7:      $atualizarSolucao()$ 
8:      $residuo()$ 
9:     Se ( $|r| < tol1$ ) Então
10:      Pare
11:    Senão
12:       $b \leftarrow r$  // Atualiza vetor independente
13:    Fim_se
14:  Enquanto ( $it \leq ITERMAX$ )
15: Fim

```

O primeiro passo do refinamento é fazer o vetor independente (b) iniciar recebendo o resíduo original (ro), obtido antes de iniciar o refinamento ao resolver o sistema tratado com determinado método. Perceba que a cada iteração, o método GMRES(m) é aplicado para resolver um sistema diferente gerado pelo refinamento.

Caso a norma do resíduo para a solução obtida no refinamento atenda a tolerância preestabelecida, o algoritmo termina, senão ele atualiza o vetor independente e repete os passos a partir da linha 5. Outra forma de parada do algoritmo é quando o total de iterações (it) do refinamento atinge o máximo de iterações também já preestabelecido pelo $ITERMAX$.

Por meio de experimento preliminares percebeu-se que o uso desse método pode ajudar a alcançar uma qualidade de solução desejada. Apesar do GMRES(m) resolver diversos problemas, pode ocorrer situações em que não consiga obter uma solução, seja por características da própria matriz ou por falta de um valor de m adequado.

Assim, sugere-se também outro método que possa ser utilizado no refinamento. Pensando na estratégia de aproveitar a fatoração única de LU antes dos refinamentos, a Fatoração Incompleta LU com zero de preenchimento (veja a Seção 4.2.3) foi utilizada. Neste caso, agora serão dois sistemas simples de serem resolvidos, pois suas matrizes são triangulares. Nesta fase do refinamento tais matrizes são da seguinte forma:

$$\begin{cases} \mathbf{L}\mathbf{y} = \mathbf{r}^k \\ \mathbf{U}\boldsymbol{\delta} = \mathbf{y} \end{cases} \quad (4.47)$$

onde $\mathbf{L}$ e $\mathbf{U}$ são as matrizes resultantes da fatoração que preservam as posições nulas de $\mathbf{A}$.

O Algoritmo 4.9, descrito a seguir, utiliza a Fatoração Incompleta LU para realizar o refinamento.

Algoritmo 4.9 – Refinamento com ILU(0)

Entrada: Matriz de coeficientes $\mathbf{A}$ e vetor independente $\mathbf{b}$

Saída: Vetor solução $\mathbf{x}$

```

1: Início
2:    $\mathbf{b} \leftarrow \mathbf{r}_0$ 
3:   fatoracaoilu() // Realiza fatoração ILU(0) na matriz
                     // principal
4:    $it \leftarrow 0$ 
5:   Faça
6:      $it \leftarrow it + 1$ 
7:      $\mathbf{y} \leftarrow \text{subsProgressiva}(\mathbf{b})$ 
8:      $\boldsymbol{\delta} \leftarrow \text{subsRegressiva}(\mathbf{y})$ 
9:     atualizarSolucao()
10:    residuo()
11:    Se ( $|\mathbf{r}| < \text{tol1}$ ) Então
12:      Pare
13:    Senão
14:       $\mathbf{b} \leftarrow \mathbf{r}$  // Atualiza vetor independente
15:    Fim_se
16:  Enquanto ( $it \leq \text{ITERMAX}$ )
17: Fim
```

No Algoritmo 4.9 a fatoração é realizada uma única vez (linha 3) antes de iniciar a repetição com os passos do refinamento. Desta forma, pode-se observar a vantagem do uso deste método, pois faz-se uma única fatoração e a utiliza em todos os sucessivos refinamentos.

Na resolução desses sistemas, primeiro é utilizado a substituição progressiva (linha 7) para resolver o primeiro sistema e a regressiva (linha 8) para o segundo presente em (4.50), obtendo no final $\boldsymbol{\delta}$, o vetor de correção. Depois, basta atualizar a solução (4.48) e calcular o resíduo para essa nova solução.

Assim, todos os pré-condicionadores e suas variações, juntamente com o refinamento utilizando GMRES(m) e o ILU(0), foram implementados e utilizados nos problemas que serão apresentados a seguir. Os resultados obtidos serão apresentados no Capítulo 6.

Capítulo 5

Problemas Tratados

Neste capítulo são apresentados brevemente os problemas estudados, bem como suas principais características. Os problemas foram obtidos de dois repositórios on-line que fornecem as matrizes de sistemas lineares oriundos de diversas naturezas. Desenvolveu-se também um gerador aleatório, no qual pode-se facilmente definir algumas características como dimensão e formato da estrutura dos elementos que compõem a matriz: cheio, triangular ou tridiagonal.

5.1 Problemas dos Repositórios

Esses problemas foram retirados de dois repositórios: Matrix Market (<https://math.nist.gov/MatrixMarket/>) e SuiteSparse Matrix Collection (<https://sparse.tamu.edu/>). Na Tabela 1 são apresentadas suas principais características.

Tabela 1 – Propriedades dos problemas dos repositórios.

NOME	SIMÉTRICA	DEFINIDA POSITIVA	DIMENSÃO	NÃO-NULOS	DENSIDADE	NÚMERO DE CONDIÇÃO
<i>sherman1</i>	Sim	Não	1000	3750	0,375%	2,3E+04
<i>sherman2</i>	Não	Não	1080	23094	1,980%	1,4E+12
<i>b_dyn</i>	Não	Não	1089	4144	0,349%	2,5E+14
<i>b2_ss</i>	Não	Não	1089	3895	0,328%	7,6E+11
<i>sherman4</i>	Não	Não	1104	3786	0,311%	7,2E+03
<i>fpga_dcop_42</i>	Não	Não	1220	5892	0,396%	3,1E+12
<i>add20</i>	Não	Não	2395	17319	0,302%	1,7E+04
<i>cavity10</i>	Não	Não	2597	76367	1,132%	2,9E+06
<i>sherman5</i>	Não	Não	3312	20793	0,189%	3,9E+05
<i>c-26</i>	Sim	Não	4307	34537	0,186%	4,6E+08
<i>add32</i>	Não	Não	4960	23884	0,097%	2,1E+02
<i>sherman3</i>	Não	Não	5005	20033	0,080%	6,9E+16
<i>pres_poisson</i>	Sim	Sim	14822	715804	0,326%	2.0E+06
<i>memplus</i>	Não	Não	17758	126150	0,040%	1.2E+05

Observe que destas matrizes, somente três são simétricas, das quais uma única é definida positiva. No que se trata acerca de densidade, tem-se que todos os problemas possuem baixa densidade, na maioria abaixo de 1%, ou seja, a maior parte de seus elementos são nulos.

Outra característica importante é a dimensão dos problemas. Perceba que a maioria possui dimensões entre 1000 e 5000, considerados problemas médios, com dois, *pres_poisson* e *memplus*, que possuem dimensões elevadas, 14822 e 17758 respectivamente. Veja ainda que, pelo número de condição, todas são mal condicionadas, mostrando-se assim um desafio para resolvê-las.

5.2 Gerador de Problemas Aleatórios

Na busca por novos problemas para serem aplicados, a fim de observar o desempenho do método GMRES(m) e dos demais, buscou-se por um gerador de sistemas lineares interessante. Junior (2011) descreve um gerador simples, baseado em manipulações algébricas com vetores e matrizes. A seguir, será mostrado o passo a passo desse gerador.

Para implementação desse gerador foi utilizada a ferramenta Octave (Octave, 2019), pois oferece algumas facilidades, aproveitando característica dos próprios geradores, como, por exemplo, a necessidade de calcular os autovalores de uma matriz.

O gerador apresentado por Junior (2011), é baseado em sucessivas manipulações algébricas utilizando vetores e matrizes. Para gerar os problemas, tomou-se os seguintes passos:

1. Gerar um vetor aleatório $\mathbf{u} \in \mathbb{R}^n$ (onde n é a dimensão do problema), que seja uniformemente distribuído entre 0 e 1:

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \quad (5.1)$$

2. A partir do vetor $\mathbf{u}$ gera-se uma matriz ortogonal $\mathbf{M}$ em que $\mathbf{U} = \mathbf{I} - (2\mathbf{u}\mathbf{u}^T) / (\mathbf{u}^T \mathbf{u})$
e

$$\mathbf{M} = \begin{bmatrix} 1 - \frac{2u_1^2}{c} & -\frac{2u_1u_2}{c} & \dots & -\frac{2u_1u_n}{c} \\ -\frac{2u_2u_1}{c} & 1 - \frac{2u_2^2}{c} & \dots & -\frac{2u_2u_n}{c} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{2u_nu_1}{c} & -\frac{2u_nu_2}{c} & \dots & 1 - \frac{2u_n^2}{c} \end{bmatrix}, c = \mathbf{u}^T \mathbf{u} \quad (5.2)$$

3. Em seguida, gera-se uma nova matriz diagonal $\mathbf{D}$, na qual seus elementos serão os autovalores de $\mathbf{M}$ gerado anteriormente, lembrando que, por ser diagonal, é possível armazená-la em um vetor:

$$\mathbf{D} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} \quad (5.3)$$

4. Agora com $\mathbf{M}$ e $\mathbf{D}$ obtêm-se $\mathbf{A}$ da seguinte maneira: $\mathbf{A} = \mathbf{MDM}^T$, onde cada elemento é dado por:

$$a_{ij} = \sum_{k=1}^n m_{ik} d_k m_{jk}, i = 1, 2, \dots, n, j = 1, 2, \dots, n \quad (5.4)$$

5. Por último, basta gerar o vetor $\mathbf{b}$, gerando antes um vetor solução $\mathbf{x}$ aleatório e multiplicando por $\mathbf{A}$, obtendo, assim, $\mathbf{b}$.

Destaca-se, para este gerador, a característica das matrizes serem simétricas e definidas positivas. No entanto, nestes casos as matrizes são 100% densas (cheias), o que torna inviável computacionalmente resolver problemas maiores, a partir de algumas centenas de variáveis conforme testes realizados.

Observe a matriz **E2** a seguir como um exemplo de uma matriz gerada com dimensão

10.

$$\mathbf{E2} = \begin{bmatrix} -0,902 & 0,117 & 0,198 & 0,203 & 0,203 & 0,110 & 0,037 & 0,021 & 0,109 & 0,154 \\ 0,117 & 0,992 & -0,012 & -0,012 & -0,012 & -0,006 & -0,002 & -0,001 & -0,006 & -0,009 \\ 0,198 & -0,012 & 0,979 & -0,021 & -0,021 & -0,011 & -0,003 & -0,002 & -0,011 & -0,016 \\ 0,203 & -0,012 & -0,021 & 0,978 & -0,021 & -0,011 & -0,003 & -0,002 & -0,011 & -0,016 \\ 0,203 & -0,012 & -0,021 & -0,021 & 0,978 & -0,011 & -0,003 & -0,002 & -0,011 & -0,016 \\ 0,110 & -0,006 & -0,011 & -0,011 & -0,011 & 0,993 & -0,002 & -0,001 & -0,006 & -0,009 \\ 0,037 & -0,002 & -0,003 & -0,003 & -0,003 & -0,002 & 0,999 & -0,001 & -0,002 & -0,003 \\ 0,021 & -0,001 & -0,002 & -0,002 & -0,002 & -0,001 & -0,001 & 0,999 & -0,001 & -0,001 \\ 0,109 & -0,006 & -0,011 & -0,011 & -0,011 & -0,006 & -0,002 & -0,001 & 0,993 & -0,008 \\ 0,154 & -0,009 & -0,016 & -0,016 & -0,016 & -0,009 & -0,003 & -0,001 & -0,008 & 0,987 \end{bmatrix}$$

Observe que, além de serem simétricas definidas positivas, todos os elementos da primeira linha, com exceção do primeiro, são positivos. Essa formação é definida pelos autovalores de $\mathbf{U}$.

A partir deste gerador apresentado por Junior (2011) optou-se por fazer duas variações: uma para gerar matrizes triangulares e outra para gerar matrizes tridiagonais.

Para conseguir uma matriz triangular, primeiro gerou-se uma matriz cheia. Após este passo, utilizou-se a função do Octave chamada *trial* que retorna a parte triangular inferior de uma matriz qualquer. A matriz $\mathbf{E3}$ é um exemplo de como fica uma matriz triangular gerada a partir da matriz cheia $\mathbf{E2}$.

$$\mathbf{E3} = \begin{bmatrix} -0,902 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0,117 & 0,992 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0,198 & -0,012 & 0,979 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0,203 & -0,012 & -0,021 & 0,978 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0,203 & -0,012 & -0,021 & -0,021 & 0,978 & 0 & 0 & 0 & 0 & 0 \\ 0,110 & -0,006 & -0,011 & -0,011 & -0,011 & 0,993 & 0 & 0 & 0 & 0 \\ 0,037 & -0,002 & -0,003 & -0,003 & -0,003 & -0,002 & 0,999 & 0 & 0 & 0 \\ 0,021 & -0,001 & -0,002 & -0,002 & -0,002 & -0,001 & -0,001 & 0,999 & 0 & 0 \\ 0,109 & -0,006 & -0,011 & -0,011 & -0,011 & -0,006 & -0,002 & -0,001 & 0,993 & 0 \\ 0,154 & -0,009 & -0,016 & -0,016 & -0,016 & -0,009 & -0,003 & -0,001 & -0,008 & 0,987 \end{bmatrix}$$

Para a matriz tridiagonal, seguiu-se os passos de maneira análoga. Depois de ter gerado a matriz cheia, a função *trial* mais uma vez foi usada, juntamente com a *triau*, que semelhante à anterior, retorna a parte triangular superior de uma matriz. Essas duas também possuem um parâmetro que permite excluir diagonais a partir da principal.

Assim, ao chamar ambas as funções, excluiu-se a diagonal principal de ambas, como também a subdiagonal abaixo em *trial* e a subdiagonal superior em *triau*. Por fim, basta subtrair da matriz original essas duas quase triangulares, que se tem no final uma matriz tridiagonal. A matriz **E4** demonstra um exemplo de uma matriz tridiagonal gerada a partir da matriz **E2**.

$$E4 = \begin{bmatrix} -0,902 & 0,117 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0,117 & 0,992 & -0,012 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0,012 & 0,979 & -0,021 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0,021 & 0,978 & -0,021 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0,021 & 0,978 & -0,011 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0,011 & 0,993 & -0,002 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0,002 & 0,999 & -0,001 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0,001 & 0,999 & -0,001 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0,001 & 0,993 & -0,008 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0,008 & 0,987 \end{bmatrix}$$

Para cada uma das variações (cheia, triangular e tridiagonal) foram gerados problemas menores com dimensões entre 100 e 500 e problemas maiores com dimensões entre 1000 e 8000 para matrizes triangulares e tridiagonais.

Capítulo 06

Resultados

Neste capítulo, primeiramente é feita uma breve discussão sobre a configuração do ambiente no qual os testes foram realizados. Em seguida, são apresentadas algumas considerações gerais relevantes para o trabalho e dos meios utilizados para a execução dos testes. Por último, tem-se a apresentação dos resultados obtidos e os comentários sobre o desempenho dos métodos e técnicas utilizados.

6.1 Especificações e Parâmetros

Os experimentos computacionais realizados neste trabalho têm as seguintes especificações: notebook rodando o sistema operacional Windows 10 Home Single Language, com um processador i7-6500U de 2.5 GHz e com 8 GB de memória RAM.

Os algoritmos foram implementados em linguagem C, com o compilador GCC para Windows chamado MinGW. Além disso, cabe destacar que as matrizes e vetores usados foram alocados dinamicamente. Já para os testes envolvendo a plataforma Octave, foi utilizada a versão 5.2.0 (Octave, 2019).

No método GMRES(m) definiu-se os valores do parâmetro m como 1%, 5%, 10% e 15%, os quais demonstraram melhores resultados de acordo com Sousa e Silva (2016). As tolerâncias utilizadas foram: 10^{-12} para verificar se algum elemento da matriz $\mathbf{H}$ do GMRES(m) está muito próximo de zero e 10^{-8} para a tolerância da norma do resíduo como critério de parada. Esta última também foi utilizada nos critérios de parada para os demais métodos.

Por último, definiu-se também um tempo limite para o tempo de execução. Foi observado que, para os problemas maiores e mais complexos, os métodos levavam um tempo excessivo para resolvê-los, ou às vezes não convergiam. Por isso, o tempo máximo de execução foi estabelecido em 10 horas.

6.2 Conversão de Arquivos de Entrada

Os dados dos problemas dos repositórios utilizados são disponibilizados em apenas um arquivo que contém na primeira seção (verde) informações sobre as características do problema, seguida da dimensão da matriz e a quantidade de elementos não nulos. Na Figura 6.1 é apresentado um exemplo de arquivo de entrada para um problema de dimensão 3x3 com 5 elementos não nulos.

```
%%MatrixMarket matrix array real general
3 3 5
1 1 6.0945762000000e-01
3 1 5.2021017350000e-01
2 2 2.8000000000000e+01
1 3 4.7630000000000e+03
3 3 8.8472040000000e+06
.7257
.3091
.3627
```

Figura 6.1: Arquivo de entrada.

Observe que na parte destacada de vermelho aparecem três informações: primeiro o índice da linha, depois o índice da coluna e, por último, o valor do elemento que ocupa esta posição. Vale citar aqui que os elementos são apresentados por coluna na ordem que ocupam na matriz original. Ou seja, os elementos não nulos da primeira coluna da matriz original são apresentados, a seguir os elementos não nulos da segunda coluna e assim por diante.

Na última seção do arquivo, quando disponível, encontra-se os elementos não nulos do vetor $\mathbf{b}$ independente.

Esse arquivo de entrada, da forma como se obtém no repositório, é ideal para se trabalhar com a estrutura de dados CSC, pois seus elementos se encontram ordenados por coluna. Mas, como visto no Capítulo 3, neste trabalho foi utilizado a estrutura CSR, logo os elementos devem estar ordenados por linha.

Para isso, foi utilizado o algoritmo descrito por Silva (2017), o qual converte esse arquivo de entrada para um novo, onde seus elementos estão ordenados por linha. Logo, o arquivo ficará conforme mostrado na Figura 6.2:

```

%%MatrixMarket matrix array real general
3 3 5
1 1 6.0945762000000e-01
1 3 4.7630000000000e+03
2 2 2.8000000000000e+01
3 1 5.2021017350000e-01
3 3 8.8472040000000e+06
.7257
.3091
.3627

```

Figura 6.2: Arquivo de entrada após conversão.

Quando se tem os problemas oriundos do gerador proposto, os problemas são grandes e consideravelmente densos. No caso de uma matriz triangular é possível, logo após gera-la, fazer a sua transposta, a qual é gravada em um novo arquivo.

Agora, para as matrizes cheias do gerador, basta seguir a mesma ideia anterior, simplificando o processo: Quando a matriz gerada for simétrica, não é necessário fazer sua transposta.

6.3 Arquivo de Saída

Para uma análise adequada dos dados obtidos após aplicação de um método em um determinado problema, há a necessidade de um arquivo de saída bem estruturado, organizado e limpo, que dê todas as informações possíveis sobre o processo de resolução. Dentre estes dados, o resíduo (calculado conforme a Equação 2.1 do Capítulo 2) pode ser utilizado para verificar o quão boa uma solução é, ou ainda, quão próxima da solução exata ela está.

Portanto, para melhor análise do mesmo, estabeleceu-se a legenda a seguir, ligando o valor do resíduo com a qualidade da solução.

- C1 (REGULAR): Convergiu com norma do resíduo na faixa de 10^{-01} a 10^{-04} ;
- C2 (RAZOÁVEL): Convergiu com norma do resíduo na faixa de 10^{-05} a 10^{-09} ;
- C3 (BOA): Convergiu com norma do resíduo na faixa de 10^{-10} a 10^{-14} ;
- C4 (ÓTIMA): Convergiu com norma do resíduo na faixa de 10^{-15} a 10^{-19} ;
- C5 (EXCELENTE): Convergiu com norma do resíduo inferior a 10^{-20} .

Na Figura 6.3, é apresentado um exemplo de um arquivo de saída:

```
INI: Mon May 17 16:42:48 2021
```

```
<<GMRES(m) - Pré-condicionador Lumped - Versão 02>>
```

```

                Matriz: p10.mtx
                Dimensão: 3
                Valor de m: 3 (100% de n)
Tempo de verificação de elementos: 0.000
  Elementos faltantes na diagonal: 0
    Elementos nulos na diagonal: 0
      Problema tratável: SIM

```

```
<<Solução do Problema>>
```

```

Saídas do Arnoldi: 1
  Norma do resíduo: 1.359740e-016
Qualidade da solução: C4 (Ótima)
  Total de iterações: 1

```

```
<<Tempo de Resolução>>
```

```

Tempo de leitura do arquivo de entrada: 0.000 (0.00%)
  Tempo de resolução do sistema: 0.001 (100.00%)
    Tempo Total: 0.001 (100.00%)

```

```
FIN: Mon May 17 16:43:02 2021
```

Figura 6.3: Arquivo de saída do GMRES(m).

Neste exemplo, o método GMRES(m) juntamente com o pré-condicionador Lumped foi utilizado para resolver um problema pequeno (p10.mtx) de dimensão 3.

O arquivo de saída pode ser dividido em 3 partes principais: na primeira aparecem as principais informações sobre o problema tratado no momento, tais como qual método e/ou pré-condicionador está sendo utilizado, nome do arquivo de entrada, a dimensão do problema, tempo de validação desse arquivo (será discutido sobre essa validação na próxima seção), se existem elementos faltantes e/ou nulos no arquivo de entrada e, por último, se este problema pode ser aplicado com o método e pré-condicionador utilizado.

Na segunda parte, tem-se os dados principais obtidos com a solução do problema: quantidades de saídas do método de Arnoldi dentro do GMRES(m) (ocorre quando ele encontra uma divisão por zero), a norma do resíduo, o total de iterações e a qualidade da solução.

Na última parte, estão os tempos detalhados para resolução do problema, juntamente com sua porcentagem em relação ao tempo total. No exemplo considerado surgiram os tempos de leitura do arquivo de entrada, de resolução do sistema e o tempo total. Porém, dependendo do pré-condicionador, pode-se ter outros tempos, como, por exemplo, o Escala Simétrica, o ILU(0), e outros baseados neles.

Observe agora, na Figura 6.4, mais um exemplo de arquivo de saída, agora utilizando outro método.

```

INI: Thu Aug 26 09:01:40 2021

      <<Gradiente Conjugado - Estrutura CSR>>

                Matriz: p9.mtx
                Dimensão: 8
Tempo de verificação de elementos: 0.000
  Elementos faltantes na diagonal: 0
    Elementos nulos na diagonal: 0
      Problema tratável: SIM

      <<Solução do Problema>>

        Norma do resíduo: 2.468706e-01
Qualidade da solução: C4 (Ótima)
    Total de Iterações: 4

      <<Tempo de Resolução>>

Tempo de leitura do arquivo de entrada: 0.000 (0.00%)
    Tempo de resolução do sistema: 0.000 (0.00%)
          Tempo Total: 0.000 (0.00%)

FIN: Thu Aug 26 09:01:41 2021

```

Figura 6.4: Arquivo de saída do Gradiente Conjugado.

Desta vez, está sendo utilizado o método do Gradiente Conjugado para resolver um problema pequeno de dimensão 8 (p9.mtx). Note que o padrão do arquivo não mudará muito de acordo com os métodos utilizados. A principal diferença se encontra em quais dados serão apresentados, que neste caso, o valor de m e a quantidade de saídas do processo de Arnoldi não aparecem, pois são informações específicas do método GMRES(m).

6.4 Validação do Arquivo de Entrada

No decorrer dos primeiros testes foram observados casos que chamaram a atenção, principalmente envolvendo divisões por zeros. Após uma breve verificação, foi constatado que os arquivos de entrada baixados dos repositórios vieram com alguns elementos faltantes e/ou com nulos em sua diagonal.

Elementos faltantes podem ser considerados possíveis de ocorrer nesse caso, porém, nulos não deveriam estar nestes arquivos, já que este formato de arquivo armazena somente os elementos não-nulos. Esse fato torna esses problemas impossíveis de serem aplicados em determinadas situações (como, por exemplo, os pré-condicionadores que dependem que a diagonal da matriz seja diferente de zero).

Considerando este problema, foi elaborada uma rotina para verificar essas ocorrências, e assim, terminar o algoritmo antes de aplicar o método de resolução. Tal rotina é simples de ser elaborada.

Vale citar aqui também, que elementos negativos na diagonal podem trazer problemas para os pré-condicionadores Escala Simétrica e variantes, quando estes necessitarem realizar a raiz quadrada dos elementos da diagonal. Porém, para esses casos, utiliza-se o módulo destes elementos.

6.5 Resultados

A partir de agora serão apresentados os resultados obtidos com os testes realizados. Primeiro virão os dados com os métodos Jacobi, Gauss-Seidel e Gradiente Conjugado e depois apresenta-se os resultados relacionados aos pré-condicionadores incorporados ao GMRES(m).

Vale destacar também que, cada seção aqui, se subdivide em mais 2 subseções: uma para os problemas dos repositórios e outra para gerador aleatório.

Para facilitar a leitura dos resultados para todos os métodos, foi utilizada a seguinte legenda:

- I: Número de Iterações requeridas pelo método para encontrar a solução;
- R: Qualidade do resíduo (com base na legenda apresentada na Seção 6.2.2) referente a solução encontrada;
- TLA: Tempo de Leitura do Arquivo de Entrada;
- TRS: Tempo de Resolução do Sistema;
- TCNS: Tempo de Construção do Novo Sistema;
- TRNS: Tempo de Resolução do Novo Sistema;
- TF: Tempo de Fatoração;
- TAPD: Tempo de Aplicação do Pré-condicionador Diagonal;
- NC: Não Convergiu com Resíduo Muito Alto;

- TL: Ultrapassou Tempo Limite de 10 horas;
- EV: Variável Recebeu um Valor Maior que o Limite do seu Tipo Comporta;
- EM: Memória Insuficiente para Alocar Vetor e/ou Matriz;

6.5.1 Jacobi com as Estrutura de Dados Indexada e CSR

6.5.1.1 Problemas dos Repositórios

Na Tabela 2 são apresentados a quantidade de iterações, a qualidade do resíduo e os tempos de leitura (envolve o tempo de leitura do arquivo de entrada e para armazenar os dados lidos na estrutura de dados) e o de resolução do método de Jacobi.

Tabela 2 – Jacobi com estrutura indexada e CSR – Repositórios.

PROBLEMAS	Estrutura Indexada					Estrutura CSR			
	I	R	TLA	TRS		I	R	TLA	TRS
<i>sherman1</i>	34488	C2	0,015	104,694		34488	C2	0,016	0,593
<i>sherman4</i>	6900	C2	0,015	25,588		6900	C2	0,000	0,140
<i>add20</i>	19016	C2	0,031	327,932		19016	C2	0,031	13,822
<i>c-26</i>	3	C5	0,047	0,172		3	C5	0,031	0,014
<i>add32</i>	547	C2	0,047	40,918		547	C2	0,047	0,234
<i>sherman3</i>	75044	C2	0,031	5830,457		75044	C2	0,047	28,294
<i>pres_poisson</i>	131	C2	0,718	87,263		131	C2	0,687	0,219
<i>memplus</i>	EM					7622	C2	0,237	7,630

Nesta tabela, é possível verificar que, nas quantidades de iterações e qualidade dos resíduos não houve diferenças comparando as estruturas indexada e CSR, já que a mudança de estrutura de dados não interfere nesses valores.

Vale destacar que o problema *c-26*, que foi resolvido com um resíduo com qualidade excelente e um tempo por iteração baixo (0,057 segundos por iteração) na estrutura indexada. Com a estrutura CSR, as iterações permaneceram as mesmas, porém o tempo de resolução deste caiu bastante, reduzindo, assim, também o tempo por iteração (passou a ser, aproximadamente, 0,005 segundos por iteração).

Para os demais problemas, o método com a estrutura indexada teve que realizar muitas iterações, juntamente com um aumento do tempo. Porém, ao utilizar a estrutura CSR, foi possível perceber uma grande redução dos tempos de resolução. Tome como exemplo o problema *sherman3*. Na estrutura indexada foram necessários, aproximadamente, 5830 segundos para resolvê-lo. Com a nova estrutura apenas 28 segundos foram utilizados. Como a

quantidade de iterações não é alterada, nota-se que o esforço que o método teve por iteração reduziu drasticamente.

Obviamente, a considerável redução no tempo está relacionada a densidade do problema tratado com a estrutura CSR. Com uma densidade de 0,080%, antes, na estrutura indexada, o método realizava cálculos, mesmo com os elementos nulos. Porém, a estrutura CSR permitiu que fosse utilizado somente os cálculos que envolvam os não nulos, economizando assim operações desnecessárias e, conseqüentemente, diminuindo o esforço necessário para cada iteração.

Para os problemas *b_dyn*, *b2_ss*, *fpga_dcop_42* e *cavity10* não foi possível utilizar este método, pois constam elementos faltantes e/ou nulos na diagonal principal de suas matrizes. Quando aplicado no maior problema, *memplus*, não houve memória suficiente para alocar as estruturas utilizadas. Porém, com o uso da estrutura CSR no método foi possível resolver este problema, com um tempo baixo.

Por último, o método não conseguiu resolver dois problemas, *sherman2* e *sherman5*, pois no decorrer das iterações, o conteúdo das variáveis continuou crescendo, chegando ao ponto que não poderia comportar mais esse valor, ocorrendo, assim, *overflow* ou estouro da variável.

6.5.1.2 Problemas do Gerador Aleatório

Nas Tabelas 3, 4 e 5, são apresentados os números de iterações, qualidade dos resíduos e os tempos de leitura e os de resolução, para os problemas de matrizes cheias, triangulares e tridiagonais, respectivamente.

Tabela 3 - Jacobi com estrutura indexada e CSR – Gerador aleatório – Matriz cheia.

PROBLEMAS		Estrutura Indexada				Estrutura CSR			
TIPO	DIMENSÃO	I	R	TLA	TRS	I	R	TLA	TRS
CHEIA	100	41	C2	0,016	0,004	41	C2	0,015	0,016
	500	18	C2	0,453	0,015	18	C2	0,437	0,031
	1000	13	C2	1,781	0,047	13	C2	1,812	0,047
	1500	9	C3	4,062	0,062	9	C3	4,092	0,078
	2000	11	C2	7,654	0,141	11	C2	7,311	0,138
	3500	9	C2	22,448	0,343	3	C2	22,573	0,328
	5000	7	C2	46,380	0,531	7	C2	46,255	0,547
	6500	7	C3	78,964	0,906	7	C3	78,340	0,921
	8000	9	C3	117,799	1,734	9	C3	118,504	1,771

Através da Tabela 3 é possível notar que, para esses problemas, o método de Jacobi foi eficiente ao resolver todos os problemas em poucas iterações e baixo tempo de resolução. Perceba que os tempos de leitura dos arquivos de entrada cresce conforme a dimensão dos problemas fica maior em ambas as estruturas. O método também apresentou ligeiras diferenças nos tempos de resolução para esses problemas após o uso da estrutura CSR, mas nada tão evidente.

Tabela 4 - Jacobi com estrutura indexada e CSR – Gerador aleatório – Matriz triangular.

PROBLEMAS		Estrutura Indexada				Estrutura CSR			
TIPO	DIMENSÃO	I	R	TLA	TRS	I	R	TLA	TRS
TRIANGULAR	100	5	C2	0,000	0,000	5	C2	0,016	0,000
	500	5	C3	0,219	0,015	5	C3	0,219	0,012
	1000	5	C3	0,890	0,017	5	C3	0,922	0,000
	1500	4	C3	2,047	0,033	4	C3	2,046	0,016
	2000	4	C2	3,640	0,047	4	C2	3,671	0,016
	3500	4	C3	11,217	0,140	4	C3	11,326	0,078
	5000	4	C3	22,963	0,313	4	C3	23,667	0,156
	6500	3	C2	39,188	0,390	3	C2	39,147	0,204
	8000	4	C3	58,814	0,799	4	C3	59,346	0,406

Para os problemas triangulares, o método de Jacobi conseguiu resolvê-los com poucas iterações e, mais uma vez, com tempos de resolução abaixo de 1 segundo. Para o tempo de leitura, perceba que agora foi, aproximadamente, reduzido pela metade em comparação com os problemas cheios.

O uso da estrutura CSR não exerceu tanta influência com relação aos tempos da estrutura indexada para os problemas menores. Porém, conforme o tamanho do problema aumenta (a partir de 1500), a estrutura conseguiu reduzir os tempos de resolução consideravelmente.

Tabela 5 - Jacobi com estrutura indexada e CSR – Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		Estrutura Indexada				Estrutura CSR			
TIPO	DIMENSÃO	I	R	TLA	TRS	I	R	TLA	TRS
TRIDIAGONAL	100	4	C2	0,000	0,000	4	C2	0,000	0,000
	500	5	C3	0,000	0,000	5	C3	0,000	0,000
	1000	5	C2	0,016	0,015	5	C2	0,000	0,016
	1500	4	C2	0,015	0,032	4	C2	0,015	0,000
	2000	4	C3	0,015	0,047	4	C3	0,016	0,000
	3500	4	C2	0,016	0,156	4	C2	0,031	0,000
	5000	4	C2	0,031	0,308	4	C2	0,031	0,000
	6500	4	C3	0,046	0,511	4	C3	0,047	0,000
	8000	4	C3	0,047	0,793	4	C3	0,047	0,000

Nos problemas tridiagonais os resultados foram bem próximos aos descritos na Tabela 4, com exceção dos tempos de leituras que foram bem menores, já que neste caso a densidade é menor. Destaca-se aqui que, usando a estrutura indexada, o método conseguiu obter um resíduo de qualidade boa com um baixo tempo por iteração (0,198 segundos para a dimensão 8000).

Agora, usando a estrutura de dados CSR conseguiu-se resultados mais interessantes em relação aos tempos. Levando em conta três casas decimais, quase todos os problemas foram resolvidos em zero milissegundo. Novamente, ressalta-se a vantagem do uso dessa estrutura para tratar problemas esparsos.

6.5.2 Gauss-Seidel com Estrutura de Dados Indexada e CSR

6.5.2.1 Problemas dos Repositórios

Na Tabela 6 serão apresentadas as quantidades de iterações, os resíduos e os tempos de leitura e de resolução para o método de Gauss-Seidel.

Tabela 6 - Gauss-Seidel com estrutura indexada e CSR - Repositórios.

PROBLEMAS	Estrutura Indexada					Estrutura CSR			
	I	R	TLA	TRS		I	R	TLA	TRS
<i>sherman1</i>	17845	C2	0,016	54,159		17845	C2	0,016	0,328
<i>sherman4</i>	3622	C2	0,000	15,295		3622	C2	0,000	2,658
<i>add20</i>	8909	C2	0,031	159,535		8909	C2	0,031	7,255
<i>c-26</i>	2	C5	0,045	0,110		2	C5	0,032	0,000
<i>add32</i>	251	C2	0,046	18,796		251	C2	0,047	0,062
<i>sherman3</i>	40248	C2	0,031	3125,375		40248	C2	0,031	8,322
<i>pres_poisson</i>	2	C5	0,703	1,405		2	C5	0,703	0,000
<i>memplus</i>	EM					3857	C2	0,250	3,618

O método de Gauss-Seidel conseguiu resolver os mesmos problemas que o Jacobi, porém, nota-se um desempenho melhor que o anterior, reduzindo consideravelmente o número de iterações e também o seu tempo de resolução.

Cabe destacar a performance do método para os problemas: *c-26* e *pres_poisson*. Além de resolvê-los com apenas 2 iterações, também conseguiu obter a melhor qualidade de solução. Note ainda o tempo de resolução obtido para ambos os problemas de, apenas, menos de um segundo por iteração para resolvê-los.

Ainda nesta tabela é possível observar, mais uma vez, a grande vantagem do uso de uma estrutura de dados que trata a esparsidade. Os tempos sofreram uma redução considerável comparados com a estrutura indexada. Destaca-se que o método conseguiu resolver um problema, *memplus*, que antes causava estouro de memória.

Para os demais problemas registra-se as mesmas ocorrências que no método de Jacobi: problemas com elementos nulos e/ou faltantes na diagonal, assim como, momentos em que ocorriam *overflow*.

6.5.2.2 Problemas do Gerador Aleatório

Nas Tabelas 7, 8 e 9 abaixo são apresentadas as quantidades de iterações, os resíduos e os tempos de leitura e de resolução, para os problemas de matrizes cheias, triangulares e tridiagonais, respectivamente.

Tabela 7 - Gauss-Seidel com estrutura indexada e CSR – Gerador aleatório – Matriz cheia.

PROBLEMAS		Estrutura Indexada				Estrutura CSR			
TIPO	DIMENSÃO	I	R	TLA	TRS	I	R	TLA	TRS
CHEIA	100	20	C2	0,015	0,000	20	C2	0,016	0,005
	500	10	C3	0,438	0,015	10	C3	0,437	0,014
	1000	8	C3	1,780	0,021	8	C3	1,843	0,032
	1500	5	C2	4,061	0,031	5	C2	4,124	0,035
	2000	7	C3	7,311	0,078	7	C3	7,296	0,082
	3500	6	C3	23,120	0,219	6	C3	22,525	0,234
	5000	5	C3	45,895	0,390	5	C3	46,161	0,391
	6500	5	C3	78,294	0,661	5	C3	78,593	0,655
	8000	6	C3	117,831	1,161	6	C3	118,800	1,203

Para os problemas de matrizes cheias o método de Gauss-Seidel conseguiu ótimos resultados reduzindo a quantidade de iteração para em torno da metade com relação ao método Jacobi e também uma leve redução dos tempos de resolução nos problemas com dimensão a partir de 1000. O método conseguiu melhorar levemente os resíduos dos problemas de dimensão 500 e 1000, apesar do contrário ter acontecido em alguns problemas.

O uso da estrutura CSR não demonstrou, nestes problemas, vantagem pois não trouxe mudanças relevantes no tempo de leitura e até mesmo os tempos de resolução ficaram ligeiramente maiores, quando comparados com os da estrutura indexada.

Tabela 8 - Gauss-Seidel com estrutura indexada e CSR – Gerador aleatório – Matriz triangular.

PROBLEMAS		Estrutura Indexada				Estrutura CSR			
TIPO	DIMENSÃO	I	R	TLA	TRS	I	R	TLA	TRS
TRIANGULAR	100	2	C5	0,000	0,000	2	C5	0,000	0,000
	500	2	C5	0,219	0,000	2	C5	0,219	0,015
	1000	2	C5	0,891	0,000	2	C5	0,890	0,016
	1500	2	C5	2,031	0,016	2	C5	2,109	0,008
	2000	2	C5	3,624	0,032	2	C5	3,687	0,016
	3500	2	C5	11,232	0,068	2	C5	11,294	0,032
	5000	2	C5	22,948	0,156	2	C5	23,150	0,078
	6500	2	C5	38,662	0,264	2	C5	39,022	0,125
	8000	2	C5	58,908	0,406	2	C5	59,173	0,203

Nos problemas triangulares destaca-se que este método conseguiu a melhor qualidade de solução para todos os problemas, além da redução do tempo de resolução também ser bem melhor. Agora, ficou visível também a vantagem da utilização da estrutura de dados CSR, pois os tempos de resolução sofreram uma queda considerável em seu valor.

Tabela 9 - Gauss-Seidel com estrutura indexada e CSR – Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		Estrutura Indexada				Estrutura CSR			
TIPO	DIMENSÃO	I	R	TCE	TRS	I	R	TCE	TRS
TRIDIAGONAL	100	3	C2	0,000	0,000	3	C2	0,000	0,015
	500	4	C3	0,000	0,000	4	C3	0,000	0,000
	1000	4	C3	0,016	0,016	4	C3	0,015	0,000
	1500	3	C3	0,000	0,031	3	C3	0,000	0,000
	2000	3	C3	0,000	0,047	3	C3	0,016	0,000
	3500	3	C3	0,016	0,125	3	C3	0,015	0,016
	5000	3	C2	0,031	0,234	3	C2	0,031	0,016
	6500	3	C3	0,047	0,390	3	C3	0,047	0,000
	8000	3	C3	0,047	0,594	3	C3	0,063	0,000

O método de Gauss-Seidel continuou apresentando bons resultados, agora nos problemas tridiagonais. Da mesma forma que nos outros problemas, mas não tão evidente, conseguiu reduzir um pouco as iterações e o tempo de resolução, e aumentou um pouco a qualidade dos resíduos. Isso mostra que, em comparação com o método de Jacobi, este método demonstra ser mais eficiente em obter uma solução de melhor qualidade.

O uso da estrutura CSR, com estes problemas, demonstrou excelentes resultados. Como esses problemas são bem esparsos, a estrutura conseguiu reduzir bastante os tempos computacionais, utilizando-se apenas alguns milésimos de segundos em alguns problemas.

6.5.3 Gradiente Conjugado com Estrutura de Dados Indexada e CSR

6.5.3.1 Problemas dos Repositórios

Na Tabela 10 são apresentados as iterações, resíduos e os tempos de leitura e de resolução obtidos com o uso do método do Gradiente Conjugado.

Tabela 10 - Gradiente Conjugado com estrutura indexada e CSR - Repositórios.

PROBLEMAS	Estrutura Indexada					Estrutura CSR			
	I	R	TLA	TRS		I	R	TLA	TRS
<i>sherman1</i>	577	C2	0,015	2,047		577	C2	0,000	0,485
<i>sherman4</i>	512	C2	0,016	512		512	C2	0,015	0,454
<i>fpga_dcop_42</i>	2440	C2	0,015	12,342		2440	C2	0,015	2,145
<i>add20</i>	1	C3	0,031	0,015		1	C3	0,032	0,013
<i>cavity10</i>	1	C3	0,125	0,032		1	C3	0,125	0,000
<i>add32</i>	1	C3	0,047	0,078		1	C3	0,047	0,000
<i>sherman3</i>	6240	C1	0,032	474,411		6240	C1	0,031	3,404
<i>pres_poisson</i>	EM					4468	C2	0,687	13,592
<i>memplus</i>	EM					1	C3	0,235	0,000

Através desta tabela, nota-se que o método do Gradiente Conjugado foi melhor que os dois métodos anteriores tanto em quantidade de iterações como em tempo de resolução.

Observe que no caso do problema *sherman3* este método com a estrutura indexada precisou de, aproximadamente, 0,078 segundos por iteração para resolvê-lo, mesmo tempo por iteração dos outros dois métodos. Assim, pode-se dizer, quanto a quantidade de iteração, que o mesmo foi melhor que os outros, apesar de perder um pouco de precisão no resíduo.

Porém, ao utilizar a estrutura de dados CSR, fica claro, mais uma vez, as vantagens de seu uso para problemas esparsos. Primeiro a redução do tempo de resolução. Por exemplo, novamente para o problema *sherman3*, para a CSR foram necessários apenas 0,0005 segundos por iteração.

Outra vantagem é que problemas que antes, na estrutura indexada, ocasionavam estouros de memórias (*pres_poisson* e *memplus*), agora, além de serem resolvidos, possuem ótimos resultados. Observe o problema *memplus*. Além de obter uma boa qualidade de resíduo, isso foi feito com uma única iteração em zero milissegundo.

Quando este método foi aplicado nos problemas *sherman2*, *b_dyn*, *b2_ss*, *sherman5* e *c-26*, os valores dos resíduos foram altos, ou seja, ele não convergiu para esses casos. Vale citar que destes, com relação aos quatros primeiros problemas a matriz não é simétrica e nem

definida positiva. Para o problema c-26, apesar de ser simétrica, a matriz não é definida positiva.

6.5.3.2 Problemas do Gerador Aleatório

Nas Tabelas 11, 12 e 13, abaixo, são apresentadas as quantidades de iterações, os resíduos e os tempos de leitura e de resolução, para os problemas de matrizes cheias, triangulares e tridiagonais, respectivamente.

Tabela 11 - Gradiente Conjugado com estrutura indexada e CSR – Gerador aleatório – Matriz cheia.

PROBLEMAS		Estrutura Indexada				Estrutura CSR			
TIPO	DIMENSÃO	I	R	TLA	TRS	I	R	TLA	TRS
CHEIA	100	3	C4	0,016	0,000	3	C4	0,016	0,000
	500	3	C4	0,453	0,000	3	C4	0,437	0,016
	1000	3	C4	1,781	0,015	3	C4	1,781	0,016
	1500	3	C4	4,062	0,031	3	C4	4,077	0,038
	2000	3	C3	7,264	0,047	3	C3	7,451	0,047
	3500	3	C4	22,448	0,153	3	C4	22,666	0,157
	5000	3	C4	45,911	0,312	3	C4	46,176	0,313
	6500	3	C4	77,604	0,528	3	C4	78,309	0,554
	8000	3	C3	117,882	0,765	3	C3	118,316	0,835

Nos problemas com matrizes cheias, o Gradiente Conjugado mostrou ótimos resultados, destacando a qualidade do resíduo, onde, até este ponto, foi o que melhor trabalhou com esses tipos de problemas, desde que suas matrizes sejam simétricas e definidas positivas. Neste caso o uso da estrutura de dados CSR não proporcionou nenhuma mudança considerável para os problemas cheios.

Tabela 12 - Gradiente Conjugado com estrutura indexada e CSR – Gerador aleatório – Matriz triangular.

PROBLEMAS		Estrutura Indexada				Estrutura CSR			
TIPO	DIMENSÃO	I	R	TLA	TRS	I	R	TLA	TRS
TRIANGULAR	100	200	C2	0,000	0,172	200	C2	0,000	0,156
	500	8	C3	0,218	0,016	8	C3	0,219	0,015
	1000	2000	C1	0,890	7,093	2000	C1	0,890	4,261
	1500	7	C3	2,031	0,062	7	C3	2,047	0,031
	2000	9	C2	3,640	0,125	9	C2	3,686	0,078
	3500	916	C2	11,200	34,453	916	C2	11,341	18,495
	5000	7	C2	22,948	0,603	7	C2	23,088	0,313
	6500	5	C3	38,648	0,766	5	C3	39,471	0,407
	8000	570	C2	59,033	109,625	570	C2	59,003	58,003

Para as matrizes triangulares, como perderam a propriedade de serem definidas positivas, o método já se comportou de maneira mais instável. Por exemplo, para o problema de dimensão 1000, apesar que conseguir resolvê-lo em apenas 7 segundos com a estrutura indexada, foi preciso o máximo de iterações para que ele conseguisse uma solução com um resíduo regular. O uso da estrutura CSR nos problemas triangulares mostrou ser adequado desde que para uma parte deles (problemas com dimensão maior que 1000), os tempos de resolução foram reduzidos em até quase 50 %.

Tabela 13 - Gradiente Conjugado com estrutura indexada e CSR – Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		Estrutura Indexada				Estrutura CSR			
TIPO	DIMENSÃO	I	R	TLA	TRS	I	R	TLA	TRS
TRIDIAGONAL	100	3	C4	0,000	0,000	3	C4	0,000	0,000
	500	4	C3	0,000	0,000	4	C3	0,000	0,000
	1000	4	C3	0,000	0,015	4	C3	0,015	0,000
	1500	4	C4	0,016	0,031	4	C4	0,000	0,000
	2000	3	C4	0,000	0,063	3	C4	0,000	0,015
	3500	3	C4	0,015	0,141	3	C4	0,016	0,016
	5000	3	C3	0,031	0,300	3	C3	0,031	0,000
	6500	3	C4	0,032	0,515	3	C4	0,047	0,000
	8000	3	C4	0,062	0,776	3	C4	0,062	0,000

Com os problemas tridiagonais, novamente ocorreram ótimos resultados no que diz respeito a qualidade e iterações, apesar de seus tempos de resolução serem maiores que os do Gauss-Seidel. Note que, até mesmo, o maior problema (de dimensão 8000) foi resolvido utilizando apenas 3 iterações com, aproximadamente, 0,258 segundos para cada uma.

Por fim, confirmando o que já foi discutido até este ponto, mais uma vez a estrutura de dados CSR demonstra a principal vantagem de seu uso: a redução do tempo de resolução em problemas muito esparsos.

6.5.4 GMRES(m) com Estrutura de Dados CSR

Silva et. al. (2017), em seu artigo, comparam o desempenho da estrutura de dados CSR com a estrutura indexada, no método GMRES(m), obtendo resultados que demonstram a vantagem do uso da CSR para resolver problemas grandes e esparsos. Portanto esta comparação não será feita aqui, apresentando somente os resultados envolvendo a estrutura CSR.

6.5.4.1 Problemas dos Repositórios

A Tabela 14 abaixo traz as quantidades de iterações e as qualidades dos resíduos obtidos pelo método GMRES(m).

Tabela 14 – GMRES(m) com estrutura CSR - Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
	I	R	I	R	I	R	I	R
<i>sherman1</i>	1001	C2	49	C2	12	C2	6	C2
<i>sherman4</i>	154	C2	6	C2	2	C3	2	C3
<i>fpga_dcop_42</i>	7	C2	73	C2	2	C2	6	C2
<i>add20</i>	1	C3	1	C3	1	C4	1	C5
<i>cavity10</i>	1	C3	1	C3	1	C3	1	C3
<i>sherman5</i>	1798	C2	79	C2	25	C2	13	C2
<i>add32</i>	1	C5	1	C4	1	C4	1	C4
<i>sherman3</i>	2783	C2	30	C2	4	C3	3	C3
<i>pres_poisson</i>	5	C2	2	C2	TL		TL	
<i>memplus</i>	1	C3	1	C4	1	C3	TL	

Da Tabela 14, nota-se a influência do parâmetro m no método GMRES reinicializado tanto nas iterações e quanto no resíduo. No problema *add20*, conforme o valor desse parâmetro aumentava, a qualidade do resíduo também melhorava, ao ponto que, quando foi utilizado 15% da dimensão do problema, o método conseguiu um resíduo de qualidade C5 (muito próximo de zero).

Para o problema *sherman5*, o método GMRES(m) conseguiu resolvê-lo, sendo que conforme o parâmetro m crescia, a quantidade de iterações realizadas era reduzida. Vale destacar ainda que os métodos anteriores não convergiram para este problema.

Dentre esses resultados, destacam-se o *add20* com m de 15% e o *add32* com m de 1% que conseguiram resíduos de qualidade C5. Houve ainda alguns problemas em que o resíduo foi elevado e/ou os valores das variáveis ultrapassaram seus limites.

Na Tabela 15 são apresentados os tempos de leitura e os tempos de resolução obtidos pelo método GMRES(m).

Tabela 15 – GMRES(m) com estrutura CSR - Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
	TLA	TRS	TLA	TRS	TLA	TRS	TLA	TRS
<i>sherman1</i>	0,000	1,437	0,000	1,168	0,000	2,595	0,000	5,458
<i>sherman4</i>	0,000	0,266	0,000	0,218	0,000	0,623	0,015	2,687
<i>fpga_dcop_42</i>	0,016	0,000	0,015	3,601	0,014	0,890	0,016	11,967
<i>add20</i>	0,031	0,005	0,047	0,531	0,031	6,167	0,032	31,026
<i>cavity10</i>	0,125	0,032	0,141	0,718	0,141	9,389	0,141	43,418
<i>sherman5</i>	0,047	51,209	0,047	135,556	0,031	584,446	0,031	2003,241
<i>add32</i>	0,047	0,092	0,047	8,826	0,047	158,460	0,047	761,648
<i>sherman3</i>	0,031	248,033	0,047	276,883	0,031	658,205	0,047	2380,994
<i>pres_poisson</i>	0,672	17,485	0,687	1759,375	X		X	
<i>memplus</i>	0,234	6,266	0,250	1679,941	0,250	32363,550	X	

Nesta tabela destaca-se que os tempos de construção são independentes do parâmetro m , e, levando em conta os problemas maiores, não é tão predominante quanto os tempos de resolução.

Também é possível notar que, conforme esse parâmetro aumenta, o tempo para resolver determinado problema também aumenta. Isso fica mais claro quando se observa os tempos por iteração. Tome, por exemplo, o problema *sherman5*. Quando é usado 1% em m apesar de realizar 2783 iterações, seu tempo para cada uma delas foi de 0,028 segundos. Agora, quando m equivale a 15%, para cada iteração foram necessários, aproximadamente, 154 segundos.

6.5.4.2 Problemas do Gerador Aleatório

Nas Tabelas 16, 17 e 18 são apresentadas as quantidades de iterações e os resíduos para os problemas de matrizes cheias, triangulares e tridiagonais, respectivamente.

Tabela 16 – GMRES(m) com estrutura CSR – Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\%$ de n		$m = 5\%$ de n		$m = 10\%$ de n		$m = 15\%$ de n	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
CHEIA	100	18	C2	2	C4	5	C4	12	C3
	500	1	C3	2	C3	2	C3	1	C2
	1000	3	C3	EV		EV		EV	
	1500	6	C3	6	C3	EV		EV	
	2000	40	C3	6	C3	EV		EV	
	3500	5	C3	EV		EV		EV	

Da tabela acima, nota-se que o método GMRES(m) teve algumas ocorrências de estouro de variáveis, que foram mais frequentes conforme o crescimento do valor da dimensão do problema e do parâmetro m escolhido. Em alguns casos, principalmente para os menores, ele convergiu com boas e ótimas qualidades de resíduo e baixas iterações. Para os problemas de dimensão 5000, 6500 e 8000, o método não conseguiu resolver para nenhum dos valores de m pois ocorreram casos de estouro de variável.

Tabela 17 – GMRES(m) com estrutura CSR – Gerador aleatório – Matriz triangular.

PROBLEMAS		$m = 1\%$ de n		$m = 5\%$ de n		$m = 10\%$ de n		$m = 15\%$ de n	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIANGULAR	100	13	C2	1	C3	6	C3	15	C3
	500	1	C3	8	C3	3	C4	3	C3
	1000	6	C2	6	C2	EV		EV	
	1500	27	C3	EV		EV		EV	
	2000	22	C2	EV		EV		EV	
	3500	130	C3	EV		EV		EV	
	5000	34	C3	EV		EV		EV	
	6500	3	C3	EV		EV		EV	
	8000	56	C3	EV		EV		EV	

Nos problemas triangulares, para o problema com dimensão 500 com m valendo 10% o método convergiu, obtendo uma solução de ótima qualidade. Para $m = 1\%$, nota-se que o método convergiu para todos problemas. Mas, há o estouro de variável conforme este parâmetro aumenta.

Tabela 18 – GMRES(m) com estrutura CSR – Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIDIAGONAL	100	12	C2	1	C4	9	C3	3	C2
	500	1	C3	17	C2	29	C3	7	C2
	1000	3	C3	13	C3	14	C3	4	C3
	1500	10	C2	7	C3	28	C3	EV	
	2000	20	C3	EV		EV		EV	
	3500	EV		EV		EV		EV	
	5000	9	C3	EV		EV		EV	
	6500	6	C2	EV		EV		EV	
	8000	2	C3	EV		EV		EV	

Na Tabela 18 destaca-se o problema de 100 de dimensão com parâmetro $m = 5\%$ em que obteve uma ótima qualidade. Note ainda que o método convergiu também para o maior problema (8000 de dimensão) em apenas 2 iterações.

Nas Tabelas 19, 20 e 21 são apresentados os tempos de leitura e os tempos de resolução do método para os problemas de matrizes cheias, triangulares e tridiagonais, respectivamente.

Tabela 19 – GMRES(m) com estrutura CSR – Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	TLA	TRS	TLA	TRS	TLA	TRS	TLA	TRS
CHEIA	100	0,016	0,009	0,016	0,012	0,016	0,000	0,016	0,012
	500	0,437	0,016	0,437	0,074	0,453	0,140	0,453	0,156
	1000	1,797	0,156	X		X		X	
	1500	4,092	1,016	4,077	6,561	X		X	
	2000	7,498	18,355	7,420	15,967	X		X	
	3500	22,587	13,356	X		X		X	

A tabela acima mostra que, apesar de serem matrizes cheias, o método conseguiu resolvê-las com tempos baixos. No caso do problema de dimensão 100, foi obtido um tempo por iteração de apenas 0,006 segundos.

Tabela 20 – GMRES(m) com estrutura CSR – Gerador aleatório – Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	TLA	TRS	TLA	TRS	TLA	TRS	TLA	TRS
TRIANGULAR	100	0,000	0,000	0,000	0,000	0,016	0,000	0,015	0,016
	500	0,219	0,019	0,219	0,141	0,219	0,130	0,234	0,313
	1000	0,890	0,157	0,906	0,844	X		X	
	1500	2,046	2,265	X		X		X	
	2000	3,671	4,519	X		X		X	
	3500	11,326	168,409	X		X		X	
	5000	23,076	130,217	X		X		X	
	6500	39,272	25,822	X		X		X	
	8000	65,859	936,542	X		X		X	

Nos problemas triangulares, é visível, conforme o parâmetro m cresce, que sua influência tende a fazer com que o tempo de resolução cresça também.

Tabela 21 – GMRES(m) com estrutura CSR – Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	TLA	TRS	TLA	TRS	TLA	TRS	TLA	TRS
TRIDIAGONAL	100	0,000	0,015	0,000	0,000	0,000	0,002	0,000	0,001
	500	0,000	0,000	0,000	0,047	0,000	0,484	0,016	0,473
	1000	0,000	0,016	0,015	0,296	0,000	3,003	0,000	3,621
	1500	0,000	0,031	0,000	0,686	0,000	26,833	X	
	2000	0,015	0,125	X		X		X	
	3500	X		X		X		X	
	5000	0,031	0,754	X		X		X	
	6500	0,031	1,125	X		X		X	
	8000	0,047	0,750	X		X		X	

Da Tabela 21, nota-se como os tempos de resolução foram baixos para os problemas tridiagonais. Destaca-se o problema de dimensão 100 com $m = 5\%$, em que obteve uma ótima solução com o menor tempo registrado.

6.5.5 GMRES(m) com Soma de Vetores Esparsos

A técnica de soma de vetores esparsos, descrita na seção 3.3.1 do Capítulo 3, foi utilizada somente no processo de Arnoldi do método GMRES(m), mais precisamente na linha 8 do algoritmo 2.4 apresentado no capítulo 2. foram realizadas observações para alguns problemas e notou-se que em determinadas situações o vetor w era esparsos. no entanto, não se pode afirmar que para todos os problemas e todo o processo iterativo de Arnoldi durante a execução do GMRES(m) esta esparsidade é mantida.

6.5.5.1 Problemas dos Repositórios

Na Tabela 22 são apresentadas as quantidades de iterações e a qualidade dos resíduos obtidos pelo método GMRES(m) fazendo uso da estrutura de dados para vetores esparsos.

Tabela 22 – GMRES(m) com soma de vetores esparsos – Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
	I	R	I	R	I	R	I	R
<i>sherman1</i>	1001	C2	49	C2	12	C2	6	C2
<i>sherman4</i>	154	C2	6	C2	2	C3	2	C3
<i>fpga_dcop_42</i>	7	C2	EM		EM		EM	
<i>add20</i>	1	C3	1	C3	1	C4	EM	
<i>cavity10</i>	1	C3	1	C3	1	C3	EM	
<i>add32</i>	1	C5	EM		EM		EM	

Nesta tabela, nota-se um aumento de casos com estouros de memória, apesar de se ter algumas soluções excelentes (como é o caso do *add32* com parâmetro m de 1%) e ótimas (*add20* com m de 10%).

Na Tabela 23 são apresentados os tempos de leitura e de resolução obtidos por esse método.

Tabela 23 – GMRES(m) com soma de vetores esparsos - Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
	TLA	TRS	TLA	TRS	TLA	TRS	TLA	TRS
<i>sherman1</i>	0,000	21,869	0,000	21,515	0,000	21,638	0,015	26,008
<i>sherman4</i>	0,015	3,655	0,015	2,611	0,015	3,662	0,013	9,483
<i>fpga_dcop_42</i>	0,015	0,317	X		X		X	
<i>add20</i>	0,031	0,218	0,032	8,604	0,031	41,630	X	
<i>cavity10</i>	0,125	0,304	0,141	11,091	0,140	53,753	X	
<i>add32</i>	0,046	2,500	X		X		X	

Comparando esses tempos com os obtidos na seção anterior (veja a Tabela 15), percebe-se o aumento considerável do tempo de resolução deste problema. Note ainda a ausência nesta tabela de alguns problemas. Pois, mesmo utilizando 1% no parâmetro m , ocorreram estouro de memória para os problemas ausentes.

Acredita-se que esse aumento se deve ao uso constante de realocação dinâmica de memória (*realloc*) sempre que um novo elemento não nulo surge em determinada posição que antes da soma era nula e também que a esparsidade do vetor w pode mudar a cada iteração do GMRES(m) onde o processo de Arnoldi é realizado.

6.5.5.2 Problemas do Gerador Aleatório

Nas Tabelas 24, 25 e 26 são apresentadas as quantidades de iterações e os resíduos para os problemas de matrizes cheias, triangulares e tridiagonais, respectivamente.

Tabela 24 – GMRES(m) com soma de vetores esparsos – Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
CHEIA	100	18	C2	2	C3	6	C3	11	C3
	500	1	C3	10	C3	7	C3	14	C3
	1000	2	C3	28	C2	EM		EM	
	1500	3	C3	EM		EM		EM	
	2000	22	C3	EM		EM		EM	
	3500	3	C3	EM		EM		EM	

Tabela 25 – GMRES(m) com soma de vetores esparsos – Gerador aleatório – Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIANGULAR	100	13	C2	1	C3	4	C2	5	C3
	500	1	C3	6	C2	21	C2	EM	
	1000	6	C3	14	C3	EM		EM	
	1500	6	C2	EM		EM		EM	
	2000	31	C3	EM		EM		EM	
	3500	6	C3	EM		EM		EM	
	5000	2	C3	EM		EM		EM	

Tabela 26 – GMRES(m) com soma de vetores esparsos – Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIDIAGONAL	100	12	C2	1	C4	8	C3	2	C4
	500	1	C3	12	C3	7	C3	22	C3
	1000	3	C3	4	C3	6	C2	EM	
	1500	2	C3	2	C3	EM		EM	
	2000	11	C3	EM		EM		EM	
	8000	2	C3	EM		EM		EM	

Nestas tabelas nota-se o alto número de estouros de memória que ocorreram em todos os três tipos de problemas. Os seguintes problemas não foram apresentados nas tabelas, pois para todos os valores de m , ocorreram estouros de memória: do tipo cheia com dimensão de

5000, 6500 e 8000, do tipo triangular com dimensão 6500 e 8000 e, também, do tipo tridiagonal com dimensão de 3500, 5000 e 6500. É possível perceber que, quanto maior a dimensão e maior o valor do parâmetro m , mais se tem ocorrência desse estouro.

Nas Tabelas 27, 28 e 29 são apresentados os tempos de leitura e os tempos de resolução do método para os problemas de matrizes cheias, triangulares e tridiagonais, respectivamente.

Tabela 27 – GMRES(m) com soma de vetores esparsos – Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	TLA	TRS	TLA	TRS	TLA	TRS	TLA	TRS
CHEIA	100	0,016	0,015	0,015	0,000	0,015	0,016	0,015	0,079
	500	0,453	0,016	0,453	0,999	0,453	2,466	0,453	11,193
	1000	1,796	0,110	1,797	21,621	X		X	
	1500	4,124	0,516	X		X		X	
	2000	7,670	12,091	X		X		X	
	3500	22,604	12,216	X		X		X	

Tabela 28 – GMRES(m) com soma de vetores esparsos – Gerador aleatório – Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	TLA	TRS	TLA	TRS	TLA	TRS	TLA	TRS
TRIANGULAR	100	0,015	0,000	0,000	0,000	0,016	0,015	0,000	0,032
	500	0,218	0,015	0,235	0,522	0,219	7,154	X	
	1000	0,891	0,234	0,891	9,472	X		X	
	1500	2,046	0,762	X		X		X	
	2000	3,655	10,597	X		X		X	
	3500	11,236	10,236	X		X		X	
	5000	23,292	11,318	X		X		X	

Tabela 29 – GMRES(m) com soma de vetores esparsos – Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	TLA	TRS	TLA	TRS	TLA	TRS	TLA	TRS
TRIDIAGONAL	100	0,000	0,015	0,000	0,000	0,000	0,031	0,000	0,015
	500	0,000	0,000	0,000	0,951	0,015	2,194	0,000	16,046
	1000	0,009	0,047	0,000	2,296	0,000	14,683	X	
	1500	0,000	0,125	0,000	3,890	X		X	
	2000	0,009	2,042	X		X		X	
	8000	0,063	63,844	X		X		X	

Veja que os tempos também aumentaram muito conforme a dimensão e o parâmetro m crescem. Tome, como exemplo, o problema tridiagonal com 8000 de dimensão. Sem o uso dessa estrutura de vetores esparsos ele levava 0,750 segundos para ser resolvido. Agora, porém,

ele só é resolvido após 63,844 segundos, ou seja, a técnica da soma de vetores de esparsos piorou o desempenho do tempo computacional do GMRES. Vale citar novamente que nestes testes foi utilizada a realocação de memória em tempo real (*realloc*) para a atualização desta estrutura. E existe a possibilidade da perda de esparsidade do vetor w a cada iteração do método. Para trabalhos futuros, será necessário observar o grau de esparsidade do vetor w para cada iteração do GMRES(m) envolvendo cada passo do processo de Arnoldi para todos os problemas. além disso, deve-se utilizar a técnica de soma de vetores esparsos com uma quantidade de memória já previamente alocada.

6.5.6 GMRES(m) com Estrutura de Dados CSR e Pré-condicionador Lumped Modificado

6.4.6.1 Problemas dos Repositórios

Na Tabela 30 é apresentado o total de iterações realizadas e a qualidade do resíduo obtido para os problemas do repositório.

Tabela 30 – GMRES(m) com estrutura CSR e pré-condicionador Lumped - Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
	I	R	I	R	I	R	I	R
<i>sherman1</i>	210	C2	16	C2	7	C3	3	C2
<i>sherman4</i>	42	C2	5	C2	1	C2	109	C2
<i>fpga_dcop_42</i>	7	C2	1	C2	1	C2	EV	
<i>add20</i>	1	C3	1	C3	1	C2	EV	
<i>cavity10</i>	1	C3	1	C3	1	C3	1	C4
<i>sherman5</i>	358	C2	16	C2	4	C3	4	C3
<i>c-26</i>	7	C2	2	C3	EV		EV	
<i>add32</i>	1	C5	1	C4	1	C2	EV	
<i>sherman3</i>	53	C2	5	C2	3	C2	6	C2
<i>pres_poisson</i>	7	C3	28	C2	TL		TL	
<i>memplus</i>	1	C2	EV		EV		EV	

Na Tabela 30 é possível observar que o método convergiu na maior parte das vezes. Destaca-se os resultados de *add32*, onde conseguiu alcançar qualidades muito boas para a soluções ($m=1\%$ e $m=5\%$) e com uma única iteração somente. Para os demais problemas, ele alcançou uma qualidade aceitável. Cabe ainda outro destaque para o problema *memplus*, o qual possui, aproximadamente, 17000 variáveis e com este pré-condicionador o método convergiu em apenas 1 iteração.

Para os casos que não foram possíveis resolver, houve situações em que a variável não conseguia mais comportar o valor recebido, principalmente em *memplus*, além de dois casos em que o tempo limite de execução estourou. Vale destacar aqui que o GMRES(m) com o pré-condicionador Lumped não conseguiu resolver o problema *sherman2* para nenhum dos valores do parâmetro m e, para os problemas *b_dyn* e *b2_ss*, o valor do resíduo cresceu ao ponto de causar um estouro do limite das variáveis.

Comparando com o GMRES(m) sem o uso de pré-condicionadores (veja a Tabela 14), este conseguiu reduzir consideravelmente a quantidade de iterações, tome, por exemplo, o problema *sherman1* com m de 1%. Antes era necessário que o método realizasse o máximo (1001) de iterações para resolve-lo e agora foi preciso apenas 210 iterações.

Na Tabela 31 são apresentados os tempos de leitura e de resolução obtidos para esses testes.

Tabela 31 – GMRES(m) com estrutura CSR e pré-condicionador Lumped - Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
	TLA	TRS	TLA	TRS	TLA	TRS	TLA	TRS
<i>sherman1</i>	0,015	0,235	0,000	0,296	0,015	1,299	0,000	2,495
<i>sherman4</i>	0,015	0,062	0,015	0,141	0,015	0,266	0,016	134,634
<i>fpga_dcop_42</i>	0,016	0,000	0,015	0,047	0,015	0,400	X	
<i>add20</i>	0,032	0,015	0,031	0,437	0,031	5,514	X	
<i>cavity10</i>	0,141	0,016	0,125	0,623	0,141	7,686	0,140	36,695
<i>sherman5</i>	0,031	7,079	0,047	23,807	0,047	79,724	0,031	497,998
<i>c-26</i>	0,031	0,283	0,031	8,439	X		X	
<i>add32</i>	0,047	0,062	0,047	7,170	0,047	126,954	X	
<i>sherman3</i>	0,032	3,327	0,047	37,599	0,047	395,193	0,047	3782,181
<i>pres_poisson</i>	0,691	18,825	0,672	19201,603	X		X	
<i>memplus</i>	0,234	4,921	X		X		X	

A partir da tabela acima, e conforme será visto nas próximas tabelas, é possível perceber o quão baixo foram os tempos de construção da estrutura de dados CSR para cada um dos problemas, reforçando assim uma das vantagens da utilização desta estrutura: seu baixo custo computacional tanto para construção e, principalmente, na resolução.

Considere os problemas *pres_poisson* com m de 5% e *sherman3* com m de 15%, sendo estes dois os maiores problemas do repositório e, de acordo com a Tabela 1, com número de condição bem alto. Os tempos de resolução por interação foram, respectivamente, em torno de 686 segundos e 630 segundos

Cabe ainda destacar o resultado excelente obtido resolvendo o problema *memplus* com $m = 1\%$. Apesar de ser o maior dos problemas desse grupo, foi possível resolvê-lo em apenas 4,921 segundos, sendo necessária uma única iteração.

Comparando com Silva (2017), é possível ver a influência desse pré-condicionador. Houve casos como, por exemplo, para o problema *memplus* onde, antes resolvia com valor de m de 1% e 5% e, agora, ele só consegue resolver com 1%, com a qualidade um pouco menor, porém, com o tempo também um pouco menor. Para uma boa parte dos problemas houve redução do tempo de resolução e das iterações para os valores de 1%, 5% e 10% de m ao custo da qualidade diminuir um pouco.

6.5.6.2 Problemas do Gerador Aleatório

Nas Tabelas 32, 33 e 34 são apresentadas as iterações e qualidades dos resíduos obtidos pelo método.

Tabela 32 – GMRES(m) com estrutura CSR e pré-condicionador Lumped – Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
CHEIA	100	NC		3	C3	1	C2	1	C4
	500	2	C3	25	C3	NC		EV	
	1000	1	C3	7	C2	EV		EV	
	1500	3	C2	3	C2	EV		EV	
	2000	11	C3	EV		EV		EV	
	3500	133	C2	EV		EV		EV	
	5000	27	C3	EV		EV		EV	
	6500	14	C3	EV		EV		EV	
	8000	3	C2	EV		EV		EV	

Tabela 33 – GMRES(m) com estrutura CSR e pré-condicionador Lumped – Gerador aleatório
– Matriz triangular.

PROBLEMAS		$m = 1\%$ de n		$m = 5\%$ de n		$m = 10\%$ de n		$m = 15\%$ de n	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIANGULAR	100	15	C2	1	C2	2	C3	NC	
	500	1	C3	17	C3	6	C2	3	C3
	1000	2	C3	EV		EV		EV	
	1500	13	C3	11	C3	8	C3	EV	
	2000	20	C3	EV		EV		EV	
	3500	10	C4	EV		EV		EV	
	5000	123	C4	EV		EV		EV	
	6500	16	C2	EV		EV		EV	
	8000	802	C2	EV		EV		EV	

Tabela 34 – GMRES(m) com estrutura CSR e pré-condicionador Lumped – Gerador aleatório
– Matriz tridiagonal.

PROBLEMAS		$m = 1\%$ de n		$m = 5\%$ de n		$m = 10\%$ de n		$m = 15\%$ de n	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIDIAGONAL	100	12	C2	1	C4	45	C3	57	C4
	500	1	C3	60	C3	19	C2	8	C3
	1000	4	C4	12	C3	EV		EV	
	1500	12	C3	EV		EV		EV	
	2000	4	C3	12	C2	11	C3	EV	
	3500	8	C3	13	C3	EV		EV	
	5000	12	C3	EV		EV		EV	
	6500	12	C2	EV		EV		EV	

Destaca-se, nestas tabelas, que a melhor qualidade obtida foi C4 para alguns problemas (como o triangular com dimensão 5000 e valor de m de 1%). Houve muitos casos de convergência com qualidade razoável e, mais notadamente, com exceção de três problemas (cheio com dimensão 3500 e triangular com dimensões 5000 e 8000), todos levaram menos de uma centena de iterações para serem resolvidos. Apesar desses resultados, ainda houve casos de estouro de variável. Dentre estes problemas, o tridiagonal com dimensão de 8000 foi retirado da Tabela 34, pois ocorreu o estouro para todos os valores do parâmetro m .

Nas Tabelas 35, 36 e 37 são apresentados os tempos de leitura e os tempos de resolução do método para os problemas de matrizes cheias, triangulares e tridiagonais, respectivamente.

Tabela 35 – GMRES(m) com estrutura CSR e pré-condicionador Lumped – Gerador aleatório
– Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	TLA	TRS	TLA	TRS	TLA	TRS	TLA	TRS
CHEIA	100	X		0,015	0,000	0,018	0,001	0,015	0,000
	500	0,438	0,015	0,453	0,578	X		X	
	1000	1,796	0,040	1,797	1,250	X		X	
	1500	4,093	0,361	4,109	1,851	X		X	
	2000	7,405	2,968	X		X		X	
	3500	22,604	188,360	X		X		X	
	5000	47,410	110,202	X		X		X	
	6500	79,996	125,453	X		X		X	
	8000	122,363	51,395	X		X		X	

Tabela 36 – GMRES(m) com estrutura CSR e pré-condicionador Lumped – Gerador aleatório
– Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	TLA	TRS	TLA	TRS	TLA	TRS	TLA	TRS
TRIANGULAR	100	0,015	0,016	0,016	0,000	0,016	0,000	X	
	500	0,219	0,015	0,219	0,219	0,219	0,207	0,234	0,261
	1000	0,890	0,038	X		X		X	
	1500	2,046	0,769	2,062	3,808	2,047	11,170	X	
	2000	3,686	2,738	X		X		X	
	3500	11,294	7,113	X		X		X	
	5000	23,166	254,115	X		X		X	
	6500	39,006	72,348	X		X		X	
	8000	59,335	6746,636	X		X		X	

Tabela 37 – GMRES(m) com estrutura CSR e pré-condicionador Lumped – Gerador aleatório
– Matriz tridiagonal.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	TLA	TRS	TLA	TRS	TLA	TRS	TLA	TRS
TRIDIAGONAL	100	0,000	0,015	0,000	0,000	0,000	0,037	0,000	0,054
	500	0,015	0,000	0,000	0,140	0,000	0,281	0,000	0,469
	1000	0,016	0,000	0,016	0,250	X		X	
	1500	0,000	0,032	X		X		X	
	2000	0,015	0,016	0,016	2,684	0,000	29,228	X	
	3500	0,015	0,171	0,015	24,109	X		X	
	5000	0,031	0,734	X		X		X	
	6500	0,047	1,646	X		X		X	

Como era esperado, no tempo de leitura do arquivo de entrada, é possível ver que foi reduzindo conforme a menor quantidade de elementos não zeros. Considere, como exemplo, os problemas de dimensão 1500. Quando foi usado uma matriz cheia para esse problema foi

preciso 4 segundos, aproximadamente, para construí-lo. Porém, quando usado uma matriz triangular, ou seja, uma matriz onde um pouco mais da metade de seus elementos são não zeros, ele precisou apenas metade do tempo anterior. Por fim, ao utilizar o tridiagonal (maioria dos elementos são nulos) seu tempo de resolução foi mínimo, considerando três casas decimais.

Observe ainda que muitos dos tempos de resolução estão abaixo de 1 segundo, com exceção de alguns casos, dentre quais destaca-se o problema triangular com dimensão de 8000. Ele obteve o maior tempo, aproximadamente, 6746 segundos, mas que, ao observar seu tempo por iteração, levou por volta de 8 segundos. Essa diferença acontece principalmente pela influência da quantidade de iterações. Note que numa matriz cheia de mesma dimensão foram realizadas apenas 3 iterações, enquanto nesta foram realizadas 802.

6.5.7 GMRES(m) com Estrutura de Dados CSR e Pré-condicionador Escala Simétrica

6.5.7.1 Problemas dos Repositórios

Na Tabela 38 são apresentadas as iterações e a qualidade do resíduo obtidas após a utilização do pré-condicionador Escala Simétrica.

Tabela 38 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica - Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
	I	R	I	R	I	R	I	R
<i>sherman1</i>	484	C2	23	C2	7	C2	3	C2
<i>sherman4</i>	42	C2	4	C2	1	C2	EV	
<i>add20</i>	1	C3	1	C4	1	C3	1	C3
<i>sherman5</i>	49	C2	3	C3	21	C2	EM	
<i>c-26</i>	117	C3	108	C3	108	C3	108	C3
<i>add32</i>	1	C5	1	C4	1	C3	1	C3
<i>sherman3</i>	53	C2	4	C2	2	C3	TL	
<i>pres_poisson</i>	8	C2	TL		TL		TL	
<i>Memplus</i>	1	C4	TL		TL		TL	

Através desta tabela é possível perceber que, para a maioria dos problemas resolvidos, foi obtida uma qualidade de solução de acima de boa e com iterações abaixo da casa de centenas, com exceção do *c-26* e do *sherman1* com m de 1%.

Note ainda que dentre esses problemas a maioria em que não obteve resultados foi por causa de estouro do limite de tempo estabelecido. Além disso, os problemas *b_dyn*, *b2_ss*, *fpga_dcop_42* e *cavity10*, não foram considerados para este pré-condicionador devido a restrição de elementos nulos e/ou faltantes na diagonal principal, no arquivo de entrada. Para o problema *sherman2* o método não conseguiu resolver com *m* de 1% e, acima disso, ocorria estouro da capacidade da variável.

Na Tabela 39 são apresentados os tempos de leitura e de resolução do novo sistema obtidos para esses testes.

Tabela 39 – GMRES(*m*) com estrutura CSR e pré-condicionador Escala Simétrica - Repositórios.

PROBLEMAS	<i>m</i> = 1% de <i>n</i>		<i>m</i> = 5% de <i>n</i>		<i>m</i> = 10% de <i>n</i>		<i>m</i> = 15% de <i>n</i>	
	TLA	TRNS	TLA	TRNS	TLA	TRNS	TLA	TRNS
<i>sherman1</i>	0,007	0,616	0,007	0,477	0,008	1,386	0,007	2,528
<i>sherman4</i>	0,008	0,061	0,007	0,115	0,007	0,276	X	
<i>add20</i>	0,034	0,009	0,032	0,438	0,032	5,588	0,033	27,155
<i>sherman5</i>	0,041	1,013	0,040	4,539	0,040	433,509	X	
<i>c-26</i>	0,039	3,628	0,038	86,836	0,038	418,095	0,037	980,602
<i>add32</i>	0,047	0,047	0,047	7,233	0,047	127,323	0,046	608,069
<i>sherman3</i>	0,032	3,370	0,031	30,121	0,047	263,113	X	
<i>pres_poisson</i>	0,734	22,510	X		X		X	
<i>Memplus</i>	0,250	4,926	X		X		X	

Em primeiro lugar, vale citar que os tempos de construção do novo sistema e de obtenção da solução original foram nulos para todos os testes e, portanto, não foram mostrados nesta Tabela.

É possível observar que boa parte dos tempos de resolução foram baixos, com exceção de alguns problemas como, por exemplo, o *c-26* com *m* de 15%, o qual o método levou um pouco mais de 980 segundos para resolvê-lo. Mas, ao levar em conta seu tempo por iteração (aproximadamente, 9 segundos por iteração), ele foi superior a alguns problemas como o *sherman3* com *m* de 10%, que precisou de, aproximadamente, 131,5 segundos para cada iteração.

Cabe ressaltar ainda dois problemas em particular. Primeiro, o *sherman1* com *m* de 1%, levou 0,616 segundos para ser resolvido, porém, com consideráveis 484 iterações. Ou seja, tem-se que cada uma delas foi realizada em um tempo baixíssimo: aproximadamente 0,001 segundos. O outro exemplo é o *memplus*, sendo o maior problema do repositório, para o qual foi necessária apenas uma única iteração em apenas 4,926 segundos.

6.5.7.2 Problemas do Gerador Aleatório

Nas Tabela 40, 41 e 42 são apresentadas as quantidades de iterações e qualidades dos resíduos obtidos pelo método.

Tabela 40 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica – Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
CHEIA	100	23	C2	1	C3	NC		37	C3
	500	1	C3	3	C3	9	C3	6	C3
	1000	2	C3	50	C2	EV		EV	
	1500	8	C2	5	C2	EV		EV	
	2000	1	C3	1	C2	1	C2	EV	
	3500	14	C2	EV		EV		EV	
	5000	8	C3	EV		EV		EV	
	6500	109	C2	EV		EV		EV	
	8000	28	C2	EV		EV		EV	

Tabela 41 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica – Gerador aleatório – Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIANGULAR	100	13	C2	1	C3	8	C3	16	C4
	500	1	C3	9	C3	15	C3	5	C3
	1000	9	C2	14	C3	EV		EV	
	1500	7	C3	3	C3	EV		EV	
	2000	19	C3	EV		EV		EV	
	3500	100	C3	EV		EV		EV	
	5000	2	C2	EV		EV		EV	
	6500	5	C2	EV		EV		EV	
	8000	12	C3	EV		EV		EV	

Tabela 42 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica – Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIDIAGONAL	100	12	C2	1	C4	3	C4	5	C4
	500	1	C3	13	C3	5	C3	26	C3
	1000	4	C4	5	C3	22	C2	2	C3
	1500	6	C3	6	C3	EV		EV	
	2000	15	C3	EV		EV		EV	
	5000	39	C3	EV		EV		EV	
	6500	4	C3	EV		EV		EV	
	8000	2	C3	EV		EV		EV	

Para os problemas cheios e triangulares, o pré-condicionador Escala Simétrica se destaca por convergir em alguns problemas que o anterior (veja as Tabelas 32 e 33) não conseguiu como, por exemplo, o problema cheio com dimensão de 500 para os valores de m de 10% e 15%, e o problema triangular de dimensão 100 com m de 15%, entre outros.

Já para os problemas tridiagonais houve um aumento de casos em que este pré-condicionador não conseguiu resolver, como no problema de dimensão 3500, onde, para nenhum dos valores de m , ocorreu a convergência. Por outro lado, destaca-se aqui o problema de dimensão 8000 que antes, no pré-condicionador Lumped, não era resolvido para nenhum m , agora com 1%, além de convergir, conseguiu também um resíduo de qualidade boa.

Nas Tabelas 43, 44 e 45 são apresentados os tempos de leitura e os tempos de resolução do novo sistema gerado pelo método para os problemas de matrizes cheias, triangulares e tridiagonais, respectivamente.

Tabela 43 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica – Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	TLA	TRNS	TLA	TRNS	TLA	TRNS	TLA	TRNS
CHEIA	100	0,016	0,024	0,016	0,000	X		0,015	0,047
	500	0,453	0,000	0,484	0,079	0,437	0,498	0,437	0,705
	1000	1,796	0,077	1,797	9,413	X		X	
	1500	4,077	0,937	4,234	3,728	X		X	
	2000	7,562	0,328	7,478	1,848	7,431	5,653	X	
	3500	22,961	26,094	X		X		X	
	5000	47,186	43,106	X		X		X	
	6500	82,175	1381,504	X		X		X	
	8000	125,641	641,739	X		X		X	

Tabela 44 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica – Gerador aleatório – Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	TLA	TRNS	TLA	TRNS	TLA	TRNS	TLA	TRNS
TRIANGULAR	100	0,000	0,016	0,000	0,000	0,015	0,000	0,000	0,015
	500	0,219	0,016	0,219	0,109	0,312	0,531	0,218	0,438
	1000	0,891	0,172	0,891	1,390	X		X	
	1500	2,046	0,407	2,031	1,151	X		X	
	2000	3,687	2,707	X		X		X	
	3500	11,309	87,290	X		X		X	
	5000	23,041	5,235	X		X		X	
	6500	38,913	29,181	X		X		X	
	8000	59,704	132,192	X		X		X	

Tabela 45 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica – Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	TLA	TRNS	TLA	TRNS	TLA	TRNS	TLA	TRNS
TRIDIAGONAL	100	0,000	0,000	0,000	0,000	0,000	0,016	0,016	0,000
	500	0,000	0,000	0,000	0,031	0,000	0,078	0,000	1,530
	1000	0,000	0,015	0,015	0,082	0,000	4,136	0,000	1,656
	1500	0,015	0,015	0,015	0,475	X		X	
	2000	0,000	0,062	X		X		X	
	5000	0,031	2,334	X		X		X	
	6500	0,047	0,534	X		X		X	
	8000	0,047	0,547	X		X		X	

Os tempos de construção do novo sistema e de obtenção da solução original novamente não foram incluídos nas tabelas, pois dentre todos os problemas testados, seu valor não ultrapassou meio segundo.

Em todas as três tabelas, não houve mudanças significativas para problemas com dimensões menores que 1000. Para dimensões maiores, essas diferenças aumentaram. Por exemplo, para o problema triangular com dimensão de 3500 esse método obteve um tempo por iteração de 0,873 segundos para este pré-condicionador, enquanto para o Lumped ele precisou de 0,711 segundos para cada iteração.

No problema tridiagonal com 6500 de dimensão, o Escala Simétrica resolveu com 0,133 segundos por iteração, enquanto no pré-condicionador anterior precisou de 0,137 segundos. Apesar de ambos apresentarem um esforço próximo por iteração, o Escala Simétrica se saiu melhor, pois alcançou uma precisão melhor em menos iterações.

6.5.8 GMRES(m) com Estrutura de Dados CSR e Pré-condicionador Escala Simétrica – Variação 1

6.5.8.1 Problemas dos Repositórios

Na Tabela 46 são apresentadas as iterações e a qualidade do resíduo obtidas após a utilização do pré-condicionador Escala Simétrica – Variante 1.

Tabela 46 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 1 - Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
	I	R	I	R	I	R	I	R
<i>sherman1</i>	1001	C1	1001	C2	325	C2	107	C2
<i>sherman4</i>	140	C2	8	C2	2	C2	2	C3
<i>add20</i>	1	C3	1	C3	1	C4	1	C4
<i>sherman5</i>	3313	C1	24	C2	5	C2	2	C2
<i>add32</i>	1	C5	1	C3	1	C4	1	C4
<i>pres_poisson</i>	5	C2	EM		EM		EM	
<i>memplus</i>	1	C3	1	C4	1	C5	TL	

Na Tabela 46 destaca-se que esta variação conseguiu resultados interessantes na resolução de alguns problemas quando comparado ao Escala Simétrica. Porém, houve um aumento da quantidade de iterações em alguns casos. Como exemplos tem-se os problemas *sherman4* e *sherman5* como m de 15%.

Outra melhora ocorreu no *memplus*, o maior dos problemas, em que antes só convergia com m igual a 1%, e agora estendeu-se até 10% e ainda com um excelente resíduo.

Para os problemas *c-26* e *sherman3* essa modificação não se mostrou adequada dado que ocorreu o estouro do limite de armazenamento de variáveis e o estouro de memória, respectivamente.

Na Tabela 47 são apresentados os tempos de leitura e de resolução do novo sistema obtidos para esses testes.

Tabela 47 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 1
- Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
	TLA	TRNS	TLA	TRNS	TLA	TRNS	TLA	TRNS
<i>sherman1</i>	0,000	1,000	0,000	19,297	0,000	61,218	0,014	88,848
<i>sherman4</i>	0,000	0,202	0,000	0,234	0,015	0,544	0,015	2,459
<i>add20</i>	0,031	0,000	0,047	0,437	0,032	5,452	0,032	26,478
<i>sherman5</i>	0,047	66,438	0,031	35,655	0,047	100,459	0,031	248,001
<i>add32</i>	0,047	0,047	0,047	7,186	0,047	125,986	0,047	610,532
<i>pres_poisson</i>	0,703	13,485	X		X		X	
<i>memplus</i>	0,233	4,921	0,250	1306,052	0,234	27419,730	X	

Por meio desta tabela nota-se que os tempos de resolução, no geral, aumentaram nos problemas que levaram mais de 1 iteração para resolver. Mas, por outro lado, tem-se a convergência para alguns problemas, algo que não ocorre na versão original.

6.5.8.2 Problemas do Gerador Aleatório

Nas Tabelas 48, 49 e 50 são apresentadas as quantidades de iterações e qualidades dos resíduos obtidos pelo método.

Tabela 48 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 1
– Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
CHEIA	100	32	C2	1	C2	3	C3	38	C3
	500	1	C3	4	C3	9	C2	13	C2
	1000	51	C2	71	C3	EV		EV	
	1500	6	C3	274	C3	EV		EV	
	2000	2	C3	2	C3	2	C3	EV	
	3500	44	C3	EV		EV		EV	
	5000	20	C3	EV		EV		EV	
	6500	5	C3	EV		EV		EV	
	8000	28	C3	EV		EV		EV	

Tabela 49 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 1
– Gerador aleatório – Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIANGULAR	100	13	C2	1	C3	5	C3	39	C4
	500	1	C3	10	C3	25	C3	10	C3
	1000	8	C3	29	C3	EV		EV	
	1500	2	C2	EV		EV		EV	
	2000	7	C3	EV		EV		EV	
	3500	311	C3	EV		EV		EV	
	5000	18	C4	EV		EV		EV	

Tabela 50 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 1
– Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIDIAGONAL	100	12	C2	1	C4	2	C4	2	C4
	500	1	C3	24	C3	6	C3	4	C2
	1000	4	C2	27	C2	EV		EV	
	1500	9	C3	12	C3	EV		EV	
	2000	14	C3	EV		EV		EV	
	5000	28	C3	EV		EV		EV	
	6500	8	C2	EV		EV		EV	
	8000	11	C3	EV		EV		EV	

Nos problemas cheios, houve um aumento no número de iterações, porém, no pré-condicionador original, a matriz de dimensão 100 com valor m de 10% que não convergia, agora converge com um bom resíduo.

Para os triangulares, apesar de ocorrer um caso a mais de estouro de variável, esta variante do pré-condicionador Escala Simétrica conseguiu melhorar a qualidade do resíduo para o problema de dimensão 5000, além de reduzir as iterações realizadas.

Quando aplicando nos problemas tridiagonais, essa variante não demonstrou ter feito muita diferença, além de aumentar um pouco os casos de estouro de variável. Vale ainda destacar que os problemas do tipo triangular com dimensão 6500 e 8000 e o tridiagonal com 3500 de tamanho não foram apresentados, pois ocorreram estouros em todas as tentativas.

Nas Tabelas 51, 52 e 53 são apresentados os tempos de leitura, de construção do novo sistema e os tempos de resolução do novo sistema gerado pelo método para os problemas de matrizes cheias, triangulares e tridiagonais, respectivamente.

Tabela 51 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variação 1
– Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$			$m = 5\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
CHEIA	100	0,031	0,000	0,031	0,016	0,000	0,000
	500	0,453	0,000	0,014	0,453	0,000	0,094
	1000	1,796	0,000	1848	1,812	0,000	13,323
	1500	4,077	0,016	0,734	4,077	0,016	203,991
	2000	7,435	0,032	0,625	7,467	0,031	3,672
	3500	22,667	0,093	79,625	X		
	5000	46,911	0,172	106,545	X		
	6500	83,259	0,328	59,158	X		
	8000	126,315	0,531	645,708	X		
PROBLEMAS		$m = 10\% \text{ de } n$			$m = 15\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
CHEIA	100	0,016	0,000	0,000	0,015	0,000	0,052
	500	0,437	0,000	0,500	0,453	0,000	1,531
	1000	X			X		
	1500	X			X		
	2000	7,311	0,031	11,216	X		

Tabela 52 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variação 1
– Gerador aleatório – Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$			$m = 5\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
TRIANGULAR	100	0,015	0,000	0,000	0,015	0,000	0,000
	500	0,218	0,000	0,000	0,219	0,000	0,125
	1000	0,891	0,015	0,141	0,890	0,000	2,890
	1500	2,031	0,016	0,125	X		
	2000	3,655	0,016	1,001	X		
	3500	11,294	0,031	275,532	X		
	5000	23,088	0,094	46,777	X		
PROBLEMAS		$m = 10\% \text{ de } n$			$m = 15\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
TRIANGULAR	100	0,000	0,000	0,016	0,000	0,000	0,047
	500	0,218	0,000	0,852	0,218	0,000	0,870

Tabela 53 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variação 1
– Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		$m = 1\% \text{ de } n$			$m = 5\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
TRIDIAGONAL	100	0,000	0,000	0,000	0,015	0,000	0,000
	500	0,000	0,000	0,000	0,000	0,000	0,063
	1000	0,000	0,000	0,016	0,000	0,000	0,519
	1500	0,015	0,000	0,016	0,015	0,000	0,958
	2000	0,016	0,000	0,062	X		
	3500	X			X		
	5000	0,032	0,000	1,676	X		
	6500	0,046	0,000	1,103	X		
	8000	0,046	0,000	3,081	X		
PROBLEMAS		$m = 10\% \text{ de } n$			$m = 15\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
TRIDIAGONAL	100	0,000	0,000	0,000	0,000	0,000	0,015
	500	0,000	0,000	0,078	0,000	0,000	0,234

Note que, com essa variação, agora o tempo de construção do novo sistema se torna relevante. Principalmente nos problemas cheios e triangulares com valor do parâmetro m de 1%, apesar de serem pequenos, é possível perceber que, conforme o tamanho do problema aumenta, essa construção é mais trabalhosa.

6.5.9 GMRES(m) com Estrutura de Dados CSR e Pré-condicionador Escala Simétrica – Variante 2

6.5.9.1 Problemas dos Repositórios

Na Tabela 54 são apresentadas as iterações e a qualidade do resíduo obtidas após a utilização do pré-condicionador Escala Simétrica – Variante 2.

Tabela 54 –GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 2 - Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
	I	R	I	R	I	R	I	R
<i>sherman1</i>	469	C2	23	C2	7	C2	3	C2
<i>sherman4</i>	41	C2	4	C2	1	C2	EV	
<i>fpga_dcop_42</i>	5	C2	1221	C2	12	C2	1	C2
<i>add20</i>	1	C3	1	C4	1	C3	1	C3
<i>cavity10</i>	1	C3	1	C3	1	C3	1	C4
<i>sherman5</i>	74	C2	3	C2	17	C2	EM	
<i>add32</i>	1	C5	1	C4	1	C4	1	C3
<i>sherman3</i>	53	C2	4	C2	2	C3	TL	
<i>pres_poisson</i>	7	C3	12	C2	TL		TL	
<i>memplus</i>	1	C5	1	C3	TL		TL	

Para esses problemas do repositório, essa variante 2 demonstrou um desempenho melhor que o pré-condicionador anterior. Para o *sherman3*, enquanto a variante anterior não conseguiu resolvê-lo, aqui foi obtido o mesmo resultado que o Escala Simétrico original.

Ainda, em relação ao pré-condicionador original, esta variante também conseguiu vantagem ao resolver os dois maiores problemas, *pres_poisson* e *memplus* com o valor de $m=5\%$, apesar da variante 1 também ter conseguido para o *memplus* em m igual a 10%.

Por fim, destaca-se que este pré-condicionador conseguiu fazer o método convergir para os problemas *fpga_dcop_42* e *cavity10*, algo que nem o original nem a variante 1 conseguiram.

Na Tabela 55 são apresentados os tempos de leitura e de resolução do novo sistema obtidos para esses testes.

Tabela 55 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 2 - Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
	TLA	TRNS	TLA	TRNS	TLA	TRNS	TLA	TRNS
<i>sherman1</i>	0,000	0,547	0,015	0,425	0,015	1,296	0,000	2,499
<i>sherman4</i>	0,000	0,063	0,012	0,109	0,015	0,266	X	
<i>fpga_dcop_42</i>	0,015	0,000	0,016	47,948	0,015	4,743	0,000	1,815
<i>add20</i>	0,032	0,000	0,047	0,422	0,031	5,456	0,047	26,587
<i>cavity10</i>	0,140	0,022	0,140	0,609	0,125	7,717	0,140	36,476
<i>sherman5</i>	0,047	1,474	0,047	4,452	0,031	339,563	X	
<i>add32</i>	0,047	0,063	0,047	7,192	0,047	126,551	0,047	630,761
<i>sherman3</i>	0,032	3,356	0,031	29,976	0,047	265,856	X	
<i>pres_poisson</i>	0,684	18,902	0,688	8254,572	X		X	
<i>memplus</i>	0,234	4,936	0,250	1330,315	X		X	

Na Tabela 55, observa-se que os tempos de resolução foram bem próximos dos obtidos pelo pré-condicionador original. Porém, conforme o problema fica maior, se nota uma ligeira vantagem dessa variante com o registro de tempos menores.

Note ainda que o tempo de construção do novo sistema e de obtenção da solução original foram excluídos, pois essa variante do Escala Simétrica obteve todos esses tempos nulos, considerando três casas decimais.

6.5.9.2 Problemas do Gerador Aleatório

Nas Tabelas 56, 57 e 58 são apresentadas as quantidades de iterações e qualidades dos resíduos obtidos pelo método.

Tabela 56 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 2 – Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
CHEIA	100	NC		3	C3	1	C2	1	C4
	500	2	C3	14	C3	27	C2	EV	
	1000	1	C3	24	C2	EV		EV	
	1500	14	C3	8	C3	EV		EV	
	2000	21	C2	EV		EV		EV	
	3500	34	C2	EV		EV		EV	
	5000	14	C2	EV		EV		EV	
	6500	9	C2	EV		EV		EV	
	8000	18	C3	EV		EV		EV	

Tabela 57 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 2 – Gerador aleatório – Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIANGULAR	100	24	C2	2	C3	1	C4	NC	
	500	1	C2	13	C3	18	C3	240	C3
	1000	1	C3	35	C2	EV		EV	
	1500	17	C4	166	C2	EV		EV	
	2000	6	C2	EV		EV		EV	
	3500	26	C4	EV		EV		EV	
	5000	17	C3	EV		EV		EV	
	8000	12	C2	EV		EV		EV	

Tabela 58 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 2
– Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		$m = 1\%$ de n		$m = 5\%$ de n		$m = 10\%$ de n		$m = 15\%$ de n	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIDIAGONAL	100	12	C2	1	C4	27	C2	42	C2
	500	1	C3	67	C2	121	C3	115	C3
	1500	39	C3	EV		EV		EV	
	2000	2	C3	2	C4	EV		EV	
	3500	3	C3	88	C2	EV		EV	
	5000	2	C4	EV		EV		EV	
	6500	5	C3	EV		EV		EV	

Nestes resultados, destacam-se dois casos que o pré-condicionador original não conseguiu resolver e esta variante conseguiu: problema de dimensão 100 cheio com m de 15% e o problema de dimensão 3500 tridiagonal com valores de m em 1% e 5%. Nos problemas triangulares houve um aumento de ocorrência de soluções com resíduos excelentes. Note também que houve redução da quantidade de iterações realizadas, principalmente nos problemas cheios. Os dados dos problemas do tipo triangular com dimensão de 6500 e tridiagonal com dimensão 1000 e 8000 foram omitidos das tabelas pois em todos os casos ocorreram estouros de variáveis.

Nas Tabelas 59, 60 e 61 são apresentados os tempos de leitura, de construção do novo sistema e os tempos de resolução do novo sistema gerado pelo método para os problemas de matrizes cheias, triangulares e tridiagonais, respectivamente.

Tabela 59 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 2
– Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\%$ de n			$m = 5\%$ de n		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
CHEIA	100	X			0,016	0,000	0,000
	500	0,437	0,000	0,016	0,438	0,000	0,328
	1000	1,796	0,000	0,047	1,812	0,000	4,389
	1500	4,078	0,015	1,647	4,077	0,016	5,976
	2000	7,326	0,031	6,540	X		
	3500	22,591	0,110	61,912	X		
	5000	46,380	0,234	74,245	X		
	6500	81,762	0,375	106,605	X		
	8000	120,291	0,593	401,989	X		
PROBLEMAS		$m = 10\%$ de n			$m = 15\%$ de n		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
CHEIA	100	0,015	0,000	0,000	0,031	0,000	0,001
	500	0,437	0,000	1,474	X		

Tabela 60 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 2
– Gerador aleatório – Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$			$m = 5\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
TRIANGULAR	100	0,015	0,000	0,016	0,000	0,000	0,000
	500	0,219	0,000	0,000	0,219	0,016	0,156
	1000	0,891	0,000	0,030	0,907	0,000	3,472
	1500	2,047	0,015	1,001	2,047	0,015	63,197
	2000	3,655	0,016	0,856	X		
	3500	11,263	0,047	22,692	X		
	5000	23,026	0,109	44,125	X		
	6500	X			X		
	8000	59,455	0,297	132,122	X		
PROBLEMAS		$m = 10\% \text{ de } n$			$m = 15\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
TRIANGULAR	100	0,015	0,000	0,000	X		
	500	0,219	0,000	0,607	0,218	0,016	20,897

Tabela 61 – GMRES(m) com estrutura CSR e pré-condicionador Escala Simétrica Variante 2
– Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		$m = 1\% \text{ de } n$			$m = 5\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
TRIDIAGONAL	100	0,000	0,000	0,001	0,000	0,000	0,000
	500	0,000	0,000	0,000	0,000	0,000	0,156
	1000	X			X		
	1500	0,000	0,000	0,094	X		
	2000	0,016	0,000	0,000	0,000	0,000	0,453
	3500	0,031	0,000	0,052	0,016	0,000	159,637
	5000	0,031	0,000	0,125	X		
	6500	0,078	0,000	0,735	X		
PROBLEMAS		$m = 10\% \text{ de } n$			$m = 15\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
TRIDIAGONAL	100	0,001	0,000	0,016	0,000	0,000	0,031
	500	0,006	0,000	1,750	0,000	0,000	6,711

Não houve muitas diferenças com relação aos tempos obtidos em relação a primeira variante. Porém, observa-se uma tendência de que, conforme o parâmetro m cresce, o tempo de resolução fica maior quando comparado aos tempos dos pré-condicionador original e da

variante 1. Por exemplo, o problema tridiagonal com dimensão de 500 com m de 15%, o qual levou 6,711 segundos para ser resolvido, enquanto no Escala Simétrica, sem modificações, levou apenas 1,530 segundos e, na variante 1, apenas 0,234 segundos.

6.5.10 GMRES(m) com Estrutura de Dados CSR e Pré-condicionador ILU(0)

6.5.10.1 Problemas do Repositório

Na Tabela 62 são apresentadas as iterações e a qualidade do resíduo obtidas após a utilização do pré-condicionador ILU(0).

Tabela 62 – GMRES(m) com estrutura CSR e pré-condicionador ILU(0) - Repositórios.

PROBLEMAS	$m = 1\%$ de n		$m = 5\%$ de n		$m = 10\%$ de n		$m = 15\%$ de n	
	I	R	I	R	I	R	I	R
<i>sherman1</i>	9	C2	2	C3	EM		EM	
<i>sherman2</i>	1081	C2	EV		EV		EV	
<i>sherman4</i>	13	C4	50	C3	EV		EV	
<i>add20</i>	1	C3	1	C4	1	C4	1	C4
<i>sherman5</i>	2	C3	EV		EV		EV	
<i>c-26</i>	1	C3	1	C3	1	C3	1	C3
<i>add32</i>	1	C5	1	C4	1	C4	1	C4
<i>sherman3</i>	5	C3	EM		EM		EM	
<i>pres_poisson</i>	1	C3	1	C3	1	C3	1	C3
<i>memplus</i>	1	C4	1	C4	1	C3	TL	

Da Tabela 62 nota-se que houve muitas soluções com qualidades classificadas como boas e ótimas, com destaque para o problema *add32* com $m = 1\%$, que obteve um resíduo de qualidade excelente.

Observe ainda que, para a maioria dos problemas, foi necessário apenas uma única iteração, exceto algumas exceções, como o *sherman1*, *sherman4*, *sherman5* e *sherman3*, que precisaram de um pouco mais de iterações.

Destaca-se que o método GMRES(m) com este pré-condicionador convergiu para o problema *sherman2* com $m = 1\%$, algo que nenhum outro pré-condicionador ou método conseguiu.

Na Tabela 63 são apresentados os tempos de leitura, de fatoração da matriz do problema e de resolução do sistema obtidos para esses testes.

Tabela 63 – GMRES(m) com estrutura CSR e pré-condicionador ILU(0) - Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$			$m = 5\% \text{ de } n$		
	TLA	TF	TRS	TLA	TF	TRS
<i>sherman1</i>	0,000	0,000	0,016	0,000	0,000	0,047
<i>sherman2</i>	0,031	0,015	4,234	X		
<i>sherman4</i>	0,000	0,000	0,031	0,000	0,000	1,531
<i>add20</i>	0,031	0,016	0,015	0,047	0,000	0,453
<i>sherman5</i>	0,031	0,015	0,093	X		
<i>c-26</i>	0,031	0,000	0,000	0,032	0,000	0,015
<i>add32</i>	0,047	0,000	0,082	0,047	0,000	7,233
<i>sherman3</i>	0,032	0,000	0,374	X		
<i>pres_poisson</i>	0,687	0,172	0,021	0,687	0,187	0,019
<i>memplus</i>	0,250	0,516	5,184	0,250	0,516	1290,022
PROBLEMAS	$m = 10\% \text{ de } n$			$m = 15\% \text{ de } n$		
	TLA	TF	TRS	TLA	TF	TRS
<i>add20</i>	0,032	0,015	5,499	0,032	0,015	26,588
<i>c-26</i>	0,047	0,000	0,000	0,047	0,000	0,006
<i>add32</i>	0,047	0,000	126,767	0,063	0,000	607,873
<i>pres_poisson</i>	0,672	0,188	0,031	0,671	0,188	0,031
<i>memplus</i>	0,235	0,531	27504,857	X		

Note que os tempos de resolução da maioria dos problemas presentes na Tabela 63 foram baixos, conseguindo em alguns casos ser mais rápido que o Escala Simétrica, como no problema *pres_poisson* com parâmetro $m = 1\%$, onde convergiu em apenas 0,021 segundos, e no Escala Simétrica, foram necessários mais de 20 segundos para resolvê-lo.

Vale ressaltar, ainda, que o tempo de fatoração não exerceu tanta influência no método para estes problemas dos repositórios. Porém, a seguir é possível observar melhor essa influência em diferentes tipos de problemas.

6.5.10.2 Problemas do Gerador Aleatório

Nas Tabelas 64, 65 e 66 são apresentadas as quantidades de iterações e qualidades dos resíduos obtidos pelo método.

Tabela 64 – GMRES(m) com estrutura CSR e pré-condicionador ILU(0) – Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
CHEIA	100	1	C4	1	C4	1	C4	1	C4
	500	1	C3	1	C3	1	C3	1	C3
	1000	1	C3	1	C3	1	C3	1	C3
	1500	1	C3	1	C3	1	C3	1	C3
	2000	1	C3	1	C3	1	C3	1	C3

Tabela 65 – GMRES(m) com estrutura CSR e pré-condicionador ILU(0) – Gerador aleatório – Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIANGULAR	100	1	C4	1	C4	1	C4	1	C4
	500	1	C4	1	C4	1	C4	1	C4
	1000	1	C3	1	C3	1	C3	1	C3
	1500	1	C3	1	C3	1	C3	1	C3
	2000	1	C3	1	C3	1	C3	1	C3
	3500	1	C3	1	C3	1	C3	1	C3

Tabela 66 – GMRES(m) com estrutura CSR e pré-condicionador ILU(0) – Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIDIAGONAL	100	1	C4	1	C4	1	C4	1	C4
	500	1	C4	1	C4	1	C4	1	C4
	1000	1	C4	1	C4	1	C4	1	C4
	1500	1	C3	1	C3	1	C3	1	C3
	2000	1	C3	1	C3	1	C3	1	C3
	3500	1	C3	1	C3	1	C3	1	C3
	5000	1	C3	1	C3	1	C3	1	C3
	6500	1	C3	1	C3	1	C3	1	C3
	8000	1	C3	1	C3	1	C3	1	C3

Perceba que em todos os problemas do gerador aleatório, independentemente do valor do parâmetro m , as iterações e resíduos não mudaram para sistema considerado. Note ainda que, até o presente momento nenhum outro pré-condicionador ou método conseguiu obter tantos resíduos de qualidade ótima para os problemas desse gerador.

Cabe ressaltar que todos precisaram de apenas uma única iteração para convergirem. Os únicos problemas para os quais o método não convergiu foram os do tipo cheio com dimensão

de 3500 em diante e os triangulares com 5000, 6500 e 8000 de dimensão, pois estourou o tempo limite de 10 horas estabelecido.

Nas Tabelas 67, 68 e 69 são apresentados os tempos de leitura, de fatoração e os tempos de resolução do sistema gerado pelo método para os problemas de matrizes cheias, triangulares e tridiagonais, respectivamente.

Tabela 67 – GMRES(m) com estrutura CSR e pré-condicionador ILU(0) – Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$			$m = 5\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TF	TRS	TLA	TF	TRS
CHEIA	100	0,016	0,031	0,004	0,016	0,031	0,004
	500	0,453	22,462	0,000	0,453	19,480	0,000
	1000	1,796	308,952	0,016	1,797	310,540	0,032
	1500	4,092	1575,425	0,046	4,093	1594,280	0,047
	2000	7,499	4927,842	0,078	7,327	4901,072	0,078
PROBLEMAS		$m = 10\% \text{ de } n$			$m = 15\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TF	TRS	TLA	TF	TRS
CHEIA	100	0,016	0,031	0,003	0,015	0,031	0,004
	500	0,438	19,495	0,000	0,438	19,214	0,000
	1000	1,796	308,908	0,021	1,796	310,242	0,025
	1500	4,109	1573,538	0,047	4,467	1557,177	0,046
	2000	7,326	4948,516	0,078	7,404	4898,099	0,086

Tabela 68 – GMRES(m) com estrutura CSR e pré-condicionador ILU(0) – Gerador aleatório – Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$			$m = 5\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TF	TRS	TLA	TF	TRS
TRIANGULAR	100	0,015	0,000	0,000	0,000	0,016	0,000
	500	0,219	3,874	0,000	0,234	3,874	0,016
	1000	0,891	62,907	0,015	0,891	63,469	0,000
	1500	2,046	320,253	0,017	2,062	319,002	0,019
	2000	3,656	1091,690	0,453	3,749	1012,112	0,046
	3500	11,325	9489,611	0,125	11,310	9588,238	0,156
PROBLEMAS		$m = 10\% \text{ de } n$			$m = 15\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TF	TRS	TLA	TF	TRS
TRIANGULAR	100	0,000	0,015	0,000	0,015	0,016	0,000
	500	0,219	3,890	0,007	0,219	3,890	0,000
	1000	0,890	63,048	0,000	0,890	63,565	0,016
	1500	2,062	323,005	0,031	2,047	319,712	0,016
	2000	3,734	1016,309	0,047	3,780	1018,602	0,031
	3500	11,404	9488,979	0,136	11,310	9428,975	0,141

Tabela 69 – GMRES(m) com estrutura CSR e pré-condicionador ILU(0) – Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		$m = 1\% \text{ de } n$			$m = 5\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TF	TRS	TLA	TF	TRS
TRIDIAGONAL	100	0,000	0,000	0,000	0,000	0,000	0,000
	500	0,000	0,000	0,000	0,000	0,000	0,007
	1000	0,000	0,000	0,000	0,010	0,000	0,000
	1500	0,000	0,000	0,016	0,000	0,000	0,000
	2000	0,015	0,000	0,000	0,015	0,000	0,000
	3500	0,016	0,000	0,000	0,015	0,000	0,016
	5000	0,047	0,000	0,000	0,031	0,000	0,000
	6500	0,032	0,000	0,000	0,031	0,000	0,000
	8000	0,063	0,000	0,000	0,062	0,000	0,004
PROBLEMAS		$m = 10\% \text{ de } n$			$m = 15\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TF	TRS	TLA	TF	TRS
TRIDIAGONAL	100	0,000	0,000	0,002	0,000	0,000	0,000
	500	0,000	0,000	0,000	0,000	0,000	0,000
	1000	0,000	0,000	0,000	0,000	0,000	0,000
	1500	0,000	0,000	0,015	0,000	0,000	0,015
	2000	0,016	0,000	0,000	0,016	0,000	0,000
	3500	0,032	0,000	0,000	0,016	0,000	0,015
	5000	0,032	0,000	0,013	0,031	0,000	0,000
	6500	0,031	0,000	0,017	0,047	0,000	0,007
	8000	0,047	0,000	0,016	0,047	0,000	0,017

Nessas tabelas é possível perceber que o tempo de fatoração é determinado pela quantidade de elementos não nulos de uma matriz. Nos problemas cheios, onde todos os elementos são nulos, o tempo para realizar a fatoração foi muito alto. Veja o problema de dimensão 2000 que levou, aproximadamente, 4948 segundos para esse processo.

Conforme a quantidade de elementos não nulos diminui, esse tempo também é reduzido. O problema de dimensão 1500 agora precisou de apenas 320 segundos, aproximadamente. Observando os problemas triangulares, nota-se que esses tempos foram todos zeros. Ou seja, quanto mais esparsa for a matriz a ser fatorada, menos esforço será necessário.

Note ainda os tempos de resolução. Nenhum deles ultrapassou 1 segundo, mostrando que o processo mais custoso deste pré-condicionador se encontra na fatoração que é feita apenas uma vez e não em tentar resolvê-la.

6.5.11 GMRES(m) com Estrutura de Dados CSR e Pré-condicionador Diagonal/Escala Simétrica

6.5.11.1 Problemas dos Repositórios

Na Tabela 70 são apresentadas as iterações e a qualidade do resíduo obtidas após a utilização do pré-condicionador Diagonal/Escala Simétrica.

Tabela 70 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/Escala Simétrica - Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
	I	R	I	R	I	R	I	R
<i>sherman1</i>	214	C2	23	C2	7	C2	3	C2
<i>sherman4</i>	52	C2	5	C3	1	C2	84	C2
<i>add20</i>	1	C3	1	C3	EV		EV	
<i>sherman5</i>	25	C3	2	C2	EV		EV	
<i>c-26</i>	1	C3	1	C3	1	C3	1	C3
<i>add32</i>	1	C5	1	C3	EV		EV	
<i>sherman3</i>	57	C2	6	C2	4	C2	6	C3
<i>pres_poisson</i>	8	C2	TL		TL		TL	
<i>memplus</i>	1	C3	EV		EV		EV	

Na Tabela 70, o método aplicado ao problema *add32* com valor de m em 1% destacou-se por convergir com uma iteração e ainda com um resíduo de excelente qualidade. Outro problema com bons resultados foi o *c-26* que, em todos os valores de m , realizou apenas uma iteração com qualidade boa, o que não ocorreu com outros pré-condicionadores como a Escala Simétrica e o Lumped (veja as Tabelas 30 e 38).

Na Tabela 71 são apresentados os tempos de leitura e de resolução do novo sistema obtidos para esses testes.

Tabela 71 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/Escala Simétrica - Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
	TLA	TRNS	TLA	TRNS	TLA	TRNS	TLA	TRNS
<i>sherman1</i>	0,000	0,234	0,000	0,439	0,000	1,312	0,012	2,488
<i>sherman4</i>	0,000	0,078	0,008	0,140	0,015	0,265	0,015	103,486
<i>add20</i>	0,031	0,000	0,031	0,438	X		X	
<i>sherman5</i>	0,046	0,493	0,031	2,968	X		X	
<i>c-26</i>	0,047	0,000	0,031	0,000	0,031	0,000	0,032	0,000
<i>add32</i>	0,047	0,062	0,047	7,373	X		X	
<i>sherman3</i>	0,047	3,609	0,047	45,190	0,047	525,299	0,047	3823,580
<i>pres_poisson</i>	0,672	21,601	X		X		X	
<i>memplus</i>	0,235	4,952	X		X		X	

Para os tempos de resolução (TRNS) não houve grandes mudanças, sendo muito próximos dos tempos obtidos pelo Lumped, com exceção do problema *c-26*. Neste caso, para todos os valores de m o método conseguiu chegar à solução em zero milissegundos.

6.5.11.2 Problemas do Gerador Aleatório

Nas Tabelas 72, 73 e 74 são apresentadas as quantidades de iterações e qualidades dos resíduos obtidos pelo método.

Tabela 72 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/Escala Simétrica – Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
CHEIA	100	55	C2	1	C2	3	C3	3	C3
	500	1	C3	8	C2	8	C3	192	C3
	1000	EV		EV		EV		EV	
	1500	1	C2	1	C3	2	C3	EV	
	2000	13	C3	12	C2	EV		EV	
	3500	12	C3	EV		EV		EV	

Tabela 73 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/Escala Simétrica – Gerador aleatório – Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIANGULAR	100	4	C2	1	C3	4	C3	15	C3
	500	1	C4	2	C2	2	C3	3	C3
	1000	28	C3	9	C3	EV		EV	
	1500	25	C4	586	C3	EV		EV	
	2000	6	C3	EV		EV		EV	
	3500	41	C3	EV		EV		EV	
	5000	2	C2	EV		EV		EV	

Tabela 74 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/Escala Simétrica – Gerador aleatório – Matriz tridiaogonal.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIDIAGONAL	100	3	C2	1	C4	NC		EV	
	500	1	C3	32	C4	EV		EV	
	1000	3	C2	8	C2	14	C3	EV	
	1500	15	C3	30	C3	EV		EV	
	2000	5	C4	9	C2	EV		EV	
	3500	8	C2	EV		EV		EV	
	5000	9	C3	EV		EV		EV	
	6500	20	C3	EV		EV		EV	
	8000	3	C3	EV		EV		EV	

Através desses resultados, percebe-se que houve um aumento de casos de estouro de variáveis para os problemas desses geradores, e como não houve mudanças para os problemas cheio com dimensão 5000, 6500 e 8000 e o triangular de 6500 e 8000, os resultados dos mesmos não foram apresentados. Alguns problemas, como o triangular com dimensão de 1500, conseguiram aumentar a qualidade de seus resíduos.

Nas Tabelas 75, 76 e 77 são apresentados os tempos de leitura, de aplicação do pré-condicionador Diagonal, de construção do novo sistema e os tempos de resolução desse novo sistema gerado pelo método para os problemas de matrizes cheias, triangulares e tridiaogonais, respectivamente.

Tabela 75 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/Escala Simétrica – Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$				$m = 5\% \text{ de } n$			
TIPO	DIMENSÃO	TLA	TAPD	TCNS	TRNS	TLA	TAPD	TCNS	TRNS
CHEIA	100	0,016	0,000	0,000	0,062	0,015	0,000	0,000	0,006
	500	0,453	0,000	0,000	0,020	0,453	0,000	0,000	0,187
	1000	X				X			
	1500	4,077	0,016	0,016	0,125	4,093	0,000	0,015	0,750
	2000	7,571	0,012	0,029	4,142	7,429	0,012	0,029	22,523
	3500	23,109	0,037	0,087	22,248	X			
PROBLEMAS		$m = 10\% \text{ de } n$				$m = 15\% \text{ de } n$			
TIPO	DIMENSÃO	TLA	TAPD	TCNS	TRNS	TLA	TAPD	TCNS	TRNS
CHEIA	100	0,016	0,000	0,000	0,016	0,016	0,000	0,000	0,000
	500	0,437	0,000	0,000	0,438	0,453	0,000	0,000	22,639
	1000	X				X			
	1500	4,077	0,016	0,015	4,187	X			

Tabela 76 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/Escala Simétrica – Gerador aleatório – Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$				$m = 5\% \text{ de } n$			
TIPO	DIMENSÃO	TLA	TAPD	TCNS	TRNS	TLA	TAPD	TCNS	TRNS
TRIANGULAR	100	0,015	0,000	0,000	0,000	0,015	0,000	0,000	0,000
	500	0,219	0,000	0,015	0,004	0,235	0,000	0,000	0,015
	1000	0,906	0,000	0,000	0,515	0,891	0,000	0,000	0,906
	1500	2,031	0,016	0,000	1,468	2,046	0,000	0,016	223,637
	2000	3,671	0,015	0,016	0,857	X			
	3500	11,294	0,016	0,047	35,776	X			
	5000	23,150	0,032	0,093	5,245	X			
PROBLEMAS		$m = 10\% \text{ de } n$				$m = 15\% \text{ de } n$			
TIPO	DIMENSÃO	TLA	TAPD	TCNS	TRNS	TLA	TAPD	TCNS	TRNS
TRIANGULAR	100	0,015	0,000	0,000	0,000	0,000	0,000	0,000	0,016
	500	0,218	0,000	0,000	0,073	0,218	0,000	0,000	0,266

Tabela 77 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/Escala Simétrica – Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		$m = 1\% \text{ de } n$				$m = 5\% \text{ de } n$			
TIPO	DIMENSÃO	TLA	TAPD	TCNS	TRNS	TLA	TAPD	TCNS	TRNS
TRIDIAGONAL	100	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,002
	500	0,000	0,000	0,000	0,015	0,004	0,000	0,000	0,062
	1000	0,000	0,000	0,000	0,016	0,000	0,000	0,000	0,156
	1500	0,015	0,000	0,000	0,032	0,000	0,000	0,000	2,390
	2000	0,015	0,000	0,000	0,031	0,000	0,000	0,000	2,019
	3500	0,016	0,000	0,000	0,172	X			
	5000	0,032	0,000	0,000	0,538	X			
	6500	0,047	0,000	0,000	2,759	X			
	8000	0,047	0,000	0,000	0,831	X			
PROBLEMAS		$m = 10\% \text{ de } n$				$m = 15\% \text{ de } n$			
TIPO	DIMENSÃO	TLA	TAPD	TCNS	TRNS	TLA	TAPD	TCNS	TRNS
TRIDIAGONAL	100	X				X			
	500	X				X			
	1000	0,015	0,000	0,000	2,624	X			

Com este pré-condicionador, os tempos de aplicação do pré-condicionador Diagonal e o de construção do novo sistema, se tornaram mais evidentes, principalmente nos problemas triangulares. Nos tridiagonais esses tempos foram nulos, considerando três casas decimais, enquanto nos problemas cheios, os tempos começaram a se destacar nos problemas de dimensão 1500, indicando que, para problemas maiores, esse tempo cresce também.

6.5.12 GMRES(m) com Estrutura de Dados CSR e Pré-condicionador Diagonal/ILU(0)

6.5.12.1 Problemas dos Repositórios

Na Tabela 78 são apresentadas as iterações e a qualidade do resíduo obtidas após a utilização do pré-condicionador Diagonal/ILU(0).

Tabela 78 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/ILU(0) - Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
	I	R	I	R	I	R	I	R
<i>add20</i>	1	C3	1	C4	1	C4	1	C4
<i>c-26</i>	1	C3	1	C3	1	C3	1	C3
<i>add32</i>	1	C5	1	C4	1	C4	1	C4
<i>pres_poisson</i>	1	C3	1	C3	1	C3	1	C3
<i>memplus</i>	1	C4	1	C4	TL		TL	

Nesta tabela, observa-se o mesmo comportamento visto no pré-condicionador ILU(0) (veja a Tabela 62), onde houveram muitos casos de resíduos ótimos e bons e todos realizando apenas uma única iteração.

Porém, após o uso deste pré-condicionador, houveram muitos problemas que não puderam ser aplicados por causa de problemas com elementos faltantes e/ou nulos na diagonal principal, estouros de memória e estouros de variáveis.

Na Tabela 79 são apresentados os tempos de leitura, de fatoração e de resolução do novo sistema obtidos para esses testes.

Tabela 79 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/ILU(0) - Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$			$m = 5\% \text{ de } n$		
	TLA	TF	TRS	TLA	TF	TRS
<i>add20</i>	0,032	0,015	0,000	0,031	0,016	0,453
<i>c-26</i>	0,046	0,000	0,000	0,047	0,000	0,000
<i>add32</i>	0,047	0,000	0,078	0,047	0,000	7,295
<i>pres_poisson</i>	0,671	0,188	0,015	0,671	0,203	0,016
<i>memplus</i>	0,297	0,437	5,092	0,234	0,453	1300,042
PROBLEMAS	$m = 10\% \text{ de } n$			$m = 15\% \text{ de } n$		
	TLA	TF	TRS	TLA	TF	TRS
<i>add20</i>	0,031	0,016	5,483	0,031	0,016	26,560
<i>c-26</i>	0,031	0,000	0,016	0,032	0,000	0,000
<i>add32</i>	0,047	0,015	127,642	0,047	0,016	611,736
<i>pres_poisson</i>	0,688	0,187	0,015	0,671	0,203	0,047

Os tempos foram semelhantes ao pré-condicionador ILU(0), com o tempo de fatoração baixo para esses problemas. Da mesma forma, os problemas como o *memplus* e o *add32* levaram mais tempo para serem resolvidos, conforme o parâmetro m aumenta. O tempo de aplicação do pré-condicionador Diagonal foi excluído desta tabela, pois em todos os casos ele foi muito próximo de zero milissegundos.

6.5.12.2 Problemas do Gerador Aleatório

Nas Tabelas 80, 81 e 82 são apresentadas as quantidades de iterações e qualidades dos resíduos obtidos pelo método.

Tabela 80 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/ILU(0) – Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
CHEIA	100	1	C4	1	C4	1	C4	1	C4
	500	1	C3	1	C3	1	C3	1	C3
	1000	1	C3	1	C3	1	C3	1	C3
	1500	1	C3	1	C3	1	C3	1	C3
	2000	1	C3	1	C3	1	C3	1	C3

Tabela 81 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/ILU(0) – Gerador aleatório – Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIANGULAR	100	1	C4	1	C4	1	C4	1	C4
	500	1	C4	1	C4	1	C4	1	C4
	1000	1	C3	1	C3	1	C3	1	C3
	1500	1	C3	1	C3	1	C3	1	C3
	2000	1	C3	1	C3	1	C3	1	C3
	3500	1	C3	1	C3	1	C3	1	C3
	5000	1	C3	1	C3	EV		EV	

Tabela 82 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/ILU(0) – Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIDIAGONAL	100	1	C4	1	C4	1	C4	1	C4
	500	1	C4	1	C4	1	C4	1	C4
	1000	1	C4	1	C4	1	C4	1	C4
	1500	1	C3	1	C3	1	C3	1	C3
	2000	1	C3	1	C3	1	C3	1	C3
	3500	1	C3	1	C3	1	C3	1	C3
	5000	1	C3	1	C3	1	C3	1	C3
	6500	1	C3	1	C3	1	C3	1	C3
	8000	1	C3	1	C3	1	C3	1	C3

Comparando esses resultados com os do pré-condicionador ILU(0) (veja as Tabelas 64, 65 e 66), percebe-se que foram exatamente os mesmos. Isso mostra que o uso do pré-

condicionador Diagonal/ILU(0) não influenciou nos resultados referentes aos problemas do gerador aleatório.

Nas Tabelas 83, 84 e 85 são apresentados os tempos de leitura, de aplicação do pré-condicionador Diagonal/ILU(0), de fatoração e os tempos de resolução gerado pelo método para os problemas de matrizes cheias, triangulares e tridiagonais, respectivamente.

Tabela 83 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/ILU(0) – Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$				$m = 5\% \text{ de } n$			
TIPO	DIMENSÃO	TLA	TAPD	TF	TRS	TLA	TAPD	TF	TRS
CHEIA	100	0,016	0,000	0,031	0,000	0,015	0,000	0,031	0,000
	500	0,438	0,000	17,558	0,000	0,437	0,000	17,544	0,016
	1000	1,922	0,000	276,270	0,031	1,844	0,000	288,057	0,031
	1500	4,077	0,015	1390,107	0,047	4,077	0,016	1389,718	0,059
	2000	7,358	0,031	4390,219	0,094	7,405	0,031	4378,560	0,093
PROBLEMAS		$m = 10\% \text{ de } n$				$m = 15\% \text{ de } n$			
TIPO	DIMENSÃO	TLA	TAPD	TF	TRS	TLA	TAPD	TF	TRS
CHEIA	100	0,016	0,000	0,031	0,003	0,016	0,000	0,031	0,005
	500	0,438	0,015	17,512	0,015	0,453	0,000	17,856	0,000
	1000	1,796	0,000	278,225	0,015	1,780	0,016	279,608	0,016
	1500	4,077	0,016	1391,733	0,047	4,124	0,016	1404,850	0,052
	2000	7,357	0,016	4375,732	0,094	7,373	0,031	4374,484	0,078

Tabela 84 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/ILU(0) – Gerador aleatório – Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$				$m = 5\% \text{ de } n$			
TIPO	DIMENSÃO	TLA	TAPD	TF	TRS	TLA	TAPD	TF	TRS
TRIANGULAR	100	0,000	0,000	0,016	0,000	0,015	0,000	0,000	0,000
	500	0,218	0,000	3,718	0,000	0,219	0,000	3,718	0,000
	1000	0,890	0,016	56,971	0,000	0,890	0,000	56,954	0,000
	1500	2,047	0,000	283,951	0,031	2,046	0,016	284,795	0,017
	2000	3,655	0,016	891,037	0,043	3,718	0,016	894,832	0,047
	3500	11,279	0,047	8261,620	0,140	11,279	0,047	8261,079	0,125
	5000	23,104	0,078	34396,279	0,250	23,120	0,094	34265,447	0,266
PROBLEMAS		$m = 10\% \text{ de } n$				$m = 15\% \text{ de } n$			
TIPO	DIMENSÃO	TLA	TAPD	TF	TRS	TLA	TAPD	TF	TRS
TRIANGULAR	100	0,016	0,000	0,000	0,000	0,015	0,000	0,000	0,000
	500	0,234	0,000	3,716	0,000	0,219	0,000	3,733	0,000
	1000	0,891	0,000	56,971	0,000	0,891	0,000	57,000	0,015
	1500	2,047	0,015	284,283	0,032	2,047	0,000	284,268	0,031
	2000	3,656	0,015	891,907	0,052	3,655	0,016	893,760	0,031
	3500	11,310	0,047	8413,691	0,141	11,403	0,031	8264,024	0,140

Tabela 85 – GMRES(m) com estrutura CSR e pré-condicionador Diagonal/ILU(0) – Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		$m = 1\% \text{ de } n$				$m = 5\% \text{ de } n$			
TIPO	DIMENSÃO	TLA	TAPD	TF	TRS	TLA	TAPD	TF	TRS
TRIDIAGONAL	100	0,000	0,000	0,000	0,015	0,000	0,000	0,000	0,000
	500	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
	1000	0,000	0,000	0,000	0,015	0,000	0,000	0,000	0,000
	1500	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,016
	2000	0,000	0,000	0,000	0,016	0,016	0,000	0,000	0,000
	3500	0,015	0,000	0,000	0,000	0,031	0,000	0,000	0,000
	5000	0,031	0,000	0,000	0,016	0,031	0,000	0,000	0,000
	6500	0,031	0,000	0,000	0,000	0,047	0,000	0,000	0,010
	8000	0,047	0,000	0,015	0,000	0,047	0,000	0,016	0,002
PROBLEMAS		$m = 10\% \text{ de } n$				$m = 15\% \text{ de } n$			
TIPO	DIMENSÃO	TLA	TAPD	TF	TRS	TLA	TAPD	TF	TRS
TRIDIAGONAL	100	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
	500	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
	1000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
	1500	0,015	0,000	0,000	0,000	0,000	0,000	0,000	0,015
	2000	0,016	0,000	0,000	0,000	0,015	0,000	0,000	0,000
	3500	0,016	0,000	0,000	0,011	0,015	0,000	0,000	0,016
	5000	0,031	0,000	0,000	0,000	0,032	0,000	0,000	0,000
	6500	0,046	0,000	0,000	0,000	0,032	0,000	0,000	0,015
	8000	0,046	0,000	0,000	0,016	0,046	0,000	0,000	0,000

De acordo com os dados das tabelas acima, a influência do pré-condicionador Diagonal/ILU(0) se mostrou no processo de fatoração, onde o tempo foi menor que o obtido ao utilizar apenas o ILU(0). Observe, por exemplo, o problema triangular com dimensão 3500. No Diagonal/ILU(0), foi preciso 9489,979 segundos, enquanto no Diagonal/ILU(0) foi preciso 8413,691 segundos, uma considerável diferença de um pouco mais de mil segundos.

6.5.13 GMRES(m) com Estrutura de Dados CSR e Pré-condicionador P

6.5.13.1 Problemas dos Repositórios

Na Tabela 86 são apresentadas as iterações e a qualidade do resíduo obtidas após a utilização do Pré-condicionador P.

Tabela 86 – GMRES(m) com estrutura CSR e pré-condicionador P - Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
	I	R	I	R	I	R	I	R
<i>sherman1</i>	457	C2	24	C2	12	C2	7	C2
<i>sherman4</i>	88	C3	9	C3	3	C3	2	C3
<i>add20</i>	1	C3	1	C3	1	C3	1	C2
<i>sherman5</i>	3313	C2	EM		EM		EM	
<i>c-26</i>	2966	C1	EV		EV		EV	
<i>add32</i>	1	C5	1	C4	1	C3	EV	
<i>sherman3</i>	5006	C3	EM		EM		EM	
<i>pres_poisson</i>	5	C3	2	C2	TL		TL	
<i>memplus</i>	1	C4	EV		EV		EV	

É possível perceber, nesta tabela, que para alguns problemas como o *sherman1*, *sherman4*, *sherman5*, *c-26* e *sherman3* foi preciso muitas iterações ou até mesmo o máximo de iterações permitida pelo GMRES(m) para convergir.

Porém, destaca-se ainda que os problemas *add32* com m valendo 1% e 5% e *pres_poisson* com $m = 1\%$ obtiveram resíduos de qualidades excelentes e ótimas.

Na Tabela 87 são apresentados os tempos de leitura e de resolução do novo sistema obtidos para esses testes.

Tabela 87 – GMRES(m) com estrutura CSR e pré-condicionador P - Repositórios.

PROBLEMAS	$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
	TLA	TRNS	TLA	TRNS	TLA	TRNS	TLA	TRNS
<i>sherman1</i>	0,015	0,515	0,000	0,469	0,000	2,270	0,000	5,829
<i>sherman4</i>	0,015	0,125	0,015	0,249	0,000	0,832	0,000	2,473
<i>add20</i>	0,032	0,021	0,031	0,422	0,031	5,452	0,031	26,697
<i>sherman5</i>	0,031	66,479	X		X		X	
<i>c-26</i>	0,031	121,068	X		X		X	
<i>add32</i>	0,047	0,063	0,047	7,217	0,047	126,500	X	
<i>sherman3</i>	0,047	319,699	X		X		X	
<i>pres_poisson</i>	0,691	13,539	0,687	1370,092	X		X	
<i>memplus</i>	0,281	4,764	X		X		X	

Nesta tabela nota-se que esse pré-condicionador alcançou resultados parecidos com o Escala Simétrica. Uma exceção é o problema *sherman3* com m de 1%, onde foi preciso 319,699 segundos, enquanto no outro, levou apenas 3,370 segundos.

6.5.13.2 Problemas do Gerador Aleatório

Nas Tabelas 88, 89 e 90 são apresentadas as quantidades de iterações e qualidades dos resíduos obtidos pelo método.

Tabela 88 – GMRES(m) com estrutura CSR e pré-condicionador P – Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
CHEIA	100	12	C2	2	C3	5	C4	15	C4
	500	1	C3	1	C3	1	C3	39	C3
	1000	1	C3	1	C3	1	C3	1	C3
	1500	1	C2	120	C3	1	C2	2	C3
	2000	1	C3	1	C2	1	C3	EV	
	3500	3	C3	EV		EV		EV	
	5000	12	C3	EV		EV		EV	

Tabela 89 – GMRES(m) com estrutura CSR e pré-condicionador P – Gerador aleatório – Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIANGULAR	100	13	C2	1	C3	10	C2	17	C3
	500	1	C3	10	C4	4	C3	5	C2
	1000	5	C3	15	C2	42	C2	EV	
	1500	113	C3	EV		EV		EV	
	2000	15	C2	EV		EV		EV	
	3500	94	C2	19	C3	EV		EV	
	5000	44	C3	EV		EV		EV	
	6500	20	C3	EV		EV		EV	
	8000	53	C3	EV		EV		EV	

Tabela 90 – GMRES(m) com estrutura CSR e pré-condicionador P – Gerador aleatório – Matriz tridiagonal.

PROBLEMAS		$m = 1\% \text{ de } n$		$m = 5\% \text{ de } n$		$m = 10\% \text{ de } n$		$m = 15\% \text{ de } n$	
TIPO	DIMENSÃO	I	R	I	R	I	R	I	R
TRIDIAGONAL	100	12	C2	1	C4	3	C3	5	C2
	500	1	C3	4	C3	3	C3	2	C3
	1000	3	C3	4	C3	10	C3	6	C2
	1500	2	C3	2	C3	134	C3	EV	
	2000	8	C2	EV		EV		EV	
	3500	53	C2	EV		EV		EV	
	5000	17	C3	EV		EV		EV	
	6500	25	C3	EV		EV		EV	
	8000	9	C3	EV		EV		EV	

A partir destas tabelas, nota-se que este pré-condicionador teve um bom desempenho para os problemas deste gerador. Para os problemas cheios, o método convergiu para boa parte deles. Com dimensão 100, conseguiu até resultados de ótima qualidade com m de 10% e 15%, porém, a partir de problemas de dimensão 2000, começaram a ocorrer estouros de memória.

Ressalta-se que o GMRES(m) com este pré-condicionador, resolveu todos os problemas do gerador aleatório, algo que não ocorreu para os demais. Dentre os problemas triangulares e tridiagonais, também houveram casos que, no Escala Simétrica, ocorria estouro de variável, e agora, foram resolvidos.

Nas Tabelas 91, 92 e 93 são apresentados os tempos de leitura, de construção do novo sistema e os tempos de resolução desse novo sistema gerado pelo método para os problemas de matrizes cheias, triangulares e tridiagonais, respectivamente.

Tabela 91 – GMRES(m) com estrutura CSR e pré-condicionador P – Gerador aleatório – Matriz cheia.

PROBLEMAS		$m = 1\% \text{ de } n$			$m = 5\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
CHEIA	100	0,016	0,000	0,015	0,015	0,000	0,000
	500	0,437	0,016	0,000	0,437	0,000	0,032
	1000	1,796	0,000	0,047	1,796	0,000	0,188
	1500	4,092	0,016	0,109	4,140	0,015	90,189
	2000	7,311	0,015	0,328	7,342	0,031	1,843
	3500	22,635	0,078	5,458	X		
	5000	46,349	0,172	63,592	X		
PROBLEMAS		$m = 10\% \text{ de } n$			$m = 15\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
CHEIA	100	0,016	0,000	0,000	0,016	0,000	0,016
	500	0,453	0,000	0,058	0,437	0,000	4,601
	1000	1,796	0,000	0,605	1,781	0,016	1,328
	1500	4,077	0,016	2,062	4,093	0,016	11,606
	2000	7,310	0,016	5,608	X		

Tabela 92 – GMRES(m) com estrutura CSR e pré-condicionador P – Gerador aleatório –
Matriz triangular.

PROBLEMAS		$m = 1\% \text{ de } n$			$m = 5\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
TRIANGULAR	100	0,015	0,000	0,000	0,000	0,000	0,000
	500	0,219	0,000	0,000	0,218	0,000	0,129
	1000	0,906	0,000	0,094	0,906	0,000	1,499
	1500	2,046	0,016	6,622	X		
	2000	3,655	0,016	2,153	X		
	3500	11,654	0,047	81,767	11,263	0,046	121,247
	5000	23,151	0,078	114,210	X		
	6500	38,897	0,141	116,626	X		
	8000	60,110	0,219	584,276	X		
PROBLEMAS		$m = 10\% \text{ de } n$			$m = 15\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
TRIANGULAR	100	0,016	0,000	0,000	0,000	0,000	0,016
	500	0,219	0,000	0,140	0,218	0,000	0,437
	1000	0,890	0,000	15,588	X		

Tabela 93 – GMRES(m) com estrutura CSR e pré-condicionador P – Gerador aleatório –
Matriz tridiagonal.

PROBLEMAS		$m = 1\% \text{ de } n$			$m = 5\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
TRIDIAGONAL	100	0,000	0,000	0,015	0,000	0,000	0,000
	500	0,000	0,000	0,000	0,000	0,000	0,008
	1000	0,010	0,000	0,003	0,015	0,000	0,063
	1500	0,015	0,000	0,000	0,015	0,000	0,156
	2000	0,016	0,000	0,031	X		
	3500	0,031	0,000	1,093	X		
	5000	0,031	0,000	1,019	X		
	6500	0,046	0,000	3,549	X		
	8000	0,047	0,000	2,512	X		
PROBLEMAS		$m = 10\% \text{ de } n$			$m = 15\% \text{ de } n$		
TIPO	DIMENSÃO	TLA	TCNS	TRNS	TLA	TCNS	TRNS
TRIDIAGONAL	100	0,000	0,000	0,000	0,000	0,000	0,000
	500	0,000	0,000	0,047	0,000	0,000	0,125
	1000	0,000	0,000	1,881	0,000	0,000	4,983
	1500	0,015	0,000	116,248	X		

Na maioria dos casos reportados, os tempos foram semelhantes aos observados no Escala Simétrica (veja as Tabelas 43, 44 e 45), com algumas exceções, onde, conforme o parâmetro m aumenta, a diferença entre eles também aumenta.

Note ainda que este é mais um pré-condicionador que tornou tempo do processo de construção do novo sistema relevante, principalmente nos problemas triangulares.

Na Tabela 92, é possível perceber, que conforme o problema fica maior, esse tempo de construção também cresce, porém para os tempos de resolução ele não foi significativo.

6.5.14 Refinamento de Solução Utilizando GMRES(m)

6.5.14.1 Problemas dos Repositórios

Na Tabela 94 são apresentadas as quantidades de refinamentos realizados, o resíduo final obtido e o tempo total de refinamento para os problemas do repositório.

Tabela 94 – Refinamento de solução utilizando o método GMRES(m) - Repositórios.

MÁXIMO REFINAMENTOS	PROBLEMA	MÉTODO / PRÉ-CONDICIONADOR	I	R	TR
0,5% de n	<i>Sherman4</i>	GMRES($m=1\%$) / Diagonal-ILU(0)	7	C2	0,202
	<i>Sherman3</i>	GMRES($m=1\%$) / Escala Simétrica V. 01	27	C2	166,514
$n/2$	<i>Sherman4</i>	GMRES($m=1\%$) / Diagonal-ILU(0)	553	C3	1,143
	<i>Sherman3</i>	GMRES($m=1\%$) / Escala Simétrica V. 01	2503	C3	325,813
n	<i>Sherman4</i>	GMRES($m=1\%$) / Diagonal-ILU(0)	1105	C3	2,078
	<i>Sherman3</i>	GMRES($m=1\%$) / Escala Simétrica V. 01	5006	C3	489,759
$2n$	<i>Sherman4</i>	GMRES($m=1\%$) / Diagonal-ILU(0)	2209	C3	3,936

Após realizar todos os experimentos para esta classe de problemas, os resultados demonstram que o refinamento conseguiu obter soluções somente para dois problemas, no caso *sherman3* e *sherman4*. Note como o tamanho do problema pode influenciar no processo de refinamento. Tome, como exemplo, os tempos obtidos quando o máximo de iterações para o processo de refinamento eram n . Veja que, para refinar a solução do problema *Sherman3*, levou-se muito mais tempo que o *Sherman4*, sendo que o primeiro é quase 5 vezes maior que o segundo. Para os demais casos, o refinamento resultou em estouros de memória ou de variável, com exceção do *sherman2* com máximo de refinamento de 0,5% de n , onde a quantidade de refinamentos realizada não foi suficiente para melhorar a solução.

6.5.14.2 Problemas do Gerador Aleatório

Na Tabela 95 são apresentadas as quantidades de refinamentos realizados, o resíduo final obtido e o tempo total de refinamento para os problemas do gerador aleatório.

Tabela 95 – Refinamento de solução utilizando o método GMRES(m) – Gerador aleatório.

MÁXIMO REFINAMENTOS	TIPO / DIMENSÃO	MÉTODO / PRÉ-CONDICIONADOR	I	R	TR
1% de n	Cheio / 100	GMRES($m=10\%$) / Escala Simétrica	NC		
		GMRES($m=1\%$) / Escala Simétrica V. 02	NC		
	Triangular / 100	GMRES($m=1\%$) / Lumped	2	C1	0,109
		GMRES($m=15\%$) / Escala Simétrica V.02	2	C3	0,141
	Tridiagonal / 100	GMRES($m=10\%$) / Diagonal-Escala Simétrica	NC		
$n/2$	Cheio / 100	GMRES($m=10\%$) / Escala Simétrica	4	C3	0,252
		GMRES($m=1\%$) / Escala Simétrica V. 02	46	C3	0,085
	Triangular / 100	GMRES($m=1\%$) / Lumped	6	C3	0,135
		GMRES($m=15\%$) / Escala Simétrica V.02	5	C3	0,137
	Tridiagonal / 100	GMRES($m=10\%$) / Diagonal-Escala Simétrica	8	C4	0,110

Destaca-se aqui, que o refinamento para os problemas gerados aleatoriamente teve um desempenho melhor do que para os problemas anteriores, mas apenas para os problemas com dimensão 100. Pois, foram realizados poucos refinamentos e o tempo total também foi baixo (o que já era esperado para problemas pequenos). Outro comportamento interessante a se observar ocorre com o problema tridiagonal com n igual a 100 e máximo de refinamentos em metade da dimensão. Para este, o refinamento conseguiu obter uma solução com um resíduo de ótima qualidade em apenas 8 iterações.

6.5.15 Refinamento de Solução Utilizando ILU(0)

6.5.15.1 Problemas dos Repositórios

Na Tabela 96 são apresentadas as quantidades de refinamentos realizados, o resíduo final obtido e o tempo total de refinamento para os problemas do repositório.

Tabela 96 – Refinamento de solução utilizando a fatoração ILU(0) - Repositórios.

MÁXIMO REFINAMENTOS	PROBLEMA	MÉTODO / PRÉ-CONDICIONADOR	I	R	TR
0,5% de n	$c-26$	GMRES($m=1\%$) / Escala Simétrica V. 02	4	C3	0,056
		Gradiente Conjugado	6	C4	0,078
$n/2$	<i>Sherman4</i>	GMRES($m=1\%$) / Diagonal – ILU(0)	553	C1	0,110
	<i>Sherman3</i>	GMRES($m=1\%$) / Escala Simétrica V. 01	2503	C1	2,269
n	<i>Sherman4</i>	GMRES($m=1\%$) / Diagonal – ILU(0)	1105	C2	0,203
	<i>Sherman3</i>	GMRES($m=1\%$) / Escala Simétrica V. 01	5006	C2	4,401
$2n$	<i>Sherman4</i>	GMRES($m=1\%$) / Diagonal – ILU(0)	2209	C3	0,390
	<i>Sherman3</i>	GMRES($m=1\%$) / Escala Simétrica V. 01	10011	C3	7,411

A partir da Tabela 96, é possível ressaltar alguns resultados interessantes. Primeiro, quando foi usado 0,5% de n para a quantidade máxima de refinamentos, esta técnica conseguiu melhorar as soluções obtidas para o problema $c-26$ pelos métodos GMRES(m) pré-condicionado e Gradiente Conjugado. Para os demais problemas (*sherman2*, *sherman3*, *sherman4* e *sherman5*), mesmo após realizar o máximo de refinamentos, não houve melhora na solução.

Antes de prosseguir, lembre-se que o problema *sherman2* foi o problema com maior quantidade de ocorrências em que a maioria dos métodos não convergiram e que até esta seção somente o GMRES(m) com pré-condicionador ILU(0) foi capaz de encontrar a solução.

Quando se utilizou o máximo de iterações igual a $n/2$ o refinamento começou a melhorar a solução deste problema. Foram necessários 541 refinamentos para obter um resíduo de qualidade boa (C2) em tempo médio de 0,378 segundos. Como esses valores se repetiram para essas ocorrências, com uma pequena variação no tempo, o mesmo não foi apresentado na tabela.

Para os problemas *sherman3* e *sherman4* o refinamento melhorou a solução obtendo um resíduo C1, porém, com muito pouco esforço, tendo um tempo por refinamento de 0,0009 segundos e 0,0001 segundos, respectivamente.

Aumentando o máximo de refinamentos para n , os resultados acerca do problema *sherman2*, mais uma vez, se apresentaram de maneira semelhante em todas as ocorrências. O refinamento não conseguiu melhorar sua solução, que permaneceu com um resíduo de categoria C2. Porém, aumentou a quantidade de refinamentos para 1081, e o tempo médio também aumentou para 0,650 segundos.

Para os problemas *sherman3* e *sherman4* o refinamento conseguiu melhorar um pouco mais a solução (agora com um resíduo C2) ao custo de aumentar as iterações e um pouco do tempo final também.

Por último, tem-se a quantidade máxima de refinamentos igual a $2n$. Neste ponto, os refinamentos em relação ao problema *sherman2* seguiram da mesma forma: para todos os casos ele obteve um resíduo igual aos anteriores (C2), porém, aumentando a quantidade de refinamentos realizados (2161) e também seu tempo médio (1,222 segundos). Mais uma vez, para os problemas *sherman3* e *sherman4*, o refinamento conseguiu melhorar a solução para mais um nível (C3). Vale ressaltar que para nenhum dos máximos de refinamentos foi possível melhorar a solução obtida do problema *sherman5*, logo este não foi apresentado na tabela.

6.5.15.2 Problemas do Gerador Aleatório

Na Tabela 97 são apresentadas as quantidades de refinamentos realizados, o resíduo final obtido e o tempo total de refinamento para os problemas gerados aleatoriamente.

Tabela 97 – Refinamento de solução utilizando a fatoração ILU(0) – Gerador aleatório.

MÁXIMO REFINAMENTOS	TIPO / DIMENSÃO	MÉTODO / PRÉ-CONDICIONADOR	I	R	TR
1% de n	Cheio / 100	GMRES($m=1\%$) / Lumped	2	C4	0,046
		GMRES($m=1\%$) / Escala Simétrica V. 02	2	C2	0,053
	Cheio / 500	GMRES($m=10\%$) / Lumped	6	C3	18,121
	Triangular / 100	GMRES($m=15\%$) / Lumped	2	C1	0,016
		GMRES($m=15\%$) / Escala Simétrica V. 02	2	C3	0,018
$n/2$	Cheio / 100	GMRES($m=1\%$) / Lumped	2	C4	0,047
		GMRES($m=10\%$) / Escala Simétrica	4	C4	0,047
		GMRES($m=1\%$) / Escala Simétrica V. 02	3	C4	0,052
	Cheio / 500	GMRES($m=10\%$) / Lumped	6	C3	18,121
	Triangular / 100	GMRES($m=15\%$) / Lumped	3	C4	0,015
		GMRES($m=15\%$) / Escala Simétrica V. 02	3	C4	0,020
	Tridiagonal / 100	GMRES($m=10\%$) / Diagonal – Escala Simétrica	8	C5	0,015

Ao analisar os resultados apresentados na Tabela 97, percebe-se que o refinamento de solução utilizando a fatoração ILU(0) teve um melhor desempenho, principalmente quando comparado com os resultados do refinamento utilizando GMRES(m) para os mesmos problemas. Desde já, vale citar aqui que os resultados para o máximo de iterações igual a n e a $2n$ foram exatamente os mesmos que os obtidos para $n/2$, ou seja, neste último o refinamento

encontrou as soluções que atendem a tolerância estabelecida, logo, aumentar a quantidade de refinamentos não mudaria os resultados. Por isso os dados dos dois primeiros não foram apresentados na tabela.

Após os refinamentos, obteve-se muitos resultados em que a solução tinha um resíduo de qualidade ótima, destacando ainda o problema tridiagonal com 100 de dimensão que obteve um resíduo excelente (C5).

Verificando estes resultados, percebe-se que para o máximo de refinamentos em 1% de n , esta técnica melhorou a solução (com exceção do problema cheio de dimensão 100 aplicado ao pré-condicionador Escala Simétrica e do problema tridiagonal com tamanho 100 aplicado ao pré-condicionador Diagonal/Escala Simétrica). Porém, ao aumentar esse máximo de refinamentos, foi possível melhorar ainda mais a solução obtida.

6.5.16 GMRES do Octave

Afim de visualizar o desempenho do GMRES(m) que foi implementado ~~aqui~~, os mesmos problemas testes foram executados para o método GMRES disponibilizado pela ferramenta Octave.

Para o método GMRES no Octave foram utilizados dois parâmetros, no caso tolerância para o critério de parada (tol) e o máximo de iterações ($maxite$). Por último, o método pede a entrada de um pré-condicionador (M), o qual, para estes testes, foi estabelecido como sendo a matriz identidade, funcionando como um elemento neutro, ou seja, sem pré-condicionamento.

6.5.16.1 Problemas dos Repositórios

Na Tabela 98 são apresentadas o número de iterações, a qualidade do resíduo e o tempo de resolução do sistema obtidos após a utilização GMRES do software Octave para os problemas dos repositórios.

Tabela 98 – GMRES do Octave - Repositórios.

PROBLEMAS	I	R	TRS
<i>sherman1</i>	357	C2	8,829
<i>sherman2</i>	732	C2	75,776
<i>b_dyn</i>	1089	C1	277,53
<i>b2_ss</i>	1089	C3	279,790
<i>sherman4</i>	127	C2	0,429
<i>fpga_dcop_42</i>	36	C2	0,043
<i>add20</i>	284	C2	5,120
<i>cavity10</i>	629	C2	51,438
<i>sherman5</i>	986	C2	192,230
<i>c-26</i>	2750	C2	6303,700
<i>add32</i>	63	C2	0,362

Destaca-se que, com o método GMRES do Octave foi possível resolver a maioria dos problemas dos repositórios (não houve problemas com estouros de memória ou de variáveis), com exceção dos três maiores, *sherman3*, *pres_poisson* e *memplus*, que, depois 10 horas rodando, ainda tentava resolvê-los.

Outro ponto importante é que, apesar de conseguir resolver quase todos os problemas, a qualidade do resíduo, na maioria deles, era apenas razoável, sem contar que os tempos de resolução foram altos em alguns casos.

Levando em conta os outros métodos vistos no presente trabalho, juntamente com os pré-condicionadores e a estrutura de dados, nota-se claramente o melhor desempenho das técnicas propostas e implementadas frente ao software Octave. Fato este que fica bem claro ao observar alguns problemas. Para o *c-26* considerando os métodos de Jacobi e Gauss-Seidel, foi necessário, apenas, 3 e 2 iterações, respectivamente, e menos de um segundo para resolvê-lo. Outro exemplo foi obtido do GMRES(*m*) com o pré-condicionador ILU(0) para este problema. Nele, obteve-se uma solução melhor (qualidade C3) que a do Octave e isso com apenas 1 iteração, levando menos de 1 segundo, ao passo que o software precisou 2750 iterações com tempo de 6303,700 segundos.

No problema *sherman5*, mais uma vez, é possível notar uma diferença entre os métodos implementados e o do Octave. Para este software, foram necessários 986 iterações e 192,230 segundos para obter uma solução de qualidade C2. Enquanto isso, no GMRES(*m*), utilizando o pré-condicionador Escala Simétrica, foi possível obter uma solução de melhor qualidade, porém, em apenas 3 iterações, levando 4 segundos e meio, aproximadamente.

Por último, observe o problema *add32*, o maior resolvido por ele. Enquanto o Octave precisou de 63 iterações para resolvê-lo em 0,362 segundos, obtendo uma solução de qualidade

razoável (C2), o GMRES(m) com o pré-condicionador Lumped, precisou de apenas 1 iteração para obter uma solução de alta qualidade (C5) e em apenas 0,062 segundos.

6.5.16.2 Problemas do Gerador Aleatório

Nas Tabelas 99, 100 e 101 são apresentadas as iterações, a qualidade do resíduo e o tempo de resolução do sistema obtidos após a utilização GMRES do software Octave para os problemas do gerador aleatório.

Tabela 99 – GMRES do Octave – Gerador aleatório.

PROBLEMAS												
TIPO	DIMENSÃO	I	R	TRS	TIPO	I	R	TRS	TIPO	I	R	TRS
CHEIA	100	5	C3	0,003	TRIANGULAR	3	C4	0,002	TRIDIAGONAL	3	C3	0,002
	500	4	C3	0,010		3	C4	0,007		4	C3	0,005
	1000	4	C3	0,025		3	C4	0,022		3	C2	0,012
	1500	3	C2	0,051		3	C4	0,048		3	C2	0,025
	2000	4	C3	0,103		3	C4	0,086		3	C3	0,043
	3500	3	C2	0,263		3	C4	0,257		3	C3	0,135
	5000	3	C3	0,516		3	C4	0,516		3	C3	0,270
	6500	3	C3	0,836		3	C4	0,916		3	C3	0,438
	8000	33	C3	1,442		3	C4	1,326		3	C3	0,634

Para os problemas do gerador aleatório, pode-se dizer que o método do Octave teve um desempenho melhor em comparação a quase todos os outros visto anteriormente neste trabalho. Neste caso, o software conseguiu, em quase todos os casos, realizar menos de 10 iterações e o tempo de resolução, em exceção dos problemas do tipo cheio e triangular com dimensão de 8000, menor que 1 segundo.

Na qualidade de solução ele também obteve vários casos com resíduos de ótimas qualidades para os problemas triangulares. O GMRES(m) com o pré-condicionador ILU(0) também obteve alguns resíduos dessa qualidade.

6.6 Análise de Resultados

Para uma melhor compreensão de alguns resultados obtidos e apresentados anteriormente, será feita uma breve análise dos dados, buscando observar e discutir acerca dos principais pontos em destaque.

Assim, selecionou-se os problemas dos repositórios em comum que foram resolvidos pelos métodos utilizados, para os quais foi calculado a média aritmética tanto para iterações e quanto para tempo. O objetivo é analisar três aspectos principais: o desempenho da estrutura de dados CSR em comparação da estrutura tradicional, a eficiência dos métodos estudados e, por fim, o comportamento do método $\text{GMRES}(m)$ com o uso dos pré-condicionadores.

Primeiro são apresentados os resultados considerando o impacto do uso da estrutura de dados CSR. Para a análise, considerou-se os seguintes problemas: *sherman1*, *sherman4*, *add20*, *add32* e *sherman3*. Na Figura 6.5 são apresentadas as quantidades médias de iteração para os métodos Jacobi, Gauss-Seidel e Gradiente Conjugado, utilizando as estruturas de dados CSR e Indexada.

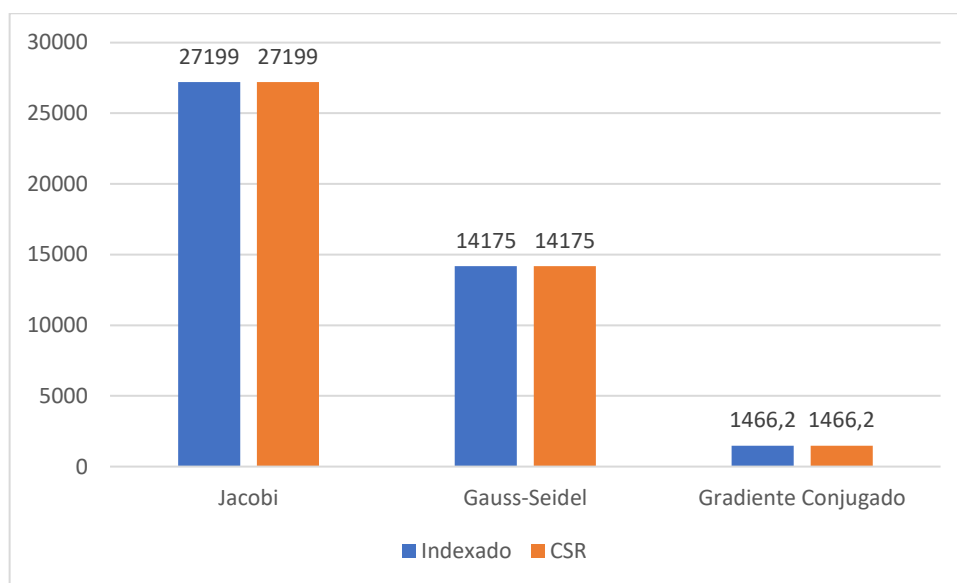


Figura 6.5: Média de iterações entre as estruturas indexada e CSR.

Note que, como já era esperado, a estrutura de dados CSR não afeta o quantitativo de iterações. Pois, a estrutura não muda o processo de resolução do método, mas torna o método muito mais eficiente com relação ao tempo computacional ao evitar cálculos desnecessários. Assim, na Figura 6.6 são apresentados os tempos médios para os métodos Jacobi, Gauss-Seidel e Gradiente Conjugado, utilizando as estruturas de dados CSR e indexada.

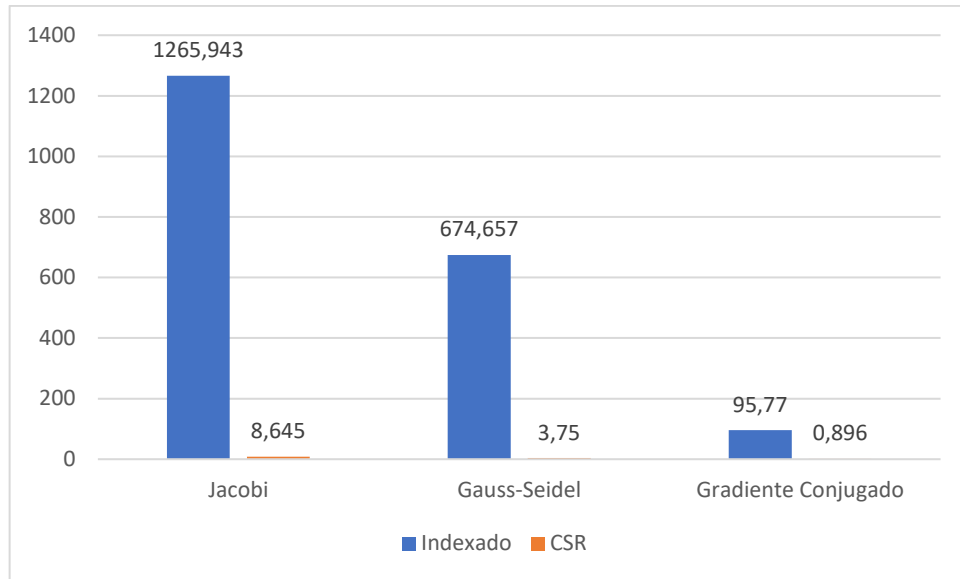


Figura 6.6: Tempo médio entre as estruturas indexada e CSR.

Observa-se que o método de Gauss-Seidel, tem um desempenho muito melhor que o Jacobi, realizando praticamente metade das iterações. O Gradiente Conjugado, sendo um método que envolve mais procedimentos que os outros dois, precisou de bem menos iterações para resolver os mesmos problemas. Note que, mais uma vez, pela representação visual gráfica, a estrutura CSR consegue reduzir drasticamente o tempo de resolução. Logo, fica evidente a importância do uso de uma estrutura de dados adequada para as principais operações dos métodos considerados e para o armazenamento dos elementos em problemas esparsos de grande porte.

Agora serão comparados os resultados obtidos para todos os métodos sem o uso pré-condicionadores no GMRES(m). Para essa comparação, foram usados os seguintes problemas: *sherman1*, *sherman4*, *add20* e *add32*. Na Figura 6.7 são apresentadas as quantidades médias de iteração para os métodos Jacobi, Gauss-Seidel, Gradiente Conjugado, GMRES(m) e GMRES-Octave.

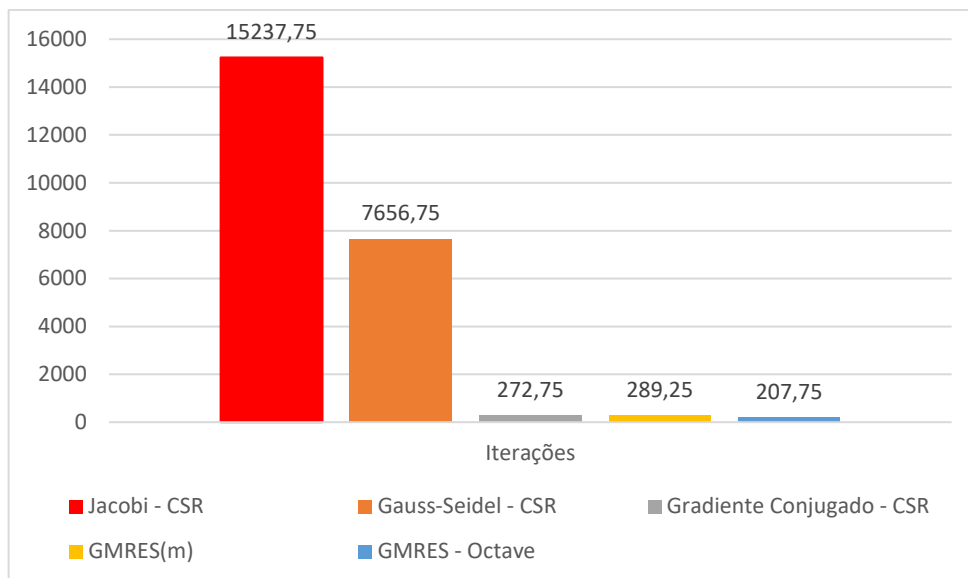


Figura 6.7: Média de iterações entre os métodos sem pré-condicionamento.

Perceba, para os métodos mais simples (Jacobi e Gauss-Seidel), que a quantidade média de iterações foi alta em comparação com os outros três, que são mais robustos, ou seja, que usam de estratégias melhores para resolver os sistemas. Note que o Gradiente Conjugado, GMRES(m) e o GMRES-Octave, apesar de apresentarem iterações médias bem próximas e muito melhores que os outros dois métodos, tendo o Octave uma ligeira vantagem, isso não representa o quadro de desempenho geral, pois, dependendo de característica do problema, um método pode apresentar melhor desempenho relação a outro.

Na Figura 6.8 são apresentados os tempos médios para os métodos Jacobi, Gauss-seidel, Gradiente Conjugado, GMRES(m) e GMRES-Octave.

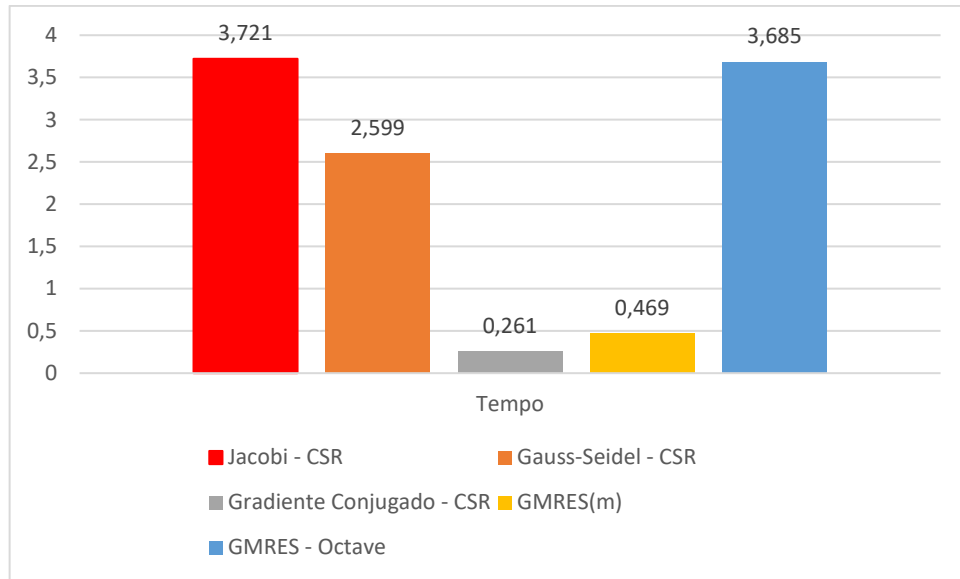


Figura 6.8: Tempo médio entre os métodos sem pré-condicionamento.

Note, na Figura 6.8, que os métodos Jacobi e GMRES-Octave levaram um tempo maior em relação aos demais métodos. No geral, os métodos Gradiente Conjugado e GMRES(m) obtiveram os menores tempos, com vantagem para o Gradiente, sendo bem inferiores aos demais, principalmente em relação a implementação do GMRES-Octave.

Por último, tem-se os resultados para o GMRES(m) sem e com os pré-condicionadores. Os problemas usados nessa comparação foram: *add20*, *add32*, *pres_poisson* e *memplus*. Na Figura 6.9 são apresentadas as médias de iterações para estes métodos.

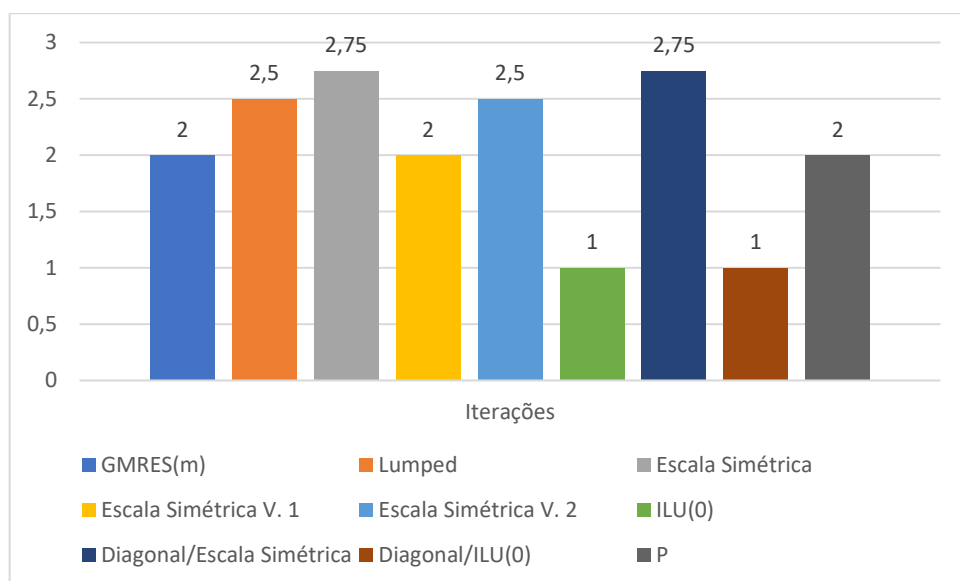


Figura 6.9: Média de iterações com e sem pré-condicionamento para o método GMRES(m).

Neste gráfico, apesar da diferença entre cada um não ser tão grande, ainda é possível extrair algumas informações. Os pré-condicionadores baseados na fatoração ILU(0) mostraram o melhor desempenho, precisando de apenas uma única iteração para resolver a maioria dos problemas. Enquanto isso, o Escala Simétrica e sua combinação com o Diagonal, foram os que precisaram de mais iterações.

Por fim, na Figura 6.10 são apresentados os tempos médios para o método GMRES(m) com esses pré-condicionadores.

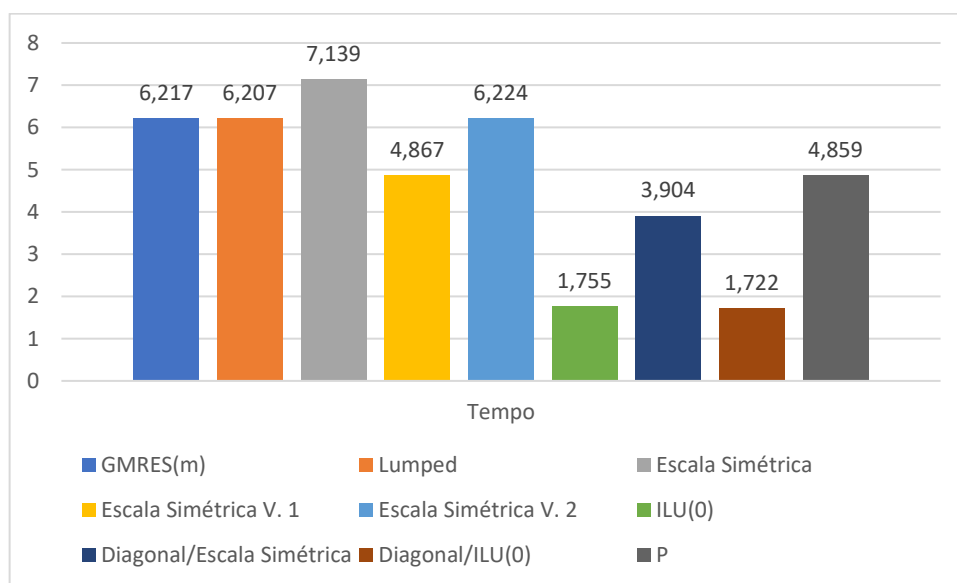


Figura 6.10: Tempo médio com e sem pré-condicionamento para o método GMRES(m).

Note que, os pré-condicionadores baseados na fatoração ILU(0) foram mais eficientes tendo um tempo médio de apenas 1,7 segundos, aproximadamente. O Escala Simétrica não obteve um bom desempenho, pois precisou de mais tempo, em média, para resolver os problemas considerados. Perceba também que neste gráfico a diferença entre os pré-condicionadores não foi tão grande.

Destaca-se que estes resultados não retratam o desempenho geral para todos os problemas que foram tratados neste trabalho sejam do repositório ou do gerador, mas fornece um pequeno panorama da performance dos métodos e das técnicas utilizadas para os problemas que foram considerados somente nesta seção.

Capítulo 7

Conclusões e Trabalhos Futuros

Para resolver sistemas de equações lineares, os métodos diretos realizam operações sobre a matriz do sistema, alterando seus elementos, podendo perder eficiência à medida que cresce o número de equações e o sistema torna-se mais esparso (ordem de milhares). Geralmente, esses métodos produzem muitos preenchimentos (surgimento de um novo elemento não-nulo) o que não ocorre com os métodos iterativos e a complexidade é muito alta para sistemas esparsos de grande porte. Por outro lado, os métodos iterativos dependem de condições de convergência, mas quando aplicáveis tendem a ser mais eficiente para resolver sistemas de grande porte.

Para tratar de forma adequada a questão de convergência dos métodos é desejável o uso de técnicas, como pré-condicionamento, refinamento da solução, estruturas de dados eficientes entre outras, para tentar conseguir a resolução dos problemas, melhorar a qualidade da solução obtida, como também para reduzir o tempo de resolução e/ou aumentar a faixa de problemas que podem ser resolvidos, evitando desperdício de memória.

Assim, neste trabalho foram estudados quatro métodos iterativos: Jacobi, Gauss-Seidel, Gradiente Conjugado, GMRES reinicializado e o GMRES do software Octave. Fez-se também um estudo comparando o comportamento dos métodos ao utilizar a estrutura de dados indexado e a estrutura CSR. Foi aplicado, ainda, especificamente no GMRES(m), a técnica de soma de vetores esparsos, na tentativa de torná-lo mais eficiente.

Outro conjunto de técnicas utilizado neste trabalho foram os pré-condicionadores, onde trabalhou-se com alguns já conhecidos, juntamente com variações e modificações propostas. Ainda acerca dos pré-condicionadores, foi utilizado também uma nova proposta, onde são utilizados dois pré-condicionadores em sequência. Uma última técnica utilizada, o refinamento de solução, teve por foco melhorar a qualidade das soluções obtidas.

Os problemas utilizados foram separados em dois grupos: primeiro, aqueles retirados de repositórios online já apresentados no Capítulo 5. O segundo provém de um gerador aleatório desenvolvido, os quais eram problemas cheios, triangulares e tridiagonais.

Dentre estes métodos, nota-se que cada um teve suas vantagens e desvantagens. Observa-se de um modo geral, como os métodos Jacobi e Gauss-Seidel, que são mais simples, tendem a realizar mais iterações, porém, o esforço é menor a cada procedimento iterativo. Os métodos dos Gradientes Conjugados e GMRES(m) tendem a realizar menos iterações, mas com um custo maior para cada uma. Porém, vale citar que em algumas situações, estes métodos tiveram um melhor desempenho por iteração em termos de tempo computacional quando comparado aos outros dois.

Para resolver sistemas lineares esparsos de forma a explorar a esparsidade da matriz dos coeficientes, foi utilizada a estrutura de dados CSR, com os objetivos de evitar desperdício de memória e a realização de cálculos desnecessários. Logo, os resultados mostraram que a mudança de estrutura de dados da forma indexada para a estrutura CSR permitiu a resolução de alguns problemas (médios e grandes), o que não era possível na estrutura de dados tradicional, devido à falta de memória, além de reduzir drasticamente o tempo computacional.

Em seguida, foi proposta aqui a técnica de soma de vetores esparsos no processo de Arnoldi, rotina interna do método GMRES(m), para tentar melhorar ainda mais o tempo computacional com a estrutura CSR. No entanto, seu desempenho não provocou o efeito desejado. Pelo contrário, além de aumentar os tempos de resolução, também aumentou os casos de estouro de memória, logo, a quantidade de problemas resolvidos diminuiu. Ressalta-se que é necessário investigar melhor o uso desta técnica no GMRES(m), além de modificar o procedimento de alocação dinâmica de memória que foi feito a cada iteração.

Ainda neste trabalho, foram estudados alguns pré-condicionadores conhecidos, juntamente com algumas variantes propostas. A nova estratégia, também abordada nesta pesquisa, foi a proposta de utilização de dois pré-condicionadores em sequência, no caso Diagonal/Escala Simétrica e Diagonal/ILU(0). No geral, os resultados obtidos mostram que um único pré-condicionador não tem o melhor desempenho para todos os tipos problemas. Porém, dentre eles, há alguns se destacam, como foi o caso da segunda variante do Escala Simétrica a qual mostrou melhor desempenho em relação ao original, e também a combinação Diagonal/ILU(0), tendo mostrado ótimos resultados para vários problemas.

Cabe enfatizar que a técnica de refinamento de solução utilizada proporcionou a resolução de alguns problemas, que antes não tinham sido solucionados e melhorou a qualidade da solução em vários outros problemas com destaque para a fatoração LU incompleta sem preenchimento. Sua principal vantagem é que o processo de fatoração é realizado uma única vez antes dos refinamentos, e depois é reaproveitada a cada nova iteração sem alterar a densidade da matriz.

Por fim, foi testado também o método GMRES reinicializado do software Octave. Os resultados revelam que o tempo de resolução no geral foi bem menor do que o GMRES(m) implementado com a estrutura CSR. Este resultado já era esperado dado que o Octave é uma ferramenta bem desenvolvida e difundida na comunidade acadêmica já há algum tempo. Por outro lado, a qualidade dos resíduos das soluções obtidas com este software foi restrita a categoria C3, sendo que os métodos utilizados tiveram, em algumas situações, resultados melhores em termos de qualidade da solução. Ressalta-se que o Octave não conseguiu resolver os problemas maiores (pois seu tempo de execução extrapolou o limite estabelecido), ao passo que os métodos implementados tiveram êxito na resolução destes problemas, principalmente com o uso da estrutura de dados CSR.

Contudo, com base nessas conclusões, é possível elencar, a seguir, algumas atividades que podem ser desenvolvidas como trabalhos futuros:

- Implementar a estrutura de dados CSR para a matriz de Hessemberg gerada pelo processo de Arnoldi no método GMRES(m);
- Com base nos resultados obtidos, identificar os gargalos computacionais para a técnica de soma de vetores esparsos para então melhorar a utilização desta abordagem e tentar propor como parâmetro uma quantidade pré-determinada de memória alocada para tentar agilizar o procedimento de realocação de memória;
- Pesquisar novos pré-condicionadores, além de propor novas variantes e/ou combinações entre eles;
- Aplicar os métodos e técnicas abordadas neste trabalho para sistemas lineares que surgem como um subproblema na resolução de equações diferenciais parciais ou em outras situações em que resolver sistemas lineares seja muito relevante;
- Utilizar de técnicas mais avançadas, como processamento paralelo, para melhorar a eficiência do GMRES(m) e reduzir seu tempo de execução.

Referências

- ARNOLDI, W. **The principle of minimized iteration in the solution of the matrix eigenvalue problem**, Quarterly of Applied Mathematics, v. 9, pp. 17–25, 1951.
- ASANO, C. H., COLLI, E. **Cálculo numérico-fundamentos e aplicações**. Departamento de Matemática Aplicada, IME-USP, 15, 2009.
- BENZI, M. **Técnicas de pré-condicionamento para grandes sistemas lineares: um levantamento**. Journal of computational Physics , 182 (2), 418-477, 2002.
- BOCANEGRA, S. **Algoritmos de Newton-Krylov Pré-condicionados para Métodos de Pontos Interiores**. Tese de Doutorado. Universidade Federal de Minas Gerais-UFMG, 2005.
- BORTOLI, A., CARDOSO, C., FACHIN, M., CUNHA, R. **Introdução ao Cálculo Numérico**. Porto Alegre: Cadernos de Matemática e Estatística B, 59, 2003.
- CAMPOS, J. L. E. **Análise Numérica do Transporte de Contaminantes em Meios Porosos com Reações Químicas**. Departamento de Engenharia Civil. Tese de Doutorado. PUC-Rio, 1999.
- CARVALHO, L. M., FORTES, W. R., MANGIAVACCHI, N. **Pré-condicionadores e solucionadores para problemas de reservatório**. In: CONGRESSO NACIONAL DE MATEMÁTICA APLICADA E COMPUTACIONAL, 31., 2008, Belém. Anais... . Belém: Sbmac, 2008. p. 363 - 369.
- CHUNG, C. L. **Simulação de Reservatórios de Óleo Leve com Poço Horizontal Hidraulicamente Fraturado**. TCC (Graduação) – Engenharia Mecânica, Nova Friburgo, 2015.
- COELHO, M. A. O. **Métodos Iterativos para Resolver Sistemas de Equações Algébricas Lineares em Estruturas Esparsas**. Dissertação de Mestrado. Pós-graduação em Modelagem Computacional em Ciência e Tecnologia. Universidade Federal Fluminense, Volta Redonda, 2014.
- CUNHA, M. C. C. **Métodos Numéricos**, Editora Unicamp. Campinas, SP, 2000.
- GONÇALEZ, T. T. **Algoritmos Adaptativos para o Método GMRES(m)**. Dissertação de Mestrado. Instituto de Matemática. Universidade Federal do Rio Grande do Sul, 2005.
- JENNINGS, A., MALIK, G. M. **The Solution of Sparse Linear Equations by the Conjugate Gradient Method**. International Journal for Numerical Methods in Engineering, vol 12, 141-158, 1978.

- JUNIOR, N. G. B. **Métodos Iterativos Para Sistemas Lineares**, Projeto Supervisionado, Departamento de Matemática Aplicada, IMECC – UNICAMP, 2011.
- LUGON, B. A. **Algoritmos de reordenamento de matrizes esparsas aplicados a Pré-condicionadores ILU(p)**. TCC (Graduação) - Ciência da Computação, Vitória, 2013.
- MEDEIROS, E. N. **NI-GMRES Pré-condicionado**. 2014. 61 f. Dissertação (Mestrado) – Curso de Matemática Aplicada e Estatística, Universidade Federal do Rio Grande do Norte, Natal, 2014.
- MOTTA, V. S. **Solução de Sistemas Lineares de Grande Porte com Múltiplos Lados Direitos**. Tese de Doutorado. COPPE/ Universidade Federal do Rio de Janeiro, 2010.
- OCTAVE. Disponível em: <<https://www.gnu.org/software/octave/index>>. Acesso em: 26 jun. 2021.
- PACHÁS, M. A. C. **Análise Limite com Otimizador de Grande Escala e Análise de Confiabilidade**. 2009. 188 f. Tese (Doutorado) - Curso de Engenharia Civil, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2009.
- PAIGE, C. C., SAUNDERS, M. A. **Solution of sparse indefinite systems of linear equations**. SIAM J. Num. Anal., Vol. 12, No. 4, pp. 617-629, 1975.
- PAZ, A. R. A. **Aplicação do Método GMRES na Solução de Problemas de Estabilidade em Sistemas de Energia Elétrica**. 2012. 192 f. Tese (Doutorado) - Curso de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2012.
- PEDROSA, D. P. F. **Sistemas lineares**. Métodos Computacionais em Engenharia, Notas de Aula, UFRN, 2005.
- PESCADOR, A. POSSAMAI, J. P. POSSAMAI, C. R. **Aplicação De Álgebra Linear Na Engenharia**. In: CONGRESSO BRASILEIRO DE EDUCAÇÃO EM ENGENHARIA, 39., 2011, Blumenau. Anais... . Blumenau: Cobenge, 2011.
- PESSANHA, J. E. O., PAZ, A. A., PRADA, R. **Aplicação do método GMRES em estudos de estabilidade de sistemas de energia elétrica**. *Sba Controle & Automação* [online]. 2012, vol.23, n.3, pp. 321-330. ISSN 0103-1759, 2012.
- PINI, G., GAMBOLATI, G. **Is a simple diagonal scaling the best preconditioner for conjugate gradients on supercomputers?** *Advances in Water Resources*, 13(3):147–153, 1990.
- PIRES, R. A. **Elaboração de um Código Computacional para Resolução de Sistemas Lineares de Grande Porte**. 2014. 19 f. TCC (Graduação) - Curso de Licenciatura em Matemática, Universidade Federal de São João Del-rei, São João Del-rei, 2014

- POMA, C. E. P. **Um Solucionador Iterativo Para Sistemas-Lineares: Aplicação no Problema do Fluxo de Carga**. 2010. 265 f. Tese (Doutorado) - Curso de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2010.
- PURCINA, L. A. **Técnicas de Otimização Evolutiva Aplicadas à Solução de Grandes Sistemas Lineares**. Tese de Doutorado. Faculdade de Engenharia Mecânica. UFU, 2010.
- RODRIGUES, C. F. **Análise Comparativa Entre os Métodos Decomposição em Valores Singulares e Análise de Componentes Principais Envolvendo Matrizes Esparsas de Grande Porte**. 2011. 187 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte, 2011.
- SAAD, Y. **Iterative Methods for Sparse Linear Systems**. Second Edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.
- SAAD, Y., SCHULTZ, M. H. **GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems**. SIAM Journal on Scientific and Statistical Computing, v. 7, n. 3, pp. 856–869, 1986.
- SANTOS, L. F. **Método Dual Simplex para problemas canalizados com estruturas de dados eficientes**. Dissertação de Mestrado, Programa de Pós-graduação em Modelagem Computacional em Ciência e Tecnologia, Universidade Federal Fluminense, Volta Redonda, 2020.
- SILVA, L. H. **Método GMRES Para Sistemas Lineares Esparsos**, TCC (Graduação) - Curso de Licenciatura em Computação, Universidade Federal Fluminense, Santo Antônio de Pádua, 90 f., 2017.
- SILVA, L. H., SOUSA, R. S. **Método GMRES Reinicializado Com Os Pré-Condicionadores Jacobi e Lumped**. In: Encontro Nacional De Modelagem Computacional, 21., Búzios. Anais... . Búzios, Rj, p. 1 – 2, 2018.
- SILVA, L. H., SOUSA, R. S. **Pré-Condicionadores Escala Simétrica E ILU(0) Para O Método GMRES(m)**. In: Encontro Nacional De Modelagem Computacional, 22., Juiz de Fora. Anais... . Juiz de Fora, Rj, 2019.
- SILVA, L. H., SOUSA, R. S., JUNIOR, C. A. C., CORRÊA, R. A. P. **Método GMRES Reinicializado Com A Estrutura De Dados Csr**. In: Encontro Nacional De Modelagem Computacional. 20., Friburgo. Poster. Friburgo: Enmc, 2017.
- SOUSA, R. S., SILVA, L. H. **Método GMRES Para Sistemas Lineares Esparsos**. In: Simpósio De Engenharia De Produção, 23., 2016, Bauru. Anais... . Bauru: Simpep, 2016. p. 49 - 61.

SOUZA, M. J. F., **Sistemas Lineares**, Cálculo Numérico, Notas de aula, Universidade Federal de Ouro Preto, 2018.

STRIKWERDA, J., STODDER, S. **Convergence results for GMRES(m)**. University of Wisconsin-Madison Department of Computer Sciences, 1995.