Universidade Federal da Paraíba
Centro de Informática
Programa de Pós-Graduação em Modelagem Matemática e Computacional

COMPARAÇÃO DE MÉTODOS DE ARMAZENAMENTO DE MATRIZES ESPARSAS PARA CONTRIBUIÇÃO À SIMULAÇÃO EM TEMPO REAL DE REDES ELÉTRICAS

Raphael Dantas Pinho

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Modelagem Matemática e Computacional, PPGMMC/UFPB, da Universidade Federal da Paraíba, como parte dos requisitos necessários à obtenção do título de Mestre em Modelagem Matemática e Computacional.

Orientadores: Luciano Sales Barros
Camila Mara Vital Barros

João Pessoa
Novembro de 2022

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de RAPHAEL DANTAS PINHO, candidato ao título de Mestre em Matemática Computacional, na Área de Modelagem Matemática e Computacional, realizada no dia 24 de novembro de 2022.

Aos vinte e quatro dias do mês de novembro do ano de dois mil e vinte e dois, às 08h30min, via videoconferência, reuniram-se os membros da Banca Examinadora constituída para julgar o Trabalho Final do discente **RAPHAEL DANTAS PINHO,** vinculado a Universidade Federal da Paraíba sob a matrícula nº 20201001592, candidato ao grau de Mestre em "*Modelagem Matemática e Computacional*", na linha de pesquisa *"Modelagem e Simulação de Sistemas"*, do Programa de Pós-Graduação em Modelagem Matemática e Computacional. A comissão examinadora foi composta pelos professores Luciano Sales Barros, Orientador e Presidente da Banca; Camila Mara Vital Barros, Coorientadora; Camila Seibel Gehrke, Examinadora Externa ao Programa; e Daniel Barbosa, Examinador Externo à Instituição. Dando início aos trabalhos, o Presidente da Banca cumprimentou os presentes, comunicou a finalidade da reunião e passou a palavra ao candidato para que fizesse, oralmente, a exposição do trabalho de dissertação intitulado *"Métodos de Armazenamento de Matrizes Esparsas para Simulação em Tempo Real de Redes Elétricas"*. Concluída a exposição, o candidato foi arguido pela Banca Examinadora, que emitiu o seguinte parecer: "***aprovado***". Do ocorrido, eu, Gean Paulo P. M. de Barros, secretário do Programa de Pós-Graduação em Modelagem Matemática e Computacional (PPGMMC), lavrei a presente ata, que vai assinada por mim e pelos membros da Banca Examinadora.

João Pessoa, 24 de novembro de 2022.

Gean Paulo Pereira Maurício de Barros
Secretário do PPGMMC
SIAPE 2326476

Prof. Dr. Luciano Sales Barros
Orientador (PPGMMC)

Profª. Drª. Camila Mara Vital Barros
Coorientadora (PPGMMC)

Documento assinado digitalmente
gov.br CAMILA SEIBEL GEHRKE
Data: 25/11/2022 09:24:32-0300
Verifique em https://verificador.iti.br

Profª. Drª. Camila Seibel Gehrke
Examinadora Externa ao Programa (UFPB)

Documento assinado digitalmente
gov.br DANIEL BARBOSA
Data: 24/11/2022 17:01:02-0300
Verifique em https://verificador.iti.br

Prof. Dr. Daniel Barbosa
Examinador Externo à Instituição (UFBA)

Catalogação na publicação
Seção de Catalogação e Classificação

P654c Pinho, Raphael Dantas.
Comparação de métodos de armazenamento de matrizes esparsas para contribuição à simulação em tempo real de redes elétricas / Raphael Dantas Pinho. - João Pessoa, 2022.
164 f. : il.

Orientação: Luciano Sales Barros.
Coorientação: Camila Mara Vital Barros.
Dissertação (Mestrado) - UFPB/CI.

1. Rede Elétrica - Modelagem computacional. 2. SDTR - Simulação em Tempo Real. 3. Matrizes esparsas. 4. Sistema de equações. 5. Método de armazenamento CSR. 6. Método de armazenamento CSC. I. Barros, Luciano Sales. II. Barros, Camila Mara Vital. III. Título.

UFPB/BC CDU 621.311.1(043)

Elaborado por ANNA REGINA DA SILVA RIBEIRO - CRB-15/24

Às instituições de ensino que fiz parte pela formação cidadã.

Agradecimentos

Agradeço a Deus pelo dom da vida e por sempre me amparar nos momentos de dificuldade.

Aos orientadores, professores Luciano e Camila, primeiramente, pelo acolhimento durante o período pandêmico. Seus ensinamentos, conselhos, suporte, atenção e paciência foram essenciais ao longo deste Mestrado.

A minha família pela estrutura e auxílio que sempre concederam a mim, além da confiança presente em toda caminhada acadêmica.

A Universidade Federal da Paraíba e ao Programa de Pós-Graduação em Modelagem Matemática e Computacional (PPGMMC) pela oportunidade.

Aos amigos, colegas e pessoas que integraram este período de pós-graduação. Distantes ou próximos, fizeram a diferença em diversos momentos.

Resumo da Dissertação apresentada ao PPGMMC/CI/UFPB como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

COMPARAÇÃO DE MÉTODOS DE ARMAZENAMENTO DE MATRIZES ESPARSAS PARA CONTRIBUIÇÃO À SIMULAÇÃO EM TEMPO REAL DE REDES ELÉTRICAS

Raphael Dantas Pinho

Novembro/2022

Orientadores: Luciano Sales Barros
Camila Mara Vital Barros

Programa: Modelagem Matemática e Computacional

A simulação digital em tempo real (SDTR) de redes elétricas consiste numa importante ferramenta de análise, com aplicações na operação, projeto, planejamento e ampliação dos sistemas elétricos. Dentre os simuladores, destacam-se o Real-Time Digital Simulator (RTDS®), desenvolvido pela RTDS Technologies, e o HYPERSIM e eMEGASIM, produzidos pela OPAL-RT Technologies. Para viabilizar a SDTR, faz-se necessária a modelagem computacional dos componentes das redes elétricas e suas características, desde a geração de energia e circuitos de distribuição e transmissão, até as cargas. Essa modelagem representa grande obstáculo para a SDTR, visto que ela consiste na solução de sistemas de equações lineares de grandes dimensões, advindas do alto número de barras que compõem às redes. Invariavelmente, as matrizes que descrevem o comportamento da rede passiva são esparsas e neste trabalho, dá-se ênfase às técnicas de armazenamento destas matrizes, com o intuito de otimizar o processamento, operando somente com os valores diferentes de zero. Através da coleta do tempo de processamento e uso da memória computacional, compara-se o desempenho de cinco destas técnicas: *Compressed Sparse Row* (CSR), *Compressed Sparse Column* (CSC), *Compressed Sparse Vector* (CSV), Skyline e DFA2, que se associaram aos métodos iterativos de Jacobi e Gauss-Seidel para obter as soluções do sistema de equações que descreve o comportamento da rede elétrica. Como referência para essa comparação, o OpenDSS, através da *OpenDSSDirect.py*, também foi submetido as análises propostas. Os resultados obtidos mostram que os métodos de armazenamento CSR e CSC, associados ao método de Gauss-Seidel, demonstram capacidade para colaborar com a simulação em tempo real.

Abstract of Dissertation presented to PPGMMC/CI/UFPB as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

COMPARISON OF SPARSE MATRICES STORAGE METHODS TO CONTRIBUTE TO THE REAL-TIME SIMULATION OF ELECTRIC NETWORKS

Raphael Dantas Pinho

November/2022

Advisors: Luciano Sales Barros
Camila Mara Vital Barros

Program: Computational Mathematical Modelling

Real-time digital simulation (RTDS) of electrical networks is an important analysis tool, with applications in the operation, design, planning, and expansion of electrical systems. Among the simulators, the Real-Time Digital Simulator (RTDS®), developed by RTDS Technologies, and the HYPERSIM and eMEGASIM, produced by OPAL-RT Technologies stand out. To make SDTR viable, it is necessary the computational modeling of the components of the electrical networks and their characteristics, from the generation of energy and distribution and transmission circuits to the loads. This modeling represents a major obstacle for RTDS since it consists of solving systems of large linear equations, arising from the high number of bars that make up the networks. Invariably, the matrices that describe the behavior of the passive network are sparse and, in this work, emphasis is given to the storage techniques of these matrices, to optimize the processing, operating only with values other than zero. Through the collection of processing time and computational memory use, the performance of five of these techniques is compared: Compressed Sparse Row (CSR), Compressed Sparse Column (CSC), Compressed Sparse Vector (CSV), Skyline, and DFA2, which are associated with the iterative methods of Jacobi and Gauss-Seidel to obtain the solutions of the system of equations that describes the behavior of the electrical network. As a reference for this comparison, OpenDSS, through the *OpenDSSDirect.py*, was also submitted to the proposed analyses. The results obtained show that the CSR and CSC storage methods, associated with the Gauss-Seidel method, demonstrate the ability to collaborate with the simulation in real-time.

Sumário

Lista de Figuras **ix**

Lista de Tabelas **x**

1 Introdução **1**
1.1 Contextualização 1
1.2 Objetivos 3
1.3 Metodologia 3
1.4 Estrutura do Dissertação 4

2 Revisão Bibliográfica **5**
2.1 Principais Referências, Comparações e Discussões 5

3 Fundamentação Teórica **8**
3.1 Modelagem dos Componentes da Rede Elétrica 8
3.1.1 Linhas de Transmissão 9
3.1.2 Transformadores 10
3.1.3 Cargas 11
3.1.4 Unidades de Geração Distribuída 11
3.2 Descrição e Construção da Matriz Admitância de Barra de uma Rede Elétrica 12
3.3 Sistema de Equações e Métodos de Armazenamento de Matrizes . . . 13
3.3.1 Método CSR (Compressed Sparse Row) 14
3.3.2 Método CSC (Compressed Sparse Row) 15
3.3.3 Método CSV (Compressed Sparse Vector) 16
3.3.4 Método Skyline 16
3.3.5 Método DFA2 17
3.4 Resumo sobre Métodos de Armazenamento de Matrizes 18
3.5 Métodos Iterativos de Jacobi e Gauss-Seidel 19

4 Método Proposto **20**
4.1 OpenDSSDirect.py 20

4.2 Integração dos Métodos de Armazenamento ao Método de Jacobi . . 21
4.3 Integração dos Métodos de Armazenamento ao Método de Gauss-Seidel 24

5 Resultados e Discussões 26
5.1 Descrição da Rede Elétrica 26
5.2 Matriz Admitância de Barra da Rede e Esparsidade 29
5.3 Parâmetros Utilizados 31
5.4 Resultados para o Sistema Equilibrado 33
5.4.1 Sem a inserção das Unidades de Geração Distribuída 33
5.4.2 Com a inserção das Unidades de Geração Distribuída 34
5.5 Resultados para o Sistema Desequilibrado 35
5.5.1 Cargas Desequilibradas sem a inserção das Unidades de Geração Distribuída 36
5.5.2 Cargas Desequilibradas com a inserção das Unidades de Geração Distribuída 37
5.6 Considerações sobre os Resultados 38

6 Conclusões 39
6.1 Trabalhos Futuros 40
6.2 Publicações 40

Referências Bibliográficas 41

A Algoritmos Construídos 44
A.0.1 OpenDSSDirect.py 44
A.1 Método CSR 47
A.1.1 Integrado ao Método de Jacobi 47
A.1.2 Integrado ao Método de Gauss-Seidel 56
A.2 Método CSC 65
A.2.1 Integrado ao Método de Jacobi 65
A.2.2 Integrado ao Método de Gauss-Seidel 75
A.3 Método CSV 84
A.3.1 Integrado ao Método de Jacobi 84
A.3.2 Integrado ao Método de Gauss-Seidel 95
A.4 Método Skyline 105
A.4.1 Integrado ao Método de Jacobi 105
A.4.2 Integrado ao Método de Gauss-Seidel 115
A.5 Método DFA2 125
A.5.1 Integrado ao Método de Jacobi 125
A.5.2 Integrado ao Método de Gauss-Seidel 135

B Tempos Aferidos **147**

B.1 Tempos Aferidos - Sistema Equilibrado (Sem GD) 147

B.2 Tempos Aferidos - Sistema Equilibrado (Com GD) 149

B.3 Tempos Aferidos - Sistema Desequilibrado (Sem GD) 151

B.4 Tempos Aferidos - Sistema Desequilibrado (Com GD) 153

Lista de Figuras

3.1 Pequena rede genérica 13

4.1 Exemplos de equipamentos e suas descrições 21
4.2 Fragmento do Algoritmo utilizando *opendssdirect* 21
4.3 Parte do algoritmo CSR - Jacobi 22
4.4 Parte do algoritmo CSR - Gauss-Seidel 24

5.1 Diagrama unifilar da rede elétrica 27
5.2 Médias do Uso da Memória por Método - Sistema Equilibrado (Sem GD) 33
5.3 Médias do Uso da Memória por Método - Sistema Equilibrado (Com GD) 34
5.4 Médias do Uso da Memória por Método - Sistema Desequilibrado (Sem GD) 37
5.5 Médias do Uso da Memória por Método - Sistema Desequilibrado (Com GD) 38

Lista de Tabelas

2.1 Resumo da Revisão Bibliográfica 7

5.1 Barras. 28
5.2 Linhas de Transmissão. 28
5.3 Transformadores. 29
5.4 Cargas. 29
5.5 Medidas - Tempos Aferidos - Sistema Equilibrado (Sem GD) 33
5.6 Medidas - Tempos de Processamento - Sistema Equilibrado (Com GD) 34
5.7 Medidas - Tempo de Processamento - Sistema Desequilibrado (Sem GD) 36
5.8 Medidas - Tempo de Processamento - Sistema Desequilibrado (Com GD) 37

B.1 Tempos Aferidos - Sistema Equilibrado (Sem GD) 147
B.2 Tempos Aferidos - Sistema Equilibrado (Sem GD) 148
B.3 Tempos Aferidos - Sistema Equilibrado (Com GD) 149
B.4 Tempos Aferidos - Sistema Equilibrado (Com GD) 150
B.5 Tempos Aferidos - Sistema Desequilibrado (Sem GD) 151
B.6 Tempos Aferidos - Sistema Desequilibrado (Sem GD) 152
B.7 Tempos Aferidos - Sistema Desequilibrado (Com GD) 153
B.8 Tempos Aferidos - Sistema Desequilibrado (Com GD) 154

Capítulo 1

Introdução

1.1 Contextualização

Os sistemas elétricos de potência são estruturas constituídas por equipamentos responsáveis pela geração, transmissão, distribuição e consumo da energia elétrica. Devido à complexidade e função estratégica para qualquer nação, seu gerenciamento requer o máximo rigor: falhas durante o funcionamento podem causar interrupções na operação ou danos irreversíveis à sua estrutura.

O surgimento de novos equipamentos, como a geração distribuída, e a emergente necessidade de modernização das redes exigiram transformações nos modos de comandar e gerir os sistemas elétricos. Readequar métodos de análise tornou-se necessário, pois, manter a harmonia entre as partes integrantes é fundamental para o funcionamento. Com essa conjuntura, o uso da engenharia computacional e modelagem matemática tornou-se indispensável para os sistemas elétricos e está presente na pesquisa, planejamento, implantação e operação destes. Além de trabalhar com as redes, equipamentos e suas características, as ferramentas digitais descrevem e usam padrões que tipificam erros (falhas) cujas ocorrências são previstas.

Com o advento e avanço da computação, a simulação digital em tempo real (SDTR) consiste numa importante ferramenta de análise, com aplicações na operação, projeto, planejamento e ampliação dos sistemas elétricos. Modelar os equipamentos que compõem as redes elétricas é essencial para o funcionamento dessas ferramentas digitais. Denominados softwares de simulação, reproduzem, computacionalmente, o comportamento e desempenho das redes por meio dos modelos. Dentre os simuladores, destacam-se o Real-Time Digital Simulator (RTDS®) (RTDS, 2022), desenvolvido pela RTDS Technologies, e o HYPERSIM (OPAL-RT, 2022a) e eMEGASIM (OPAL-RT, 2022b), produzidos pela OPAL-RT Technologies.

As redes elétricas, invariavelmente, possuem muitas barras e equipamentos conectados. Por essa razão, os sistemas de equações resultantes destas têm grandes

dimensões, necessitando assim, de um grande esforço computacional para se obter as soluções requeridas. Além da dimensão, algumas matrizes podem ser esparsas, isto é, possuírem pequena porcentagem de elementos diferentes de zero (Tewarson, 1973). Por esses motivos, torna-se necessária a utilização de métodos para que se reduza o esforço computacional e possibilite a simulação em tempo real.

Baseado no sistema de equações lineares $I = YV$, onde I é a matriz que contém os valores das correntes, Y é a matriz admitâncias de barra e V, a matriz de tensões da rede, este documento descreve a construção de algoritmos com o intuito de contribuir com os sistemas de simulação em tempo real.

A metodologia adotada neste trabalho para a montagem da matriz Y é a mesma utilizada pelo *software* OpenDSS. Neste, são consideradas as matrizes de admitância dos equipamentos que compõem a rede para montar a matriz do sistema. Desta forma, pode-se inserir alterações nos equipamentos e em suas operações, e adicionar modelos dinâmicos de equipamentos e seus sistemas de controle. Além disso, o *software* é capaz de atender diferentes topologias da rede elétrica e, principalmente, redes elétricas inteligentes ou com geração distribuída, em que o sentido do fluxo de energia muda com muita frequência (Andrade et al., 2020).

Com o intuito de auxiliar a otimização dos processos referentes à resolução do sistema de equações lineares, tendo a matriz admitância de barra como objeto de aplicação, serão analisados alguns métodos de armazenamento para matrizes esparsas. Descritas no Capítulo 3, essas técnicas são denominadas *Compressed Sparse Row* (CSR), *Compressed Sparse Column* (CSC), *Compressed Sparse Vector* (CSV), Skyline e o método proposto em Jiang et al. (2015), que será chamado de DFA2. Como principal característica comum entre elas, destaca-se a divisão da matriz esparsa em matrizes linha, que contém os elementos não-nulos e suas respectivas coordenadas, definidas em cada método.

O OpenDSS utiliza os métodos CSR e CSC, associados a um método do ponto fixo iterativo ou ao método de "Newton"(Rocha and Radatz, 2018b), através do software KLUSolve (Reynolds et al., 2020). Ele é uma biblioteca de funções relacionadas a matrizes complexas esparsas adaptado a sistemas elétricos de potência (Rocha and Radatz, 2018b) e é responsável por montar a matriz de admitância, armazenar e efetuar operações com ela.

Por meio dessas técnicas, associadas aos métodos de Jacobi e Gauss-Seidel, pretende-se despender a menor quantidade de tempo, memória e operações, a fim de colaborar com a simulação em tempo real. Como referência para essa comparação, o OpenDSS, através do *OpenDSSDirect.py*, também foi submetido as análises propostas. O emprego desses métodos citados e seus respectivos resultados serão apresentados no Capítulo 5.

1.2 Objetivos

O objetivo desse trabalho é comparar as técnicas de armazenamento de matrizes associadas aos métodos iterativos de Jacobi e Gauss-Seidel visando otimizar o desempenho computacional, diminuindo tempo de processamento, memória utilizada e operações despendidas, com o intuito de contribuir para a viabilização da simulação em tempo real.

Além deste objetivo principal, foram concebidos os objetivos específicos:

- implementar os métodos de armazenamento na linguagem de programação proposta (Python) com o intuito de, futuramente, formar uma biblioteca;
- construir a associação entre os métodos de armazenamento e iterativos para futura transformação em sub-rotinas.

1.3 Metodologia

O início deste trabalho foi dedicado a compreensão do funcionamento dos equipamentos elétricos e suas respectivas equações, por meio dos conteúdos encontrados em Stevenson (1986), Monticelli and Garcia (2011) e Kagan et al. (2017). A partir disto, possibilitou-se a modelagem destes, partindo de modelos matemáticos consolidados na literatura, presentes em Kersting. (2001), Arrillaga and Watson (2001) e Izudin et al. (2015).

Dado o sistema de equações lineares, junto aos conteúdos adquiridos anteriormente, foi necessário adotar um modelo para a construção da matriz de admitância, apresentada em Branin (1967) e foi utilizada a técnica descrita em Rocha and Radatz (2018b). Devido ao uso difundido e a fácil adaptação a possíveis alterações de funcionamento de redes elétricas, adotou-se a forma proposta pelo OpenDSS, que utiliza as matrizes de cada componente do sistema.

Após esta definição, a rede elétrica foi escolhida; ela representa um sistema real e está descrita no Capítulo 5. Para gerar sua matriz de admitância, construiu-se um algoritmo que associou os modelos matemáticos de cada equipamento e o método construtivo adotado.

A presença de muitos termos nulos em sua composição, originando a esparsidade, demandou a busca por métodos que aplicassem técnicas com o intuito de usar apenas os termos diferentes de zero para os cálculos, reduzindo o esforço computacional e o tempo necessário para processá-los. Os conteúdos inclusos em de Oliveira Coelho (2014) e de Oliveira (2019) foram os primeiros encontrados. Apesar de conterem a associação a métodos iterativos, deu-se atenção aos métodos de armazenamento de matrizes e suas respectivas referências.

A escolha dos métodos de armazenamento de matrizes exigiu sua posterior implentação na linguagem de programação Python. Partindo da meta de otimizar os processos em tempo real, propôs-se comparar o tempo e a memória utilizada para execução, utilizando funções próprias da linguagem selecionada (*time.process_time()* e *psutil.virtual_memory()*). Como consequência desta comparação, possibilitou-se estimar qual destas técnicas teriam melhor desempenho.

Posterior a estes processos, fez-se necessário agregar estruturas para propiciar a resolução do sistema de equações. Isso posto, em razão das propriedades inerentes, os métodos iterativos de Jacobi e Gauss-Seidel, apresentados em Cunha (2018) e Filho (2017), foram empregados . Para o êxito das operações, é necessário adaptá-los aos produtos originários dos métodos de armazenamento, requisitando tempo e testes dos códigos construídos.

Por fim, é imperativo escolher uma referência para embasar a comparação estabelecida. Assim, após diversas tentativas inexitosas de reproduzir os mecanismos presentes no OpenDSS em Python, buscou-se a aplicação do *OpenDSSDirect.py*, que é uma estrutura que permite executar o arquivo em formato dss, próprio do *software*, em Python.

1.4 Estrutura do Dissertação

Com o intuito de explicar e embasar os estudos, este documento possui mais cinco partes. No Capítulo 2, apresenta-se as principais referências bibliográficas utilizadas, bem como, comentários e destaques sobre seus usos. O Capítulo 3 descreve a modelagem dos equipamentos que compõem a rede elétrica, a matriz admitância de barra, sua respectiva construção, os métodos de armazenamento das matrizes esparsas utilizados e os métodos de Jacobi e Gauss-Seidel.

O Capítulo 4 exibe os métodos utilizados na construção da dissertação, com o devido detalhamento. O Capítulo 5 traz a descrição da rede elétrica utilizada e os resultados obtidos, com suas respectivas análises. Por fim, o Capítulo 6 apresenta as conclusões, os possíveis trabalhos futuros e as publicações advindas do trabalho executado. Os Apêndices A e B exibem os algoritmos dos métodos de armazenamento associados aos métodos de iterativos e os tempos aferidos no processamento destes.

Capítulo 2

Revisão Bibliográfica

Este capítulo abordará a revisão bibliográfica sobre os métodos de armazenamento utilizados neste trabalho. Após as devidas discussões, uma tabela trará o resumo das obras citadas e suas respectivas contribuições.

2.1 Principais Referências, Comparações e Discussões

O desenvolvimento das técnicas numéricas de armazenamento deve-se à necessidade de efetuar cálculos e obter resultados com menor tempo de processamento e esforço computacional, possibilitando a realização de mais tarefas com os equipamentos disponíveis. Seu surgimento adveio do trabalho com matrizes de grandes dimensões e que tinham como principal característica, mais elementos nulos do que diferentes de zero, gerando uma dispersão destes. Por esse motivo, essas matrizes foram denominadas esparsas.

As operações com matrizes esparsas constituem uma parte significativa de muitos cálculos científicos e a irregularidade das operações induzidas pelos padrões de esparsidade muitas vezes inibe o uso eficiente dos recursos computacionais das arquiteturas escalares e vetoriais convencionais (Stathis et al., 2003). Por esta razão, técnicas de armazenamento de matrizes esparsas foram desenvolvidas. A quantidade total de armazenamento necessária deve ser a menor possível sem impedir o uso eficiente da estrutura de dados (He, 2003).

Devido às diferentes características das matrizes esparsas e às diferentes formas de usar seus dados, muitas maneiras diferentes de armazenar matrizes esparsas foram criadas para aproveitar a estrutura das matrizes ou a especificidade do problema do qual elas surgem (Dutto et al., 2000).

Neste trabalho, são apresentados e utilizados cinco métodos de armazenamento, que serão descritos no Capítulo 3: *Compressed Sparse Row* (CSR), *Compressed*

Sparse Column (CSC), *Compressed Sparse Vector* (CSV), Skyline e DFA2. Os quatro primeiros são encontrados em outros artigos e publicações, porém, essa dissertação propõe a análise comparativa destes com a adição de um método construído com o intuito de otimizar o armazenamento de dados em dispositivos *FPGA (field-programmable gate array)*, que é o DFA2.

Os métodos CSR e CSC, apresentados em Rose and Willoughby (1972), propõem guardar os elementos não nulos da matriz e seus respectivos índices, inerentes à metodologia proposta pelos autores, em três matrizes linha. Pelo amplo uso e difusão, estes métodos são referências quando há comparações com os demais. O método Skyline (de Oliveira Coelho, 2014) propõe procedimento semelhante às técnicas citadas no parágrafo anterior. O método CSV, descrito em Farzaneh et al. (2009), apresenta uma importante diferença com relação aos índices dos termos diferentes de zero, utilizando uma regra diferente para obtê-los.

Em comum, além do objetivo, podemos classificá-los como métodos de formato geral, com armazenamento baseado em pontos (Shahnaz et al., 2006). Isto se deve ao fato de não requisitarem uma estrutura para a distribuição dos termos não nulos e da relação entre estes termos e os elementos das matrizes resultantes da aplicação de seus procedimentos: cada entrada é um elemento único da matriz.

O método DFA2 (Jiang et al., 2015) também possui armazenamento baseado em pontos, mas, necessita de um rearranjo dos termos não nulos para produzir as demais matrizes inerentes à técnica; por este motivo, pode ser classificado como método de formato específico (Shahnaz et al., 2006).

Segundo Langr and Tvrdík (2016), o método CSR destaca-se pela simplicidade e facilidade de implementação. Neste artigo, ressalta-se também a constatação que a eficiência de um método de armazenamento deve estar atrelada a estruturas e algoritmos de simples implantação, atributos que, durante a construção deste trabalho, foram observados no uso dos métodos CSR e CSC. Métodos como CSV e Skyline propõe reduzir a quantidade de dados transferidos, seja na redução de matrizes de armazenamento ou na captura de menos índices referentes aos termos diferentes de zero; porém, possuem algoritmos que exigem mais estruturas de repetição para registrar os dados necessários, o que dificulta melhor desempenho.

Durante a construção das associações aos métodos iterativos de Jacobi e Gauss-Seidel (Cunha, 2018), empregadas na resolução dos sistemas de equações advindos da análise da rede, foi possível atestar, assim como em Shahnaz et al. (2006) e Langr and Tvrdík (2016), que os métodos CSR e CSC destacam-se com relação aos demais, pois aliam a baixa complexidade de implementação e melhor desempenho na multiplicação vetor-matriz esparsa.

Por fim, a Tabela 2.1 condensa as principais referências sobre os métodos de armazenamento de matrizes esparsas utilizadas durante o mestrado.

Tabela 2.1: Resumo da Revisão Bibliográfica

Resumo da Revisão Bibliográfica			
Autores	Título	Ano	Contribuição
Rose, Donald J., Willoughby, Ralph A.	Sparse Matrices and their Applications	1972	Métodos de Armazenamento CSR (*Compressed Sparse Row*) e CSC (*Compressed Sparse Column*) .
Shahnaz, Rukhsana Usman, Anila Chughtai, Imran R.	Review of Storage Techniques for Sparse Matrices	2006	Descrição, classificação e comparativo entre métodos de armazenamento
Farzaneh, Aiyoub Kheiri, Hossein Abbaspour, Mehdi	An efficient storage format for large sparse matrices	2009	Método de Armazenamento CSV (*Compressed Sparse Vector*).
Bell, Nathan Garland, Michael	Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors	2009	Comparativo entre métodos de armazenamento
Coelho, Marco Antonio de O.	Métodos Iterativos para Resolver Sistemas de Equações Algébricas Lineares em Estruturas Esparsas	2014	Método de Armazenamento Skyline.
Jiang, Lei Tan, Jianlong Tang, Qiu	An efficient sparse matrix format for accelerating regular expression matching on field-programmable gate arrays	2015	Método de Armazenamento DFA2.
Langr, Daniel Tvrdík, Pavel	Evaluation Criteria for Sparse Matrix Storage Formats	2016	Avaliação dos métodos de armazenamento de matriz esparsa
Cunha, Maria Cristina C.	Métodos Numéricos	2018	Métodos Iterativos.

Capítulo 3

Fundamentação Teórica

Este capítulo trata da fundamentação teórica e dos procedimentos que precederam a concepção da proposta para a dissertação. Abordam-se os aspectos e características dos estudos desenvolvidos até a elaboração deste texto, com suas respectivas referências.

A Seção 3.1 exibe a modelagem utilizada para os componentes da rede elétrica. A Seção 3.2 traz a descrição e o processo de construção da matriz admitância de barra de uma rede elétrica baseado no método do OpenDSS. A Seção 3.3 cita o sistema de equações a resolver e explica os métodos de armazenamento, descrevendo cada um dos elencados. A Seção 3.4 exibe um breve resumo sobre os métodos de armazenamento descritos. A Seção 3.5 apresenta os métodos iterativos utilizados, com suas respectivas equações.

3.1 Modelagem dos Componentes da Rede Elétrica

Descrever as características e equações dos elementos que compõem a rede é fundamental para estabelecer os dados necessários e a forma como serão construídos e funcionarão os algoritmos. Estes, fornecem a matriz admitância de barra e o vetor de tensões nodais.

Para a construção da matriz admitância de barra da rede elétrica, é necessário obter as matrizes admitância de cada elemento. Os modelos matemáticos escolhidos para cada componente requerem dados diferentes para o cálculo das admitâncias: para as linhas de transmissão, são necessárias as impedâncias próprias e mútuas; para os transformadores, os valores das impedâncias de sequência zero e positiva; para as cargas, a potência aparente, o fator de potência e a tensão nominal; e, para as unidades de geração distribuída, a potência ativa e a tensão nominal.

3.1.1 Linhas de Transmissão

A rede elétrica selecionada, que será apresentada no Capítulo 5, possui oito linhas de transmissão. Elas não forneceram, diretamente, o que é requerido pelo modelo adotado. Desta forma, empregaram-se equações auxiliares para obtê-los. A partir da geometria dos cabos múltiplos e das medidas dispostas, empregam-se as seguintes equações, presentes em (Stevenson, 1986):

$$RMG = \sqrt{r_a d} \quad m \tag{3.1}$$

$$RMG = \sqrt[3]{r_a d^2} \quad m \tag{3.2}$$

nas quais, RMG significa raio médio geométrico, r_a é o valor do raio do cabo e d, a distância entre os condutores.

Devido à diferença de configuração, a equação (3.1) é utilizada para calcular o raio médio geométrico das linhas numeradas de dois a oito, que possuem dois condutores por fase. A equação (3.2) é utilizada para o cálculo do raio médio geométrico da linha um, pois esta possui três condutores por fase.

$$DMG = \sqrt[3]{D_{12} D_{23} D_{31}} \quad m \tag{3.3}$$

A equação (3.3) é empregada para calcular a distância média geométrica (DMG) das três fases que formam a linha, onde $D_{i,j}$ é a distância entre as fase i e j ($i, j = 1, 2, 3. i \neq j$).

Com os valores de RMG e DMG calculados, obtém-se as impedâncias requeridas por meio das equações (3.4) e (3.5), denominadas Equações de Carson modificadas (Kersting., 2001). Em (3.4), r_i significa a resistência elétrica CC do condutor a 20ºC.

$$Z_p = r_i + 0,09530 + j0,12134 \left(ln \frac{1}{RMG} + 7,93402 \right) \Omega/milha \tag{3.4}$$

$$Z_m = 0,09530 + j0,12134 \left(ln \frac{1}{DMG} + 7,93402 \right) \Omega/milha \tag{3.5}$$

A partir desses valores, a matriz de impedâncias da linha de transmissão é obtida. Posteriormente, calcula-se a matriz de admitância.

$$Z_{LT} = \begin{bmatrix} Z_p & Z_m & Z_m \\ Z_m & Z_p & Z_m \\ Z_m & Z_m & Z_p \end{bmatrix} \tag{3.6}$$

$$Y_{LT} = \begin{bmatrix} [Z_{LT}^{-1}] & [-Z_{LT}^{-1}] \\ [-Z_{LT}^{-1}] & [Z_{LT}^{-1}] \end{bmatrix} \tag{3.7}$$

3.1.2 Transformadores

Para os transformadores, são fornecidas as tensões do primário e secundário, a potência nominal e as reatâncias de sequência positiva e zero em p.u.. Visto que, neste trabalho, não foram utilizadas as grandezas em p.u., obtemos os valores das reatâncias em *ohms*, considerando a impedância base do primário. Os demais passos para obtenção da matriz admitância dos transformadores foram executados conforme o modelo encontrado em (Izudin et al., 2015).

Com os valores dessas impedâncias, forma-se a matriz abaixo e, em seguida, obtém-se a matriz de admitância das sequências.

$$Z_{012} = \begin{bmatrix} Z_0 & 0 & 0 \\ 0 & Z_1 & 0 \\ 0 & 0 & Z_2 \end{bmatrix} \tag{3.8}$$

$$Y_{012} = Z_{012}^{-1} = \begin{bmatrix} Y_0 & 0 & 0 \\ 0 & Y_1 & 0 \\ 0 & 0 & Y_2 \end{bmatrix} \tag{3.9}$$

Para o cálculo da matriz de admitância nas coordenadas a-b-c, utiliza-se a matriz auxiliar A:

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & a^2 & a \\ 1 & a & a^2 \end{bmatrix} \tag{3.10}$$

na qual,

$$a = -0,5 + j0,866. \tag{3.11}$$

Assim,

$$Y_P = AY_{012}A^{-1} \tag{3.12}$$

Por fim, a matriz de admitâncias nodais do transformador é calculada com o auxílio da matriz de conexões.

$$Y_{TR} = C^T Y_P C \tag{3.13}$$

A composição da matriz de conexões depende de características dos transformadores como disposição dos enrolamentos, por exemplo. Os transformadores que

estão na rede elétrica possuem a configuração DYg1. Desta forma, foi utilizada a seguinte matriz de conexão:

$$C = \begin{bmatrix} \frac{1}{\sqrt{3}} & -1 & \frac{1}{\sqrt{3}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{3}} & -1 & -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} & 0 & 0 & 0 & \frac{1}{\sqrt{3}} & -1 \end{bmatrix} \tag{3.14}$$

3.1.3 Cargas

As cargas presentes na rede elétrica são equilibradas, possuem ligação estrela-aterrado e não dispõem de acoplamento entre as fases. São fornecidas as potências ativa e reativa. Desta forma, para conseguir os dados requeridos, emprega-se a equação (3.15).

$$S^2 = P^2 + Q^2 \tag{3.15}$$

Calculada a potência aparente, calcula-se o fator de potência através da seguinte equação:

$$fp = \frac{P}{S} \tag{3.16}$$

A partir desses dados, são calculadas as impedâncias e, por fim, as admitâncias, formando a matriz seguinte.

$$Y_{CG} = \begin{bmatrix} Y & 0 & 0 \\ 0 & Y & 0 \\ 0 & 0 & Y \end{bmatrix} \tag{3.17}$$

3.1.4 Unidades de Geração Distribuída

As unidades de geração distribuída têm potência ativa de 10 MW e 20 MW, respectivamente, e fornecem a tensão nominal de 13,8 kV. Por meio destes dados, calculam-se as correntes e, por fim, as impedâncias. Abaixo, segue a matriz de impedância do componente:

$$Z_{GD} = \begin{bmatrix} Z_{aa} & Z_{ab} & Z_{ac} \\ Z_{ba} & Z_{bb} & Z_{bc} \\ Z_{ca} & Z_{cb} & Z_{cc} \end{bmatrix} \tag{3.18}$$

A matriz de admitância da unidade geradora é obtida através da inversão da matriz de impedância (3.18):

$$Y_{GD} = Z_{GD}^{-1} = \begin{bmatrix} Y_{aa} & Y_{ab} & Y_{ac} \\ Y_{ba} & Y_{bb} & Y_{bc} \\ Y_{ca} & Y_{cb} & Y_{cc} \end{bmatrix} \tag{3.19}$$

Com as matrizes dos elementos da rede, construímos a sua matriz admitância de barra. Para esta tarefa, faz-se necessário considerar as conexões das barras aos equipamentos, pois a matriz utiliza os nós como referência.

3.2 Descrição e Construção da Matriz Admitância de Barra de uma Rede Elétrica

A matriz admitância de barra é uma estrutura cujos elementos representam uma admitância e que relaciona correntes injetadas e tensões de uma rede elétrica. Visto que na sua formação são considerados os nós, que são pontos de junção entre elementos da rede, essa matriz também é denominada matriz de admitância nodal ((Rocha and Radatz, 2018b)). Sua construção é dependente das matrizes de admitâncias dos elementos que compõem a rede, também chamadas de primitivas.

A maioria dos elementos trifásicos de uma rede elétrica (linhas, transformadores e cargas) possuem matrizes de impedância com a seguinte característica: as impedâncias próprias constituem a diagonal principal e as impedâncias mútuas representam os valores fora da diagonal principal. Para obter a matriz de admitância desse elemento, além de outros cálculos, é necessário inverter a matriz de impedância.

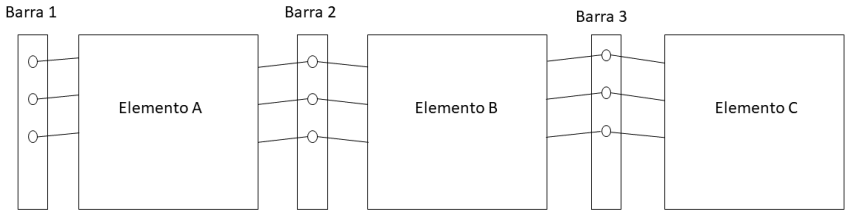
A partir das matrizes de admitância de cada elemento, é possível construir a matriz da rede. Complementando o que foi dito no primeiro parágrafo, o número de nós conectados determina o tamanho da matriz. Por isso, temos que a matriz admitância de barra é quadrada, pois possui dimensão $n \times n$, sendo n o número de nós conectados.

Em sua estrutura, cada posição corresponde aos nós conectados. Estas correspondências estão dispostas de forma crescente. Desta forma, os nós conectados da primeira barra estão relacionados as primeiras posições da matriz, seja em linha ou coluna; os da segunda barra estão relacionados com as linhas e colunas subsequentes, e assim, sucessivamente.

Para cada elemento, sua matriz de admitância é alocada nas respectivas posições que correspondem aos nós que estão conectados. Caso mais algum equipamento esteja conectado aos mesmos nós, os valores correspondentes àquelas posições serão resultado da soma das admitâncias correspondentes dos equipamentos.

Com o intuito de consolidar a metodologia descrita, consideremos a Figura 3.1 a seguir:

Figura 3.1: Pequena rede genérica



Fonte: Autoria própria. (Adaptada da figura encontrada em Rocha and Radatz (2018b))

Para a pequena rede trifásica da Figura 3.1, verificamos que há 3 barras, com 3 nós conectados para cada. Por esse motivo, a matriz terá dimensão 9 x 9. Sobre as linhas e colunas, temos que: de 1 a 3, correspondem à Barra 1; de 4 a 6, à Barra 2; e de 7 a 9, à Barra 3. Com relação aos elementos, podemos constatar que a matriz de admitância do Elemento A, conectado às Barras 1 e 2, ocupará as seis primeiras linhas e colunas; a do Elemento C, ocupará as três últimas. A matriz do Elemento B, conectado às Barras 2 e 3, ocupará as linhas e colunas de 4 a 9. Por conter nós em comum com os demais elementos, os valores das linhas e colunas de 4 a 6 e, das linhas e colunas de 7 a 9, serão resultados das somas das admitâncias do Elemento B com A e com C, respectivamente.

$$
\mathrm{Y} = \begin{bmatrix}
Y_{A_{11}} & Y_{A_{12}} & Y_{A_{13}} & Y_{A_{14}} & Y_{A_{15}} & Y_{A_{16}} & 0 & 0 & 0 \\
Y_{A_{21}} & Y_{A_{22}} & Y_{A_{23}} & Y_{A_{24}} & Y_{A_{25}} & Y_{A_{26}} & 0 & 0 & 0 \\
Y_{A_{31}} & Y_{A_{32}} & Y_{A_{33}} & Y_{A_{34}} & Y_{A_{35}} & Y_{A_{36}} & 0 & 0 & 0 \\
Y_{A_{41}} & Y_{A_{42}} & Y_{A_{43}} & Y_{A_{44}} + Y_{B_{11}} & Y_{A_{45}} + Y_{B_{12}} & Y_{A_{46}} + Y_{B_{13}} & Y_{B_{14}} & Y_{B_{15}} & Y_{B_{16}} \\
Y_{A_{51}} & Y_{A_{52}} & Y_{A_{53}} & Y_{A_{54}} + Y_{B_{21}} & Y_{A_{55}} + Y_{B_{22}} & Y_{A_{56}} + Y_{B_{23}} & Y_{B_{24}} & Y_{B_{25}} & Y_{B_{26}} \\
Y_{A_{61}} & Y_{A_{62}} & Y_{A_{63}} & Y_{A_{64}} + Y_{B_{31}} & Y_{A_{65}} + Y_{B_{32}} & Y_{A_{66}} + Y_{B_{33}} & Y_{B_{34}} & Y_{B_{35}} & Y_{B_{36}} \\
0 & 0 & 0 & Y_{B_{41}} & Y_{B_{42}} & Y_{B_{43}} & Y_{C_{11}} + Y_{B_{44}} & Y_{C_{12}} + Y_{B_{45}} & Y_{C_{13}} + Y_{B_{46}} \\
0 & 0 & 0 & Y_{B_{51}} & Y_{B_{52}} & Y_{B_{53}} & Y_{C_{21}} + Y_{B_{54}} & Y_{C_{22}} + Y_{B_{55}} & Y_{C_{23}} + Y_{B_{56}} \\
0 & 0 & 0 & Y_{B_{61}} & Y_{B_{62}} & Y_{B_{63}} & Y_{C_{31}} + Y_{B_{64}} & Y_{C_{32}} + Y_{B_{65}} & Y_{C_{33}} + Y_{B_{66}}
\end{bmatrix} \tag{3.20}
$$

Após a construção da matriz em (3.20), nota-se que ela possui poucos termos nulos. Porém essa situação cresce para as grandes redes, de tal forma que a quantidade destes supera a de termos não nulos.

3.3 Sistema de Equações e Métodos de Armazenamento de Matrizes

A partir da construção da matriz admitância de barra da rede, torna-se possível

abordar o sistema de equações

$$I = YV. \tag{3.21}$$

Através deste, podemos calcular a matriz de tensões nodais, o que possibilita obter outros valores de interesse da rede elétrica.

Para a resolução do sistema de equações, o uso da matriz admitância de barra de uma rede na sua forma densa causa grande esforço computacional. Como citado na seção anterior, essa matriz é considerada esparsa, isto é, possui grande quantidade de termos nulos, superando o número de não nulos.

O interesse no desenvolvimento dos métodos de armazenamento surgiu com a ampliação da capacidade computacional para armazenar e processar dados, advinda do progresso tecnológico. Percebeu-se que algumas grandes matrizes obtidas possuíam muitos elementos nulos, o que causava desperdício de tempo e memória, atrasando o alcance dos resultados requeridos. À vista disso, foram elaborados métodos que memorizavam os elementos não nulos, essenciais para os cálculos necessários, junto com coordenadas, definidas pelos autores dessas técnicas.

A seguir, são apresentados cinco métodos que proporcionam este ganho computacional. Para ilustrar as matrizes de cada método, consideremos a seguinte matriz:

$$A = \begin{bmatrix} Y_1 & Y_2 & 0 & 0 & 0 & 0 \\ 0 & Y_3 & Y_4 & 0 & 0 & 0 \\ Y_5 & 0 & Y_6 & 0 & 0 & 0 \\ 0 & Y_7 & 0 & Y_8 & 0 & 0 \\ 0 & 0 & 0 & 0 & Y_9 & 0 \\ 0 & 0 & 0 & 0 & 0 & Y_{10} \end{bmatrix} \tag{3.22}$$

3.3.1 Método CSR (Compressed Sparse Row)

Proposto por Fred G. Gustavson no artigo "Some Basic Techniques for Solving Sparse Systems of Linear Equations", presente em (Rose and Willoughby, 1972), o método CSR é o mais difundido e utilizado. Ele armazena a matriz esparsa em três matrizes linha, denominadas AN, JA e IA, desprezando os elementos nulos. A quantidade de colunas de AN e JA será dada pelo número de elementos não nulos da matriz esparsa.

A matriz AN armazena os elementos não nulos da matriz esparsa, na sequência em que aparecem nas linhas, que são percorridas de forma crescente. A matriz JA guarda os índices da coluna do elemento não nulo, seguindo a ordem de AN; e IA, a quantidade de elementos não nulos por linha, de forma cumulativa, sendo percorrida

da linha 1 a i-ésima linha mais 1. Assim, IA possui dimensão n+1, sendo n a ordem da matriz esparsa. Convecionou-se que o primeiro termo de IA deve ser igual a zero ou um; no nosso estudo, utilizaremos o zero. O segundo termo representa a soma da quantidade de elementos não nulos da primeira linha com o primeiro termo da matriz (zero ou um).

Do ponto de vista computacional, esse método é de fácil implementação, pois seu algoritmo necessita de poucos laços para obter as matrizes de armazenamento.

Considerando a matriz em (3.22), temos que as matrizes AN, JA e IA serão:

$$AN = \begin{bmatrix} Y_1 & Y_2 & Y_3 & Y_4 & Y_5 & Y_6 & Y_7 & Y_8 & Y_9 & Y_{10} \end{bmatrix} \tag{3.23}$$

$$JA = \begin{bmatrix} 1 & 2 & 2 & 3 & 1 & 3 & 2 & 4 & 5 & 6 \end{bmatrix} \tag{3.24}$$

$$IA = \begin{bmatrix} 0 & 2 & 4 & 6 & 8 & 9 & 10 \end{bmatrix} \tag{3.25}$$

3.3.2 Método CSC (Compressed Sparse Row)

Trata-se de uma adaptação do método CSR. Conforme visto em (de Oliveira Coelho, 2014) e (de Oliveira, 2019), difere-se deste na formação das matrizes de armazenamento, pois, a matriz esparsa é percorrida por colunas, ao invés de linhas.

Percorrendo as colunas de forma crescente, a matriz AN armazena os elementos não nulos da matriz esparsa; JA, os índices da linha do elemento não nulo, seguindo a ordem de AN. A matriz IA possui dimensão n+1 e guarda a quantidade de elementos não nulos por coluna, de forma cumulativa, sendo percorrida da coluna 1 a i-ésima coluna mais 1. Convecionou-se que o primeiro termo de IA deve ser igual a zero ou um; no nosso estudo, utilizaremos o zero. O segundo termo representa a soma da quantidade de elementos não nulos da primeira coluna com o primeiro termo da matriz (zero ou um).

Para mostrar a diferença com relação ao método CSR e as matrizes desse método, a partir da matriz em (3.22), temos que:

$$AN = \begin{bmatrix} Y_1 & Y_5 & Y_2 & Y_3 & Y_7 & Y_4 & Y_6 & Y_8 & Y_9 & Y_{10} \end{bmatrix} \tag{3.26}$$

$$JA = \begin{bmatrix} 1 & 3 & 1 & 2 & 4 & 2 & 3 & 4 & 5 & 6 \end{bmatrix} \tag{3.27}$$

$$IA = \begin{bmatrix} 0 & 2 & 5 & 7 & 8 & 9 & 10 \end{bmatrix} \tag{3.28}$$

3.3.3 Método CSV (Compressed Sparse Vector)

Apresentado em (Farzaneh et al., 2009), armazena a matriz esparsa em duas matrizes linha: em AA, onde estão os elementos não nulos da matriz esparsa e, em IA, em que se encontram os índices referentes aos elementos de AA, percorridos por linha, sendo estes referentes a contagem dos números. A ordem dessas matrizes é dada pelo número de elementos não-nulos da matriz esparsa.

Os valores alocados em IA são determinados pela seguinte regra: a contagem é iniciada no primeiro termo da matriz, percorrendo as linhas. Após cada termo diferente de zero, a contagem é reiniciada. Consideremos o seguinte exemplo: dada uma matriz quadrada de ordem 4, a primeira linha é composta por dois valores diferentes de zero, sendo que um deles está localizado na primeira posição e o outro, na última. Dessa forma, os seus respectivos valores correspondentes em IA serão 1 e 3.

Considerando a matriz em (3.22), AA e IA serão:

$$AA = \begin{bmatrix} Y_1 & Y_2 & Y_3 & Y_4 & Y_5 & Y_6 & Y_7 & Y_8 & Y_9 & Y_{10} \end{bmatrix} \tag{3.29}$$

$$IA = \begin{bmatrix} 1 & 1 & 6 & 1 & 4 & 2 & 5 & 2 & 7 & 7 \end{bmatrix} \tag{3.30}$$

3.3.4 Método Skyline

Conforme proposto em (de Oliveira Coelho, 2014), este método consiste no armazenamento da matriz esparsa em três matrizes linha, as quais serão denominadas VAL, Rowptr e Fstcol. Em VAL, serão armazenados os elementos diferentes de zero; caso estes estejam na mesma linha da matriz esparsa e possuam zeros entre si, estes também são armazenados.

Em Fstcol, são guardados os índices referentes à coluna do primeiro elemento não nulo de cada linha. Em Rowptr, está armazenada a quantidade de elementos armazenados por linha, com dimensão n+1, sendo n a dimensão da matriz. Sua estrutura é igual a das matrizes IA dos métodos CSR e CSC. Por isso, o seu primeiro termo também será igual a zero.

Seguindo a matriz em (3.22), temos que:

$$VAL = \begin{bmatrix} Y_1 & Y_2 & Y_3 & Y_4 & Y_5 & 0 & Y_6 & Y_7 & 0 & Y_8 & Y_9 & Y_{10} \end{bmatrix} \tag{3.31}$$

$$Fstcol = \begin{bmatrix} 1 & 2 & 1 & 2 & 5 & 6 \end{bmatrix} \tag{3.32}$$

$$Rowptr = \begin{bmatrix} 0 & 2 & 4 & 7 & 10 & 11 & 12 \end{bmatrix} \tag{3.33}$$

3.3.5 Método DFA2

Apresentado em (Jiang et al., 2015), esse método baseia-se em três requisitos: embalagem de memória, tabela de índices e memórias intercaladas. A motivação para o seu desenvolvimento foi o armazenamento eficiente da matriz esparsa em um FPGA (*Field Programmable Gate Array*). Por eficiente, entende-se um armazenamento que ocupa pouco espaço em relação à totalidade da matriz esparsa, e facilita a realização das operações necessárias, diminuindo o tempo de processamento. Neste primeiro momento, serão descritos os requisitos "embalagem de memória"e "tabela de índices".

Orientado pela compressão da matriz esparsa, onde são eliminados todos os elementos iguais a zero, o método propõe a criação de cinco matrizes: matriz compactada, ROWID, INDEX, OFFSET e #NZ. Destas, a matriz compactada é a única matriz que possui mais de uma linha. Ela contém os termos não nulos da matriz e representa o requisito "embalagem de memória". Estruturalmente, possui a quantidade de linhas da matriz esparsa e número de colunas compatível ao número total de elementos diferentes de zero e máximo de elementos não nulos por linha. Além dessa compatibilidade, o número de colunas é adaptado as especificações do FPGA no qual o método será implementado. Para facilitar o entendimento das demais matrizes, chamaremos a matriz compactada de VAL.

A matriz ROWID traz os números correspondentes as linhas da matriz esparsa. INDEX considera a matriz VAL e é preenchida pelo índice referente a linha de VAL do primeiro elemento não nulo das linhas da matriz esparsa. OFFSET utiliza a mesma metodologia para a formação de INDEX. Porém, são armazenados os índices referentes à coluna, em VAL, do elemento não nulo. #NZ carrega o número de elementos não nulos de cada linha da matriz esparsa. Essas matrizes representam o requisito "tabela de índices". Como descrito em (Jiang et al., 2015), as matrizes ROWID, INDEX e OFFSET, iniciam a contagem pelo número 0. Logo, seus primeiros termos serão iguais a zero.

Para a matriz em (3.22), as matrizes propostas por este método serão:

$$VAL = \begin{bmatrix} Y_1 & Y_2 \\ Y_3 & Y_4 \\ Y_5 & Y_6 \\ Y_7 & Y_8 \\ Y_9 & Y_{10} \\ 0 & 0 \end{bmatrix} \tag{3.34}$$

$$ROWID = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix} \tag{3.35}$$

$$INDEX = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 4 \end{bmatrix} \tag{3.36}$$

$$OFFSET = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.37}$$

$$\#NZ = \begin{bmatrix} 2 & 2 & 2 & 2 & 1 & 1 \end{bmatrix} \tag{3.38}$$

3.4 Resumo sobre Métodos de Armazenamento de Matrizes

A Seção 3.3 apresentou os métodos de armazenamento de matrizes utilizados neste trabalho: CSR, CSC, CSV, Skyline e DFA2. Estes, possuem diversas semelhanças e diferenças, além de objetivo comum: armazenar os termos diferentes de zero de uma matriz esparsa.

Os métodos CSR e CSC utilizam a matriz linha AN para guardar os elementos não nulos da matriz esparsa. JA e IA, demais matrizes decorrentes do emprego destas técnicas, reúnem os índices referentes a estes termos e a soma deles, respectivamente. A principal diferença entre estes métodos advém do fato da leitura da matriz esparsa para coleta dos dados necessários: para o CSR, percorre-se a matriz por linha e, para o CSC, por coluna.

O método CSV, ao contrário dos demais, usa, apenas, duas matrizes linha para o armazenamento: AA, que guarda os elementos não nulos e IA, que possui os índices referentes aos elementos de AA. Estes valores são dispostos através de uma regra de contagem, descrita na Subseção 3.3.3.

O método Skyline emprega as matrizes linha VAL, Rowptr e Fstcol para o armazenamento. A estrutura das duas primeiras é semelhante às matrizes AN e IA, respectivamente, utilizadas pelos métodos CSR e CSC. A principal diferença dá-se no processo de coleta dos valores não nulos: no Skyline, guardam-se os zeros presentes entre estes termos que estão na mesma linha. A matriz Fstcol armazena os índices referentes à coluna do primeiro elemento não nulo de cada linha.

O método DFA2 propõe o uso de cinco matrizes. Diferente das demais técnicas descritas, a matriz que armazena os valores diferentes de zero da matriz esparsa, denominada VAL, não é uma matriz linha; ela possui o mesmo número de linhas da matriz esparsa e número de colunas compatível ao número de elementos não nulos. As outras quatro matrizes são preenchidas por índices correspondentes aos

valores de VAL, sendo que INDEX e OFFSET consideram a disposição dos termos de VAL e, ROWID e #NZ, a disposição dos termos na matriz alvo das técnicas de armazenamento.

3.5 Métodos Iterativos de Jacobi e Gauss-Seidel

Os métodos iterativos se baseiam na construção de sequência de aproximações; a cada passo, os valores calculados anteriormente são usados para melhorar a aproximação (Cunha, 2018). Eles partem de uma estimativa inicial da solução e geram, sucessivamente, valores mais próximos da solução (Filho, 2017).

O método de Jacobi foi desenvolvido por Carl Gustav Jacobi (1804-1851) e consiste em exibir, a cada equação, uma das variáveis do sistema. O método de Gauss-Seidel foi proposto por Philip von Seidel (1821-1896) e é oriundo de estudos realizados por Gauss; consiste numa modificação do método de Jacobi, onde as variáveis já calculadas são usadas nas equações subsequentes do sistema (Filho, 2017).

O uso desses métodos iterativos se deve ao fato de serem indicados para resolução de sistemas de equações lineares grandes e esparsos; além disso, sua resolução por etapas permite o devido acompanhamento dos valores das tensões para cada passo de tempo estipulado.

As equações provenientes dos métodos de Jacobi e Gauss-Seidel para o cálculo do vetor de tensões nodais estão dispostas em (3.39) e (3.40), respectivamente:

$$V_i^{k+1} = \frac{1}{Y_{i,i}} \left(I_i^k - \sum_{j=1, j \neq i}^{n} Y_{i,j} V_j^k \right) \tag{3.39}$$

$$V_i^{k+1} = \frac{1}{Y_{i,i}} \left(I_i^k - \sum_{j=1}^{i-1} Y_{i,j} V_j^{k+1} - \sum_{j=i+1}^{n} Y_{i,j} V_j^k \right), \tag{3.40}$$

sendo V_j^k o vetor de tensões nodais atual, V_i^{k+1}, o vetor de tensões nodais a ser calculado, I_i^k, o vetor de correntes e $Y_{i,j}$, a matriz de admitâncias nodais do sistema.

Visto que os métodos de armazenamento já foram executados, a matriz Y presente nessas equações será substituída pelas matrizes que guardaram os elementos não nulos. Devido às características específicas de cada técnica, essa substituição é diferente entre elas, necessitando de estruturas auxiliares.

Capítulo 4

Método Proposto

O alcance do objetivo deste trabalho é resultado da associação de conteúdos e técnicas que propiciaram a resolução do sistema de equações lineares $I = YV$. Desta forma, este capítulo apresentará as métodos utilizados nos algoritmos expostos no Apêndice A; por meio destas, foi possível aliar os métodos de armazenamento aos iterativos descritos anteriormente, além de garantir isonomia no processo de aferição do tempo e memória despendidos.

A linguagem de programação Python abarcou todo o processo construtivo; por isso, os métodos apresentados se utilizam de suas bibliotecas, como a *Numpy*, e estruturas para associar as modelagens às equações estabelecidas, concretizando o processo computacional.

4.1 OpenDSSDirect.py

Esta ferramenta foi utilizada com o intuito de garantir a isonomia na análise de tempo decorrido e memória utilizada, que são aferidas através de funções disponibilizadas pela linguagem de programação. O *OpenDSSDirect.py* é um pacote Python de plataforma cruzada que implementa uma interface de biblioteca "direta" para OpenDSS usando *dss_python* (Krishnamurthy, 2022).

Com sintaxe que mescla a linguagem utilizada pelos arquivos provenientes do OpenDSS (.dss) e comandos próprios em python, este pacote permite executar os cálculos referentes à rede elétrica e extrair os resultados desejados. Sua sintaxe em Python é simples, necessitando da adição da sua biblioteca (*opendssdirect*). Caso o usuário deseje extrair mais informações, pode adicionar mais bibliotecas.

Para seu funcionamento, faz-se necessário elencar os elementos da rede elétrica e suas características, escrevendo-os na linguagem própria dos arquivos .dss, como mostra a figura 4.1, precedidos pelo comando *dss.run_command* .

Figura 4.1: Exemplos de equipamentos e suas descrições

```
dss.run_command(
     New line.Linha_1 Bus1=10 Bus2=20 Length=130 units=km Phases=3 Earthmodel=Carson
     R0=0.5358 R1=0.2499 X0=2.3215 X1=0.4360 C0=0.0 C1=0.0)

dss.run_command(
    New transformer.Trafo_4 Phases=3 Windings=2 Buses=(70 80) Conns=(delta wye) kvs=(69 13.8) kvas=(25000 25000))

dss.run_command(
    New load.Carga_2 Bus1=80 Phases=3  kv=13.8 kw=10000 pf=0.84 Model=2 Conn=wye)
```

Fonte: Autoria própria. (Extraído do algoritmo A.0.1)

Além destes atributos, outros podem ser adicionados. Este detalhamento pode ser encontrado no manual do *software* (Dugan, 2020) e nas notas técnicas (Rocha and Radatz, 2017) e (Rocha and Radatz, 2018a).

Após a descrição dos elementos da rede, através do comando *dss.Circuit.SystemY()*, constrói-se a matriz admitância de barra. Por fim, com o comando *dss.Solution.Solve()*, soluciona-se o sistema de equações e, para obter o vetor de tensões nodais, utiliza-se o comando *dss.Circuit.AllBusVMag()*. Para ilustrar este roteiro, a Figura 4.2 exibe um dos algoritmos utilizados. Nesta, as informações requisitadas foram as tensões nodais.

Figura 4.2: Fragmento do Algoritmo utilizando *opendssdirect*

```
Y=dss.Circuit.SystemY()

dss.Solution.Solve()

Voltage = dss.Circuit.AllBusVMag()

print(Voltage)
```

Fonte: Autoria própria. (Extraído do algoritmo A.0.1)

4.2 Integração dos Métodos de Armazenamento ao Método de Jacobi

A associação dos métodos de armazenamento ao método iterativo de Jacobi exigiu abordagens diferentes, devido às particularidades de cada uma das formas de guardar os elementos diferentes de zero da matriz de admitância e seus respectivos índices. Estas associações estão presentes em de Oliveira Coelho (2014) e de Oliveira (2019), porém, não contemplam todos os métodos presentes neste texto.

Os algoritmos construídos para esta associação introduzem os valores iniciais necessários para a resolução do sistema de equações já calculados. Desta forma,

antes das estruturas para armazenamento, são importados os vetores de tensão e correntes injetadas e a matriz admitâncias de barra.

Seguindo a equação (3.39), as operações envolveram as matrizes que armazenaram os termos não nulos - AN, para os métodos CSR e CSC, AA, para o CSV e VAL, para o DFA2 e Skyline. As demais matrizes serviram como balizadores para os cálculos executados; como guardam índices ou número de elementos (por linha ou coluna), elas forneceram os números para limitar a atuação dos estruturas de repetição construídas no algoritmo, respeitando a multiplicação matricial.

Figura 4.3: Parte do algoritmo CSR - Jacobi

```
cont1=0
while Iter<=IterMax:
    Iter=Iter+1
    V = np.zeros((n,1),dtype=complex)
    for k in range(n):
        d=int(IA[0,k+1]-IA[0,k])
        soma=0
        for k1 in range(d):
            d1=int(JA[0,cont1])
            if k != d1:
                soma=soma+(AN[0,cont1]*X[d1,0])
                cont1=cont1+1
            else:
                cont1=cont1+1
                continue
```

Fonte: Autoria própria. (Extraído do algoritmo A.1.1)

Na Figura 4.3, referente ao método CSR, as linhas 6 e 9 acionam as matrizes IA e JA para encaminhar a multiplicação matricial que envolve AN. A partir de IA, coleta-se o número de elementos não nulos de cada linha. Com este número, define-se a quantidade de repetições da estrutura relativa à multiplicação; isto é, será considerada apenas o total de termos da linha. Em JA foram depositados os índices relativos às colunas dos termos de AN; com estes números, se assegura a operação entre as admitâncias e as tensões correspondentes.

Para ilustrar o que foi descrito no parágrafo anterior, seguem as matrizes (3.23), (3.24) e (3.25), provenientes da aplicação do método CSR à matriz (3.22):

$$AN = \begin{bmatrix} Y_1 & Y_2 & Y_3 & Y_4 & Y_5 & Y_6 & Y_7 & Y_8 & Y_9 & Y_{10} \end{bmatrix}$$

$$JA = \begin{bmatrix} 1 & 2 & 2 & 3 & 1 & 3 & 2 & 4 & 5 & 6 \end{bmatrix}$$

$$IA = \begin{bmatrix} 0 & 2 & 4 & 6 & 8 & 9 & 10 \end{bmatrix}$$

Considerando a Figura 4.3, temos que, para as matrizes acima, $n = 6$ e $cont1 = 10$; visto que $cont1$ é um contador, este valor será alcançado ao final da repetição proposta na linha 5. Esses valores correspondem, respectivamente, ao número de linhas da matriz e o total de termos diferentes de zero.

Dadas as matrizes acima, para este exemplo, $cont1$ e os intervalos propostos terão valor inicial igual a 1. Desta forma, temos que, para $k = 1$,

$$d = IA[1, 2] - IA[1, 1] = 2. \tag{4.1}$$

Logo, $k1$ percorrerá o intervalo $[1, 2]$. Assim, temos que, para $k \neq d1$ e $d1 = JA[1, 1] = 1$,

$$soma = soma + (AN[1, 1] * X[1, 1]). \tag{4.2}$$

Visto que $k = d1 = 1$, a estrutura condicional não realiza a multiplicação prevista em (4.2), fazendo apenas a soma em $cont1$, conforme a penúltima linha da Figura 4.3. Como o intervalo percorrido por $k1$ não chegou ao fim, o próximo passo a ser realizado será, para $d1 = JA[1, 2] = 2$,

$$soma = soma + (AN[1, 2] * X[2, 1]). \tag{4.3}$$

Este processo finda quando $k = 6$, que corresponde ao número de linhas. Após, o valor obtido para a soma é remetido para o cálculo previsto na equação (3.39).

Para os demais métodos, respeitando as estruturas de suas matrizes, segue-se o mesmo procedimento; porém, em decorrência das particularidades, há acréscimo de estruturas de repetição ou de estruturas auxiliares, como o vetor "T3", presente na décima linha da Figura 4.3, que guarda os termos da diagonal da matriz de admitância. Apesar de representarem mais linhas de código, estas estruturas contribuem para a redução do uso de funções de repetição, que exigem mais tempo de processamento.

Além disso, as correntes de compensação são calculadas a cada passo de iteração para as cargas presentes na rede. Elas são obtidas a partir da seguinte equação (de Freitas (2015)):

$$\begin{bmatrix} I_{C1} \\ I_{C2} \\ I_{C3} \end{bmatrix} = \begin{bmatrix} I_{i1} \\ I_{i2} \\ I_{i3} \end{bmatrix} - \begin{bmatrix} Y_1 & Y_2 & Y_3 \\ Y_4 & Y_5 & Y_6 \\ Y_7 & Y_8 & Y_9 \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix}, \tag{4.4}$$

sendo

$$\begin{bmatrix} I_{i1} \\ I_{i2} \\ I_{i3} \end{bmatrix} = - \begin{bmatrix} \frac{S_1^*}{3V_1^*} \\ \frac{S_2^*}{3V_2^*} \\ \frac{S_3^*}{3V_3^*} \end{bmatrix} \tag{4.5}$$

Na equação (4.4), temos que o cálculo das correntes de compensação envolve os vetores de correntes injetadas e tensão nodal da carga, e a matriz de admitância da carga. Estas correntes injetadas são obtidas através da equação (4.5), por meio da potência e da tensão nodal. Após obter os valores, estas correntes de compensação são adicionadas ao vetor I.

Subsequente a esses processos, o algoritmo verifica se os valores do vetor de tensões nodais obedecem à tolerância estabelecida. Caso isso não ocorra, todo o processo se repete até que se tenha êxito nessa comparação ou se atinja o limite de iterações.

4.3 Integração dos Métodos de Armazenamento ao Método de Gauss-Seidel

Construída após a integração descrita na seção anterior, a associação do método de Gauss-Seidel aos métodos de armazenamento possui a mesma estrutura apresentada, com exceção das condições próprias da técnica, presentes na equação 3.40.

Figura 4.4: Parte do algoritmo CSR - Gauss-Seidel

```
cont1=0
while Iter<IterMax:
    Iter=Iter+1
    for k in range(n):
      d=int(IA[0,k+1]-IA[0,k])
      soma = 0
      soma1 = 1
      for k1 in range(d):
          d1=int(JA[0,cont1])
          if (k > d1):
              soma=soma+(AN[0,cont1]*V[d1,0])
          elif (k < d1):
              soma1=soma1+(AN[0,cont1]*X[d1,0])
          cont1=cont1+1
          if cont1==soma11:
              cont1=0
```

Fonte: Autoria própria. (Extraído do algoritmo A.1.2)

Esta equação possui dois somatórios. Logo, como consta na Figura 4.4, nas linhas 10 e 12, foi necessário adicionar mais uma estrutura condicional no laço que promove a multiplicação matricial, pois estas operações são feitas com intervalos

diferentes dos termos não nulos. Considerando o método CSR, as matrizes IA e JA possuem as mesmas funções: delimitar o valor para que as multiplicações se restrinjam à linha determinada pela estrutura de repetição e definir o índice correto para o vetor de tensões utilizado na multiplicação matricial, respectivamente.

Assim como no método de Jacobi, o algoritmo verifica se os valores do vetor de tensões nodais obedecem à tolerância estabelecida, repetindo todo o processo até que se tenha êxito nessa comparação ou se atinja o limite de iterações.

Capítulo 5

Resultados e Discussões

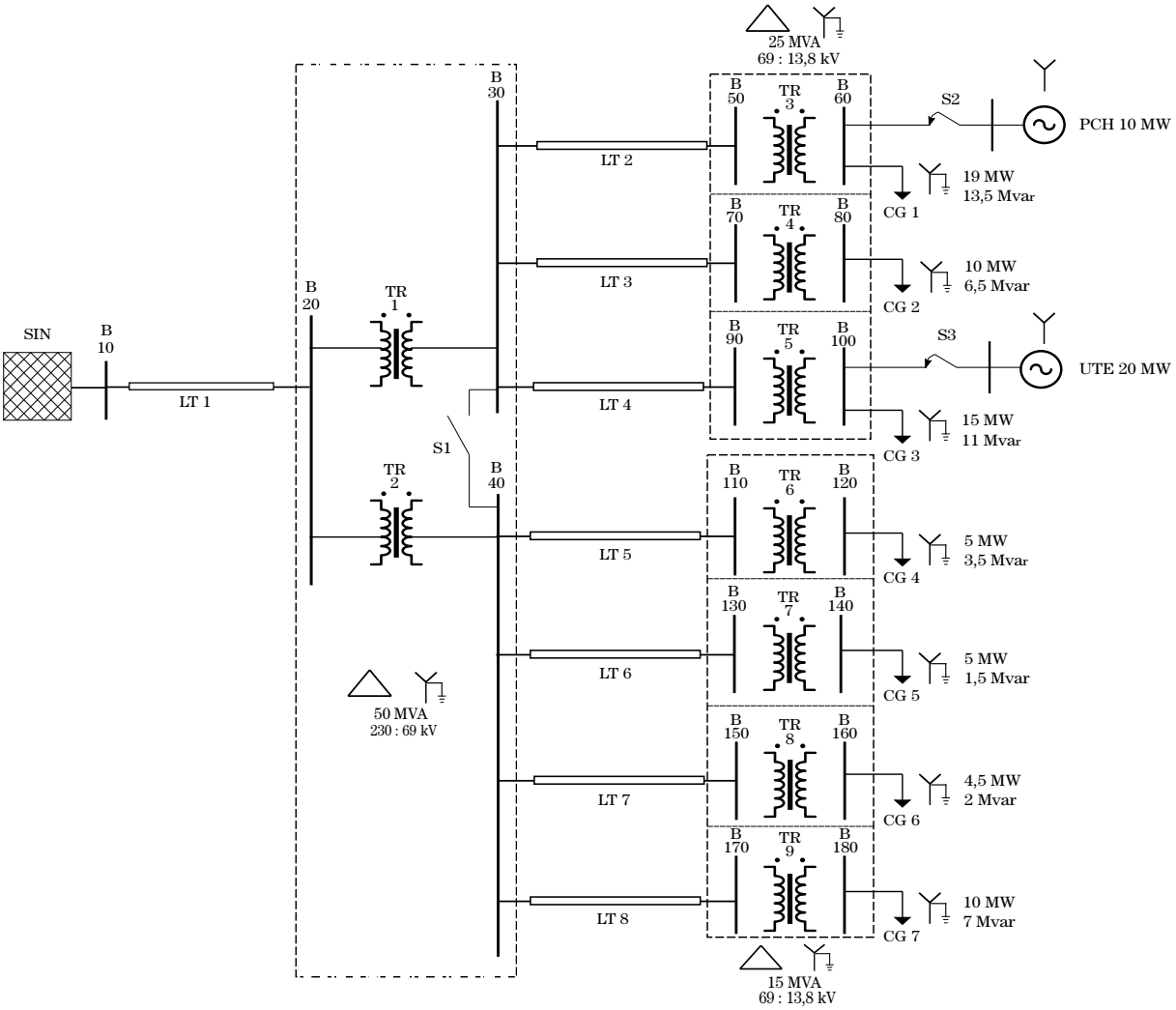
Este capítulo expõe os resultados obtidos e suas respectivas discussões. Como pré-requisitos, descreve-se a rede elétrica utilizada, sua matriz de admitância, que será alvo das técnicas de armazenamento, e os parâmetros definidos para efetuar os algoritmos.

A Seção 5.1 apresenta a rede elétrica escolhida, com a numeração e identificação de seus elementos. A Seção 5.2 descreve a matriz de admitância da rede, sua esparsidade e as características das matrizes obtidas pelos métodos de armazenamento. A Seção 5.3 exibe os parâmetros utilizados para a execução dos algoritmos construídos. As Seções 5.4 e 5.5 apresentam os resultados, discutindo-os e comparando-os. A Seção 5.6 exibe uma breve explanação sobre os resultados obtidos.

5.1 Descrição da Rede Elétrica

A rede elétrica escolhida é similar, com algumas modificações em níveis de carga e inserção de geração distribuída, à regional Açu II da Companhia Energética do Rio Grande do Norte (COSERN) e é formada por dezoito barras, oito linhas de transmissão, nove transformadores, sete cargas e duas unidades de geração distribuída. A seguir, será apresentado seu diagrama unifilar (Figura 5.1) e as Tabelas 5.1, 5.2, 5.3 e 5.4 contendo as numerações dos seus elementos.

Figura 5.1: Diagrama unifilar da rede elétrica



Fonte: Autoria Própria

As barras foram designadas por números que representam múltiplos de dez, sendo numeradas de 10 a 180. Desta forma, foram classificadas conforme a tabela a seguir:

Tabela 5.1: Barras.

Barra	Título	Tensão (kV)
10	B10	230
20	B20	230
30	B30	69
40	B40	69
50	B50	69
60	B60	13,8
70	B70	69
80	B80	13,8
90	B90	69
100	B100	13,8
110	B110	69
120	B120	13,8
130	B130	69
140	B140	13,8
150	B150	69
160	B160	13,8
170	B170	69
180	B180	13,8

As linhas de transmissão foram numeradas de um a oito. Elas podem ser divididas em três grupos; essa classificação é pautada nos cabos utilizados e na geometria. A linha 1 constitui um grupo, pois possui cabo e geometria diferente das demais. As linhas 2, 3 e 4 constituem outro grupo e, por fim, as linhas 5, 6, 7 e 8 representam o último grupo.

Tabela 5.2: Linhas de Transmissão.

LT	Início	Final	Tensão (kV)
1	Barra 10	Barra 20	230
2	Barra 30	Barra 50	69
3	Barra 30	Barra 70	69
4	Barra 30	Barra 90	69
5	Barra 40	Barra 110	69
6	Barra 40	Barra 130	69
7	Barra 40	Barra 150	69
8	Barra 40	Barra 170	69

Os transformadores foram numerados de um a nove. Possuem o mesmo tipo de ligação: delta - estrela aterrado.

Tabela 5.3: Transformadores.

TR	Entrada	Saída	Tensões (kV)
1	Barra 20	Barra 30	230:69
2	Barra 20	Barra 40	230:69
3	Barra 50	Barra 60	69:13,8
4	Barra 70	Barra 80	69:13,8
5	Barra 90	Barra 100	69:13,8
6	Barra 110	Barra 120	69:13,8
7	Barra 130	Barra 140	69:13,8
8	Barra 150	Barra 160	69:13,8
9	Barra 170	Barra 180	69:13,8

As cargas foram numeradas de um a sete e portam o mesmo tipo de ligação: estrela aterrado.

Tabela 5.4: Cargas.

Carga	Ligação	Potência (MW)	Potência (Mvar)
1	Barra 60	19	13,5
2	Barra 80	10	6,5
3	Barra 100	15	11
4	Barra 120	5	3,5
5	Barra 140	5	1,5
6	Barra 160	4,5	2
7	Barra 180	10	7

As unidades de geração distribuída possuem ligação em estrela e são conectadas à rede por meio de duas chaves seccionadoras: S2 e S3. A unidade geradora 1 é uma Pequena Central Hidrelétrica (PCH) e tem potência nominal de 10 MW. A unidade geradora 2 é uma Usina Termelétrica (UTE) e tem potência nominal de 20 MW.

5.2 Matriz Admitância de Barra da Rede e Esparsidade

A matriz admitância de barra ilustrada na Figura 5.1 foi montada de acordo com o que foi escrito na Seção 3.1. Ela possui dimensão $54\,x\,54$, ou seja, são 2916

termos. Desses, 468 são diferentes de zero e 2448 são nulos. Entre as 54 linhas que compõem a matriz, considerando os valores diferentes de zero, existem seis linhas com 15 termos, seis com 12, dezoito com 9 e vinte e quatro com 6 elementos.

Pela grande quantidade de elementos nulos, a matriz admitância de barra dessa rede elétrica é considerada esparsa. Considerando a definição de índice de esparsidade, presente em Maliska (2017), define-se este valor como o quociente entre o número de termos nulos e o número total de termos da matriz.

$$IE = \frac{Nº\ de\ Elementos\ Nulos}{Nº\ Total\ de\ Elementos} \times 100\% \tag{5.1}$$

Na rede tratada nesse trabalho, o índice de esparsidade é igual a 84%. Por esse motivo, efetuar os cálculos necessários com toda a matriz ocasionaria em uso desnecessário de memória, maior número de operações e mais tempo de processamento.

O decorrer desta seção explicará as matrizes resultantes da aplicação dos métodos de armazenamento à matriz de admitância descrita anteriormente. Para o método CSR, AN, JA e IA possuem as seguintes características:

- A matriz admitância de barra possui 468 elementos diferentes de zero. Desta forma, AN possui 468 colunas. Esses elementos aparecem na distribuição em que se encontram nas linhas, que foram percorridas na ordem crescente;
- JA, por possuir os índices referentes às colunas em que se encontram os elementos de AN, tem o mesmo tamanho desta. Os valores presentes em JA correspondem a mesma ordem de distribuição de AN;
- IA têm tamanho diferente das demais. Ela possui 55 colunas e guarda a quantidade de elementos não nulos por linha, percorridas como em AN, da linha 1 à 54. Seu primeiro termo é o número 0 e o último corresponde ao total de termos não nulos, 468.

Para o método CSC, essas matrizes apresentam algumas diferenças, visto que, para esta técnica, percorre-se a matriz admitância por colunas.

Os atributos a seguir advém das matrizes geradas pelo método CSV:

- AA possui 468 termos, que aparecem na distribuição em que se encontram nas linhas, percorridas na ordem crescente;
- IA possui o mesmo tamanho de AA e respeita a regra da contagem descrita na Subseção 3.3.3. O maior valor apresentado em IA é 49.

Utilizando a técnica Skyline, as matrizes resultantes possuem a composição abaixo:

- Por conter os termos nulos dispostos entre os elementos diferentes de zero, VAL detém 990 termos, que estão aprsentados na forma que aparecem nas linhas;
- Fstcol possui 55 termos. O maior valor que armazena, 48, refere-se à última barra da rede (Barra 180);
- Rowptr tem 55 elementos. Seu primeiro termo é igual a zero e seu último termo é igual a 990. Ressalta-se que seus valores são referentes à quantidade de elementos armazenados por linha; logo, os termos nulos presentes entre os diferentes de zero também são contabilizados.

Por fim, descrevem-se os resultados do emprego do método DFA2:

- Por ser uma compactação da matriz admitância de barra, VAL possui 54 linhas e 9 colunas. Como existem 468 termos não nulos, as últimas duas linhas serão preenchidas com zeros;
- ROWID possui 54 colunas, contendo os valores de zero a 53;
- INDEX e OFFSET também possuem 54 colunas e contém os índices, em VAL, dos primeiros elementos não nulos de cada linha da matriz admitância;
- #NZ guarda a quantidade de elementos não nulos de cada linha da matriz admitância de barra. Possui tamanho 54 e o maior valor que apresenta é 18; o menor, 6.

5.3 Parâmetros Utilizados

A execução dos códigos que obtém os valores das tensõs nodais da rede elétrica através da associação entre os métodos de armazenamento e iterativos foi feita com o emprego de determinadas ferramentas, que serão descritas a seguir. Além disso, a fim de facilitar às menções referentes às técnicas aplicadas, estas encontram-se enumeradas ao final desta seção.

As aferições do tempo de execução e memória despendida foram feitas por meio de funções próprias da linguagem de programação Python. A função *time.process_time()*, pertencente a biblioteca *time* (Foundation (2022)), foi utilizada para medir o tempo de processamento em segundos, e *psutil.virtual_memory()*, originária da biblioteca *psutil* (Rodola (2020)), para medição do espaço, em bytes, utilizado da memória.

Como consequência da utilização dos métodos iterativos, foi necessário adotar alguns critérios para o devido funcionamento. Além da condição inicial, foram estabelecidas a tolerância de 10^{-1} e o número máximo de 50 iterações para todos os

métodos. A escolha desta tolerância advém do fato desse estudo não estipular uma tensão de base (Rocha and Radatz (2018b)).

Todos os valores que serão apresentados nas próximas seções foram obtidos através do uso de um notebook com as seguintes especificações:

- Modelo: Acer Aspire A515-54G;
- Sistema Operacional: Windows 10, 64 bits;
- Memória RAM DDR4 8 GB;
- Processador Intel(R) Core(TM) i5-10210U 1.60GHz, memória cache de 6 MB, com 4 núcleos físicos e 8 threads;
- Armazenamento de 256 GB, SSD.

Por último, foram utilizados onze métodos para a realização das análises:

- OpenDSSDirect.py (DSS);
- Método CSR-JAC: CSR com Método de Jacobi;
- Método CSR-GS: CSR com Método de Gauss-Seidel;
- Método CSC-JAC: CSC com Método de Jacobi;
- Método CSC-GS: CSC com Método de Gauss-Seidel;
- Método CSV-JAC: CSV com Método de Jacobi;
- Método CSV-GS: CSV com Método de Gauss-Seidel;
- Método SKY-JAC: Skyline com Método de Jacobi;
- Método SKY-GS: Skyline com Método de Gauss-Seidel;
- Método DFA2-JAC: DFA2 com Método de Jacobi;
- Método DFA2-GS: DFA2 com Método de Gauss-Seidel.

Para estes arranjos, serão consideradas duas composições da rede: numa delas, as unidades de geração distribuída estarão conectadas à rede. A outra composição não contará com essa conexão.

5.4 Resultados para o Sistema Equilibrado

Nesta seção, serão apresentados os valores decorrentes das medições de tempo e memória despendidas para cada método listado, aferidos durante a execução dos algoritmos para a configuração da rede elétrica equilibrada.

Para cada método descrito na seção anterior, foram realizadas 60 execuções. De cada uma destas, extraiu-se o tempo de processamento e o uso da memória. A partir deste conjunto de dados, a média, variância e o desvio padrão foram calculados e serão os objetos de comparação deste trabalho.

5.4.1 Sem a inserção das Unidades de Geração Distribuída

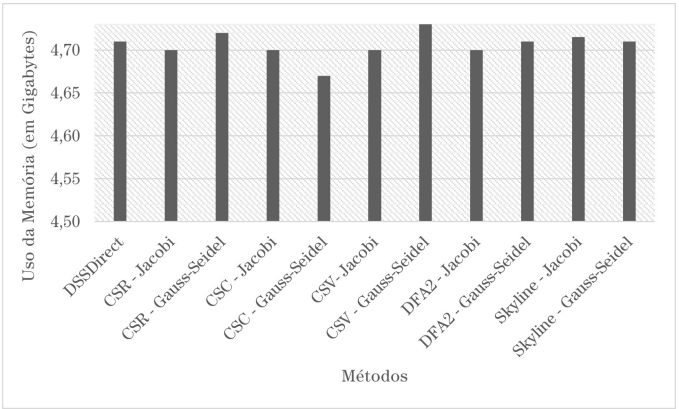
Os valores dispostos na Tabela 5.5, provenientes da análise da rede elétrica, mostram que o Método CSC-GS apresentou os menores valores para média, variância e desvio-padrão.

Tabela 5.5: Medidas - Tempos Aferidos - Sistema Equilibrado (Sem GD)

Medidas - Tempos Aferidos (em segundos)											
Medidas	Métodos										
	DSS	CSR-JAC	CSR-GS	CSC-JAC	CSC-GS	CSV-JAC	CSV-GS	DFA2-JAC	DFA2-GS	SKY-JAC	SKY-GS
Média	0,0653646	0,0460938	0,0366381	0,0541667	0,0352342	0,0519777	0,0426126	0,0557292	0,0429688	0,0513021	0,0432292
Variância	0,0014770	0,0006301	0,0004460	0,0011513	0,0003780	0,0017271	0,0007345	0,0006703	0,0005992	0,0007088	0,0007273
Desvio Padrão	0,0387559	0,0253134	0,0212960	0,0342165	0,0196074	0,0419088	0,0273308	0,0261087	0,0246844	0,0268471	0,0271955

Observando os demais valores exibidos, constata-se que a média do Método CSC-GS corresponde a, aproximadamente, 54% do número obtido pela execução do OpenDSS via Python. O número reduzido de iterações para obter a solução do sistema é o principal diferencial para este desempenho. Além disso, como apresentou menor variância, admite-se que seu processo computacional apresenta determinada regularidade, o que influe mais confiança em sua técnica.

Figura 5.2: Médias do Uso da Memória por Método - Sistema Equilibrado (Sem GD)



Fonte: Autoria própria

A análise da Figura 5.2 mostra que o *DSSDirect* possui média semelhante aos demais métodos. Com relação aos dez métodos propostos neste trabalho, observa-se que as associações ao método de Jacobi tem pequena vantagem com relação as demais, com exceção dos métodos que utilizam a técnica CSC.

5.4.2 Com a inserção das Unidades de Geração Distribuída

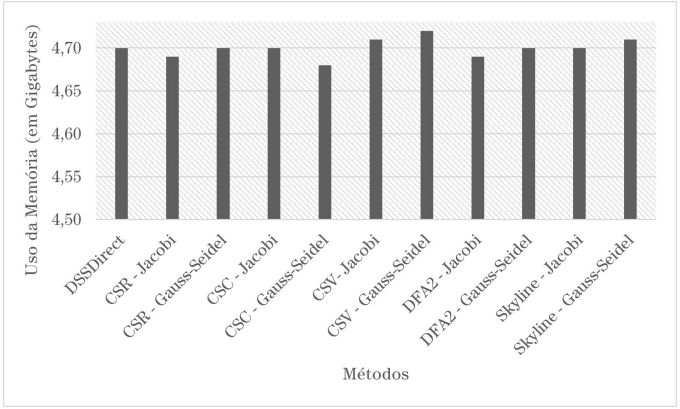
As unidades de geração distribuída 1 e 2, definidas na Seção 5.1, são conectadas à rede nesta configuração através das Barras 60 e 100, respectivamente. Com esta inserção, as médias, variâncias e desvios-padrão do conjunto de dados encontram-se na Tabela 5.6.

Tabela 5.6: Medidas - Tempos de Processamento - Sistema Equilibrado (Com GD)

Medidas - Tempos de Processamento (em segundos)											
Medidas	Métodos										
	DSS	CSR-JAC	CSR-GS	CSC-JAC	CSC-GS	CSV-JAC	CSV-GS	DFA2-JAC	DFA2-GS	SKY-JAC	SKY-GS
Média	0,0330729	0,0484375	0,0750372	0,0447917	0,0764449	0,0492188	0,0760417	0,0531250	0,0833333	0,0450521	0,0742188
Variância	0,0001879	0,0011776	0,0002819	0,0007362	0,0003209	0,0011623	0,0002886	0,0012386	0,0002821	0,0009366	0,0003225
Desvio Padrão	0,0138241	0,0346054	0,0169305	0,0273624	0,0180652	0,0343804	0,0171324	0,0354908	0,0169381	0,0308625	0,0181089

Baseado nos dados dispostos na tabela acima, constata-se que os métodos associados a Gauss-Seidel apresentam menor variância, porém, maiores médias de tempo de processamento. Representado como DSS, este método apresenta a menor média temporal.

Figura 5.3: Médias do Uso da Memória por Método - Sistema Equilibrado (Com GD)



Fonte: Autoria própria

Com relação ao uso da memória, a observação da Figura 5.3 mostra que todos os métodos construídos apresentam comportamento semelhante ao cenário da subseção anterior, com destaque para o método CSC-GS, que apresentou a melhor média.

5.5 Resultados para o Sistema Desequilibrado

Assim como na Seção 5.4, os valores presentes nas Tabelas 5.7 e 5.8 são consequência do conjunto de 60 execuções dos algoritmos contendo as condições definidas para o sistema desequilibrado. Para os valores referentes ao uso da memória, as médias obtidas estão presentes nas Figuras 5.4 e 5.5.

O desequilíbrio proposto para a rede elétrica advém das suas cargas, cujos valores das impedâncias (em *ohms*) seguem abaixo:

→ Carga 1:

- $Z_a = 21.08035942 + 12.50835072j$
- $Z_b = 20.09987759 + 14.02980086j$
- $Z_c = 20.59011850 + 13.29990267j$

→ Carga 2:

- $Z_a = 39.27960777 + 27.41733488j$
- $Z_b = 40.23764698 + 25.99095234j$
- $Z_c = 41.19568620 + 24.44408471j$

→ Carga 3:

- $Z_a = 24.26433925 + 18.83118027j$
- $Z_b = 25.49291339 + 17.13134194j$
- $Z_c = 24.87862632 + 18.01181452j$

→ Carga 4:

- $Z_a = 76.75914725 + 53.57821437j$
- $Z_b = 74.88697293 + 56.16522970j$
- $Z_c = 78.63132158 + 50.79081619j$

→ Carga 5:

- $Z_a = 107.25616149 + 21.77929250j$
- $Z_b = 102.87835898 + 37.33985823j$
- $Z_c = 105.06726024 + 30.64461757j$

→ Carga 6:

- $Z_a = 102.09541251 + 55.10526607j$
- $Z_b = 106.73611307 + 45.46939377j$
- $Z_c = 104.41576279 + 50.57086201j$

→ Carga 7:

- $Z_a = 38.37957361 + 26.78910717j$
- $Z_b = 37.44348645 + 28.08261484j$
- $Z_c = 36.50739929 + 29.28920844j$

Para obter estes valores, considerou-se que a carga trifásica desequilibrada é composta por três cargas monofásicas independentes. Para cada uma dessas partes, há um fator de potência nominal diferente das demais.

5.5.1 Cargas Desequilibradas sem a inserção das Unidades de Geração Distribuída

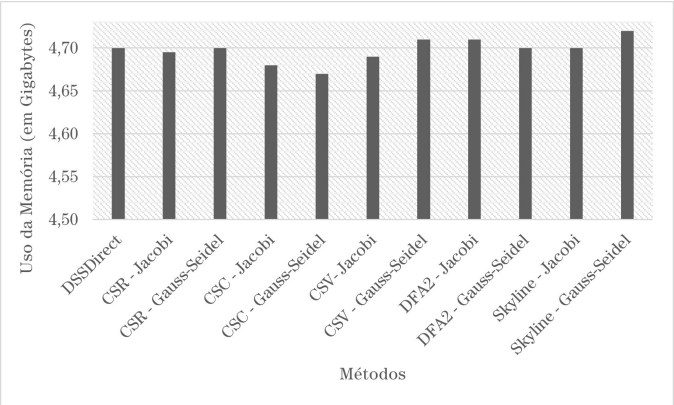
O desequilíbrio de todas as cargas presentes na rede elétrica analisada se apresentou como grande obstáculo as associações ao método de Jacobi. Isso indica que para situações de grande desequílibrio das redes elétricas, seu uso seja limitado.

Tabela 5.7: Medidas - Tempo de Processamento - Sistema Desequilibrado (Sem GD)

Medidas - Tempos de Processamento (em segundos)											
Medidas	Métodos										
	DSS	CSR-JAC	CSR-GS	CSC-JAC	CSC-GS	CSV-JAC	CSV-GS	DFA2-JAC	DFA2-GS	SKY-JAC	SKY-GS
Média	0,0794271	0,2203125	0,0717146	0,2565104	0,0751578	0,2684896	0,0804688	0,2369792	0,0781250	0,2773438	0,0768229
Variância	0,0021752	0,0086483	0,0004696	0,0065535	0,0003502	0,0038696	0,0004380	0,0055108	0,0006185	0,0056041	0,0004744
Desvio Padrão	0,0470329	0,0937809	0,0218526	0,0816367	0,0188718	0,0627308	0,0211058	0,0748612	0,0250793	0,0754919	0,0219640

As médias temporais dos métodos CSR-GS e CSC-GS e suas respectivas variâncias foram as menores para esta configuração. Com relação ao método CSC-GS, repete-se sua vantagem quando há a falta das unidades de geração distribuída.

Figura 5.4: Médias do Uso da Memória por Método - Sistema Desequilibrado (Sem GD)



Fonte: Autoria própria

A Figura 5.4 mostra que, com relação às associações construídas, as médias dos métodos CSC-JAC e CSC-GS apresentam os menores valores. Analisando junto às médias dispostas na Tabela 5.7, destacam-se os métodos CSR-GS e CSC-GS.

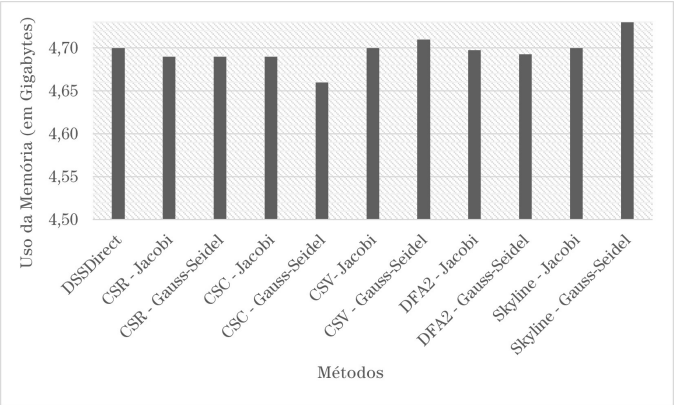
5.5.2 Cargas Desequilibradas com a inserção das Unidades de Geração Distribuída

Nesta configuração, a Tabela 5.8 apresenta cenário semelhante ao visto na subseção anterior: pequena diferença entre as médias referentes aos métodos que contém a iteração de Gauss-Seidel, além destes representarem cerca de 29% da média de tempo obtida pelas associações a Jacobi. A Figura 5.5 exibe uma semelhança com relação ao uso da memória: o método CSC-GS apresenta a melhor média.

Tabela 5.8: Medidas - Tempo de Processamento - Sistema Desequilibrado (Com GD)

Medidas - Tempo de Processamento (em segundos)											
Medidas	Métodos										
	DSS	CSR-JAC	CSR-GS	CSC-JAC	CSC-GS	CSV-JAC	CSV-GS	DFA2-JAC	DFA2-GS	SKY-JAC	SKY-GS
Média	0,0794271	0,2526042	0,0727335	0,2481771	0,0749990	0,2421875	0,0734375	0,2325521	0,0846354	0,1234375	0,0705729
Variância	0,0021752	0,0068943	0,0002840	0,0034024	0,0002939	0,0039510	0,0005965	0,0044197	0,0003849	0,0001440	0,0005329
Desvio Padrão	0,0470329	0,0837324	0,0169956	0,0588227	0,0172876	0,0633875	0,0246298	0,0670418	0,0197834	0,0121031	0,0232810

Figura 5.5: Médias do Uso da Memória por Método - Sistema Desequilibrado (Com GD)



Fonte: Autoria própria

5.6 Considerações sobre os Resultados

Os resultados apresentados na seção anterior demonstram que os métodos de armazenamento, com destaque ao CSR e CSC, associados ao método iterativo de Gauss-Seidel, possuem melhor desempenho com relação aos que contém o método de Jacobi, considerando o tempo. Por se tratar de um aperfeiçoamento da técnica apresentada por Jacobi, o método iterativo de Gauss-Seidel possibilita resolução do sistema de equações com menos iterações, o que demanda menor tempo de processamento. Além disso, os baixos valores de variância indicam que há uma regularidade em suas execuções, o que lhe transmite certa confiabilidade com relação ao tempo de processamento.

Com relação à memória despendida, observa-se que, para o sistema equilibrado, o comportamento para os cenários propostos é semelhante: há um pequeno acréscimo das associações ao método de Gauss-Seidel com relação as realizadas com o método de Jacobi, com exceção as associações ao método CSC. Isto se deve à necessidade de mais uma estrutura de repetição necessária. Quando há o desequilíbrio, as reuniões dos métodos CSR e CSR com Gauss-Seidel apresentam melhor desempenho quando se analisa tempo de processamento e memória utilizada.

Como última consideração, notou-se que, quando submetidos ao desequilíbrio de todas as cargas da rede, os códigos que contém o método de Jacobi demoram a convergir, o que indica que, para outras situações semelhantes, seu uso possa trazer limitações à resolução.

Capítulo 6

Conclusões

O presente trabalho descreveu os conceitos e a aplicação de métodos de armazenamento de matrizes em conjunto com os métodos iterativos de Jacobi e Gauss-Seidel com o intuito de colaborar para o melhoramento da simulação digital em tempo real. A partir da utilização conjunta dos métodos iterativos e de armazenamento, foi possível constatar a complexidade de operações com matrizes esparsas e a necessidade de utilizar técnicas para mitigar a esparsidade e otimizar os processos computacionais.

Durante o processo de estudos para construção deste texto, foi possível conhecer e aprender a implementar as técnicas necessárias para modelagem dos elementos que compõem os sistemas elétricos, construir a matriz admitância de barra, entender e utilizar os cinco métodos de armazenamento de matrizes descritos no texto, assim como suas associações aos métodos iterativos escolhidos. Além desse processo de aquisição e aplicação de conhecimento, foi possível aprender sobre o *software* OpenDSS, que foi utilizado como referência neste trabalho.

Os resultados obtidos mostraram que os métodos de armazenamento CSR e CSC são mais eficazes com relação aos dados guardados, pois fornecem informações mais relevantes para usos posteriores, como a associação tratada neste trabalho, sem utilizar regras de contagem ou reordenações, como CSV e DFA2, o que demanda mais esforço computacional.

A exceção da configuração do sistema equilibrado com a inserção das unidades de geração distribuída, os códigos que associaram os métodos de armazenamento ao método iterativo de Gauss-Seidel apresentaram melhores médias de tempo, além de menores variâncias, o que indica a regularidade das suas execuções. Um dos fatores que contribuíram para estas observações foi a necessidade de menos iterações para obter a solução do sistema de equações. Sobre as integrações, cabe destacar os métodos CSR e CSC que, por possuírem maior facilidade para adaptação às equações e estruturas, inerentes à linguagem de programação, necessárias para funcionamento dos métodos iterativos, são utilizados em *softwares* que lidam com matrizes esparsas.

A demanda por memória foi semelhante para os métodos construídos e a referência escolhida (*OpenDSSDirect.py*), com exceção à técnica que combinou os métodos CSC e Gauss-Seidel. O desempenho desta expressa potencial para usos futuros, com as devidas adequações e acréscimos necessários.

Por fim, observa-se que, para os processos da simulação em tempo real que tratam e utilizam matrizes esparsas, estes dados são importantes, pois a rapidez do processamento das informações e da execução dos cálculos necessários são essenciais para o funcionamento correto do simulador, que deve apresentar os resultados necessários, de forma a reproduzir as condições da rede real em estudo. Devido à linguagem de programação escolhida (Python), há a possibilidade de formular sub-rotinas e construir bibliotecas, o que otimiza a estrutura e funcionamento, com viabilidade de adequar a outras linguagens de programação.

6.1 Trabalhos Futuros

Nesta seção, listam-se os possíveis trabalhos futuros:

- Adição de métodos de armazenamento e iterativos ao processo de comparação;
- Análise de redes elétricas de maior porte;
- Criação de bibliotecas e sub-rotinas em Python referentes à associação entre métodos de armazenamento e iterativos;
- Adaptação das bibliotecas e sub-rotinas à outras linguagens de programação para possibilitar embarque dos algoritmos em diferentes dispositivos lógico programáveis.

6.2 Publicações

Os artigos a seguir foram construídos durante o desenvolvimento deste trabalho:

- "Comparação de Métodos de Armazenamento de Matrizes para o Melhoramento da Simulação em Tempo Real de Redes Elétricas". Foi apresentado no $XXIV$ Congresso Brasileiro de Automática (CBA), realizado na cidade de Fortaleza, Ceará, entre os dias 16 e 19 de outubro de 2022.
- "Analysis of Sparse Matrix Storage Methods for the Improvement of Real-time Simulation of Power Grids". Submetido ao 2023 IEEE Power & Energy Society (PES) General Meeting (GM), que será realizado entre os dias 16 e 20 de julho de 2023, em Orlando, Flórida, Estados Unidos da América.

Referências Bibliográficas

V. B. Andrade, U. C. P. Jr., C. E. Moreira., T. M. Soares., J. M. Tabora., M. E. de L. Tostes., U. H. Bezerra., B. S. Albuquerque., and L. D. S. Gouveia. Modelagem de um sistema de distribuição real desbalanceado e análise do impacto da geração distribuída utilizando o software opendss. *Anais do Simpósio Brasileiro de Sistemas Elétricos*, 1, 2020.

J. Arrillaga and N. R. Watson. *Computer Modelling of Electrical Power Systems*. John Wiley Sons, Chichester, 2 edition, 2001.

F. Branin. Computer methods of network analysis. *Proceedings of the IEEE*, 55: 1787–1801, 1967.

M. C. C. Cunha. *Métodos Numéricos*. Editora da Unicamp, Campinas, 2 edition, 2018.

P. R. R. de Freitas. Modelos avançados de análise de redes elétricas inteligentes utilizando o software opendss. Trabalho de conclusão de curso, Universidade de São Paulo, São Paulo, São Paulo, Brasil, 2015.

T. R. M. de Oliveira. Avaliação de formatos de armazenamento com compressão para resolução de sistemas de equações lineares esparsos. Trabalho de conclusão de curso, Universidade Tecnológica Federal do Paraná, Apucarana, Paraná, Brasil, 2019.

M. A. de Oliveira Coelho. Métodos iterativos para resolver sistemas de equações algébricas lineares em estruturas esparsas. Dissertação de mestrado, Universidade Federal Fluminense, Volta Redonda, Rio de Janeiro, Brasil, 2014.

M. D. Dugan, Roger C. *Reference Guide - The Open Distribution System Simulator (OpenDSS)*. Electric Power Research Institute (EPRI), Palo Alto, 2020.

L. C. Dutto, C. Y. Lepage, and W. G. Habashi. Effect of the storage format of sparse linear systems on parallel cfd computations. *Computer Methods in Applied Mechanics and Engineering*, 188:441–453, 2000.

A. Farzaneh, H. Kheiri, and M. Abbaspour. An efficient storage format for large sparse matrices. *Communications de la Faculté des Sciences de l'Université d'Ankara. Séries A1: Mathematics and Statistics*, 58, Jan. 2009.

C. F. A. B. d. Filho, Milton Brown do C. *Métodos Numéricos: fundamentos e implementação computacional.* GEN LTC, São Paulo, 1 edition, 2017.

P. S. Foundation. Time access and conversions, 2022. URL `https://docs.python.org/3/library/time.htm`. Último acesso 20 Abril 2022.

W. He. Review on storage and ordering for large sparse matrix. *Final Project, Math221*, 2003.

D. Izudin, R. A. Jabr, and H.-T. Neisius. Transformer modeling for three-phase distribution network analysis. *IEEE Transactions on Power Systems*, 30 (5):2604–2611, 2015.

L. Jiang, J. Tan, and Q. Tang. An efficient sparse matrix format for accelerating regular expression matching on field-programmable gate arrays. *Security and Communication Networks*, 8, Jan. 2015.

N. Kagan, C. de Oliveira, and E. Robba. *Introdução aos Sistemas de Distribuição de Energia Elétrica.* Editora Edgard Blucher, São Paulo, 2 edition, 2017.

W. H. Kersting. *Distribution System Modeling and Analysis.* CRC Press, New York, 1 edition, 2001.

D. Krishnamurthy. Opendssdirect.py, 2022. URL `https://pypi.org/project/OpenDSSDirect.py/`. Último acesso 13 Janeiro 2023.

D. Langr and P. Tvrdík. Evaluation criteria for sparse matrix storage formats. *IEEE Transactions on Parallel and Distributed Systems*, 27:428–440, 2016.

C. R. Maliska. *Transferência de Calor e Mecânica dos Fluidos Computacional.* LTC, Rio de Janeiro, 2 edition, 2017.

A. Monticelli and A. Garcia. *Introdução a Sistemas de Energia Elétrica.* Editora Unicamp, Campinas, 2 edition, 2011.

OPAL-RT. Hypersim, 2022a. URL `https://www.opal-rt.com/systems-hypersim/`. Último acesso 4 Novembro 2022.

OPAL-RT. emegasim, 2022b. URL `https://www.opal-rt.com/system-emegasim/`. Último acesso 4 Novembro 2022.

D. R. Reynolds, D. J. Gardner, C. S. Woodward, and C. J. Balos. Sunlinsol_-klu usage, 2020. URL `http://runge.math.smu.edu/arkode_dev/doc/guide/build/html/sunlinsol/SUNLinSol_KLU.html`. Último acesso 11 Dezembro 2021.

C. Rocha and P. Radatz. Nota técnica - elemento line do opendss. *Grupo de usuários do OpenDSS Brasil*, 2017.

C. Rocha and P. Radatz. Nota técnica - elemento load do opendss. *Grupo de usuários do OpenDSS Brasil*, 2018a.

C. Rocha and P. Radatz. Nota técnica - algoritmo de fluxo de potência do opendss. *Grupo de usuários do OpenDSS Brasil*, 2018b.

G. Rodola. psutil documentation, 2020. URL `https://psutil.readthedocs.io/en/latest/`. Último acesso 12 Dezembro 2021.

D. J. Rose and R. A. Willoughby. *Sparse Matrices and their Applications*. Springer, New York, 1 edition, 1972.

RTDS. Rtds® simulator overview, 2022. URL `https://www.rtds.com/wp-content/uploads/2020/11/RTDS-Simulator-Overview.pdf`. Último acesso 4 Novembro 2022.

R. Shahnaz, A. Usman, and I. R. Chughtai. Review of storage techniques for sparse matrices. *2005 Pakistan Section Multitopic Conference, INMIC*, pages 1–7, 2006.

P. Stathis, S. Vassiliadis, and S. Cotofana. A hierarchical sparse matrix storage format for vector processors. *In Proceedings of 17th International Parallel and Distributed Processing Symposium (IPDPS 2003)*, 2003.

W. D. Stevenson. *Elementos de Análise de Sistemas de Potência*. McGraw-Hill, São Paulo, 2 edition, 1986.

R. Tewarson. *Sparse Matrices*. Academic Press, London, 1 edition, 1973.

Apêndice A

Algoritmos Construídos

A.0.1 OpenDSSDirect.py

```
import opendssdirect as dss
import time
import psutil
import sys

sys.stdout=open("dssdipy1.txt","a")

dss.run_command(
    "New circuit.SIN bus1=10 basekv=398.371686 angle=0 phases=3 model=2")

dss.run_command(
    "New line.Linha_1 Bus1=10 Bus2=20 Length=130 units=km Phases=3
     Earthmodel=Carson R0=0.5358 R1=0.2499
     X0=2.3215 X1=0.4360 C0=0.0 C1=0.0")

dss.run_command(
    "New line.Linha_2 Bus1=30 Bus2=50 Length=15 units=km Phases=3
     Earthmodel=Carson R0=0.5572 R1=0.2713
     X0=2.7878 X1=0.3779 C0=0.0 C1=0.0")

dss.run_command(
    "New line.Linha_3 Bus1=30 Bus2=70 Length=13 units=km Phases=3
     Earthmodel=Carson R0=0.5572 R1=0.2713
     X0=2.7878 X1=0.3779 C0=0.0 C1=0.0")
```

```
dss.run_command(
    "New line.Linha_4 Bus1=30 Bus2=90 Length=17 units=km Phases=3
     Earthmodel=Carson R0=0.5572 R1=0.2713
     X0=2.7878 X1=0.3779 C0=0.0 C1=0.0")

dss.run_command(
    "New line.Linha_5 Bus1=40 Bus2=110 Length=8 units=km Phases=3
     Earthmodel=Carson R0=0.6279 R1=0.3420
     X0=2.7741 X1=0.3498 C0=0.0 C1=0.0")

dss.run_command(
    "New line.Linha_6 Bus1=40 Bus2=130 Length=10 units=km Phases=3
     Earthmodel=Carson R0=0.6279 R1=0.3420
     X0=2.7741 X1=0.3498 C0=0.0 C1=0.0")

dss.run_command(
    "New line.Linha_7 Bus1=40 Bus2=150 Length=7.5 units=km Phases=3
     Earthmodel=Carson R0=0.6279 R1=0.3420
     X0=2.7741 X1=0.3498 C0=0.0 C1=0.0")

dss.run_command(
    "New line.Linha_8 Bus1=40 Bus2=170 Length=11 units=km Phases=3
     Earthmodel=Carson R0=0.6279 R1=0.3420
     X0=2.7741 X1=0.3498 C0=0.0 C1=0.0")

dss.run_command(
    "New transformer.Trafo_1 Phases=3 Windings=2 Buses=(20 30)
     Conns=(delta wye) kvs=(230 69) kvas=(50000 50000)")

dss.run_command(
    "New transformer.Trafo_2 Phases=3 Windings=2 Buses=(20 40)
     Conns=(delta wye) kvs=(230 69) kvas=(50000 50000)")

dss.run_command(
    "New transformer.Trafo_3 Phases=3 Windings=2 Buses=(50 60)
     Conns=(delta wye) kvs=(69 13.8) kvas=(25000 25000)")

dss.run_command(
    "New transformer.Trafo_4 Phases=3 Windings=2 Buses=(70 80)
```

```
     Conns=(delta wye) kvs=(69 13.8) kvas=(25000 25000)")

dss.run_command(
    "New transformer.Trafo_5 Phases=3 Windings=2 Buses=(90 100)
     Conns=(delta wye) kvs=(69 13.8) kvas=(25000 25000)")

dss.run_command(
    "New transformer.Trafo_6 Phases=3 Windings=2 Buses=(110 120)
     Conns=(delta wye) kvs=(69 13.8) kvas=(15000 15000)")

dss.run_command(
    "New transformer.Trafo_7 Phases=3 Windings=2 Buses=(130 140)
     Conns=(delta wye) kvs=(69 13.8) kvas=(15000 15000)")

dss.run_command(
    "New transformer.Trafo_8 Phases=3 Windings=2 Buses=(150 160)
     Conns=(delta wye) kvs=(69 13.8) kvas=(15000 15000)")

dss.run_command(
    "New transformer.Trafo_9 Phases=3 Windings=2 Buses=(170 180)
     Conns=(delta wye) kvs=(69 13.8) kvas=(15000 15000)")

dss.run_command(
    "New load.Carga_1 Bus1=60 Phases=3 kv=13.8 kw=19000 pf=0.82 Model=2 Conn=wye")

dss.run_command(
    "New load.Carga_2 Bus1=80 Phases=3  kv=13.8 kw=10000 pf=0.84 Model=2 Conn=wye")

dss.run_command(
    "New load.Carga_3 Bus1=100 Phases=3 kv=13.8 kw=15000 pf=0.81 Model=2 Conn=wye")

dss.run_command(
    "New load.Carga_4 Bus1=120 Phases=3 kv=13.8 kw=5000 pf=0.82 Model=2 Conn=wye")

dss.run_command(
    "New load.Carga_5 Bus1=140 Phases=3 kv=13.8 kw=5000 pf=0.96 Model=2 Conn=wye")

dss.run_command(
    "New load.Carga_6 Bus1=160 Phases=3 kv=13.8 kw=4500 pf=0.92 Model=2 Conn=wye")
```

```
dss.run_command(
    "New load.Carga_7 Bus1=180 Phases=3 kv=13.8 kw=10000 pf=0.82 Model=2 Conn=wye")

dss.run_command("Set tolerance=0.1")

Y=dss.Circuit.SystemY()

inicio = time.process_time()

dss.Solution.Solve()

Voltage = dss.Circuit.AllBusVMag()

print(Voltage)

fim = time.process_time()
print()
print('Tempo de Processamento %f' % (fim - inicio))
print()
print(psutil.virtual_memory())

sys.stdout.flush()
sys.exit()
```

A.1 Método CSR

A.1.1 Integrado ao Método de Jacobi

```
import numpy as np
import time
import psutil
import sys

sys.stdout=open("1csrjac.txt","a")

inicio = time.process_time()
```

```
n=54

aux = np.loadtxt('inputys1.txt', dtype='complex')
aux = aux.reshape((1,n,n))

YS = np.zeros((n,n),dtype=complex)
for i in range(n):
    for j in range(n):
      YS[i,j] = aux[0][i][j]

T3 = np.zeros((n,1),dtype=complex)
for i in range(n):
    T3[i,0] = YS[i,i]

soma1=0
for i in range(n):
    for j in range(n):
        if YS[i,j] != 0:
            soma1 = soma1 + 1

AN = np.zeros((1,soma1),dtype=complex)
JA = np.zeros((1,soma1))
IA = np.zeros((1,n+1))
IA[0,0] = 0
h=0
z=0
for i in range(n):
    for j in range(n):
        if YS[i,j] != 0:
            AN[0,h] = YS[i,j]
            JA[0,h] = j
            h=h+1
            z=z+1
        IA[0,i+1] = z

aux1 = np.loadtxt('inputi1.txt', dtype='complex')
aux1 = aux1.reshape((1,n,1))
I = np.zeros((n,1),dtype=complex)
```

```
for i in range(n):
    for j in range(1):
        I[i,j] = aux1[0][i][j]


aux2 = np.loadtxt('inputx1.txt', dtype='complex')
aux2 = aux2.reshape((1,n,1))
V8 = np.zeros((n,1),dtype=complex)
for i in range(n):
    for j in range(1):
        V8[i,j] = aux2[0][i][j]

n1=3

aux2 = np.loadtxt('C1.txt', dtype='complex')
aux3 = aux2.reshape((1,n1,n1))
C1 = np.zeros((n1,n1),dtype=complex)
for i in range(n1):
    for j in range(n1):
        C1[i,j] = aux3[0][i][j]

S1=np.zeros((n1,1),dtype=complex)
S1[0,0]=complex(19000000,-13500000)
S1[1,0]=complex(2191342.961,23204482.67)
S1[2,0]=complex(-21191342.94,-9704482.688)

aux4 = np.loadtxt('C2.txt', dtype='complex')
aux5 = aux4.reshape((1,n1,n1))
C2 = np.zeros((n1,n1),dtype=complex)
for i in range(n1):
    for j in range(n1):
        C2[i,j] = aux5[0][i][j]

S2=np.zeros((n1,1),dtype=complex)
S2[0,0]=complex(10000000,-6500000)
S2[1,0]=complex(629165.115,11910254.04)
S2[2,0]=complex(-10629165.12,-5410254.047)

aux6 = np.loadtxt('C3.txt', dtype='complex')
```

```
aux7 = aux6.reshape((1,n1,n1))
C3 = np.zeros((n1,n1),dtype=complex)
for i in range(n1):
    for j in range(n1):
        C3[i,j] = aux7[0][i][j]

S3=np.zeros((n1,1),dtype=complex)
S3[0,0]=complex(15000000,-11000000)
S3[1,0]=complex(2026279.443,18490381.06)
S3[2,0]=complex(-17026279.44,-7490381.069)


aux8 = np.loadtxt('C4.txt', dtype='complex')
aux9 = aux8.reshape((1,n1,n1))
C4 = np.zeros((n1,n1),dtype=complex)
for i in range(n1):
    for j in range(n1):
        C4[i,j] = aux9[0][i][j]

S4=np.zeros((n1,1),dtype=complex)
S4[0,0]=complex(5000000,-3500000)
S4[1,0]=complex(531088.9134,6080127.019)
S4[2,0]=complex(-5531088.913,-2580127.019)


aux10 = np.loadtxt('C5.txt', dtype='complex')
aux11 = aux10.reshape((1,n1,n1))
C5 = np.zeros((n1,n1),dtype=complex)
for i in range(n1):
    for j in range(n1):
        C5[i,j] = aux11[0][i][j]

S5=np.zeros((n1,1),dtype=complex)
S5[0,0]=complex(5000000,-1500000)
S5[1,0]=complex(-1200961.897,5080127.018)
S5[2,0]=complex(-3799038.103,-3580127.021)

aux12 = np.loadtxt('C6.txt', dtype='complex')
aux13 = aux12.reshape((1,n1,n1))
```

```
C6 = np.zeros((n1,n1),dtype=complex)
for i in range(n1):
    for j in range(n1):
        C6[i,j] = aux13[0][i][j]

S6=np.zeros((n1,1),dtype=complex)
S6[0,0]=complex(4500000,-2000000)
S6[1,0]=complex(-517949.192,4897114.317)
S6[2,0]=complex(-4424633.127,-2161624.78)

aux14 = np.loadtxt('C7.txt', dtype='complex')
aux15 = aux14.reshape((1,n1,n1))
C7 = np.zeros((n1,n1),dtype=complex)
for i in range(n1):
    for j in range(n1):
        C7[i,j] = aux15[0][i][j]

S7=np.zeros((n1,1),dtype=complex)
S7[0,0]=complex(10000000,-7000000)
S7[1,0]=complex(1062177.834,12160254.04)
S7[2,0]=complex(-11062177.83,-5160254.039)


I1=np.zeros((n1,1),dtype=complex)
V1=np.zeros((n1,1),dtype=complex)
I2=np.zeros((n1,1),dtype=complex)
V2=np.zeros((n1,1),dtype=complex)
I3=np.zeros((n1,1),dtype=complex)
V3=np.zeros((n1,1),dtype=complex)
I4=np.zeros((n1,1),dtype=complex)
V4=np.zeros((n1,1),dtype=complex)
I5=np.zeros((n1,1),dtype=complex)
V5=np.zeros((n1,1),dtype=complex)
I6=np.zeros((n1,1),dtype=complex)
V6=np.zeros((n1,1),dtype=complex)
I7=np.zeros((n1,1),dtype=complex)
V7=np.zeros((n1,1),dtype=complex)

for i in range(15,18):
```

```
    I1[i-15,0]=S1[i-15,0]/(3*V8[i,0])
    V1[i-15,0]=V8[i,0]
H1=np.dot(C1,V1)
for j in range(15,18):
    I[i,0]=I1[i-15,0]-H1[i-15,0]

for i in range(21,24):
    I2[i-21,0]=S2[i-21,0]/(3*V8[i,0])
    V2[i-21,0]=V8[i,0]
H2=np.dot(C2,V2)
for j in range(21,24):
    I[i,0]=I2[i-21,0]-H2[i-21,0]

for i in range(27,29):
    I3[i-27,0]=S3[i-27,0]/(3*V8[i,0])
    V3[i-27,0]=V8[i,0]
H3=np.dot(C3,V3)
for j in range(27,29):
    I[i,0]=I3[i-27,0]-H3[i-27,0]

for i in range(33,36):
    I4[i-33,0]=S4[i-33,0]/(3*V8[i,0])
    V4[i-33,0]=V8[i,0]
H4=np.dot(C4,V4)
for j in range(33,36):
    I[i,0]=I4[i-33,0]-H4[i-33,0]

for i in range(39,42):
    I5[i-39,0]=S5[i-39,0]/(3*V8[i,0])
    V5[i-39,0]=V8[i,0]
H5=np.dot(C5,V5)
for j in range(39,42):
    I[i,0]=I5[i-39,0]-H5[i-39,0]

for i in range(45,48):
    I6[i-45,0]=S6[i-45,0]/(3*V8[i,0])
    V6[i-45,0]=V8[i,0]
H6=np.dot(C6,V6)
for j in range(45,48):
```

```
    I[i,0]=I6[i-45,0]-H6[i-45,0]

for i in range(51,54):
    I7[i-51,0]=S7[i-51,0]/(3*V8[i,0])
    V7[i-51,0]=V8[i,0]
H7=np.dot(C7,V7)
for j in range(51,54):
    I[i,0]=I7[i-51,0]-H7[i-51,0]

t = 1
T=10**-t
IterMax = 50

X = np.zeros((n,1),dtype=complex)
T1 = np.zeros((n,1),dtype=complex)
T2 = np.zeros((n,1),dtype=complex)

print('Jacobi-CSR')
print()
Iter=0
X=V8

cont1=0
while Iter<=IterMax:
    Iter=Iter+1
    V = np.zeros((n,1),dtype=complex)
    for k in range(n):
        d=int(IA[0,k+1]-IA[0,k])
        soma=0
        for k1 in range(d):
            d1=int(JA[0,cont1])
            if AN[0,cont1] != T3[k,0]:
                soma=soma+(AN[0,cont1]*X[d1,0])
                cont1=cont1+1
            else:
                cont1=cont1+1
                continue
        V[k,0]=(I[k,0]-soma)/T3[k,0]
        t1 = (V[k,0]-X[k,0])
```

```
    T1[k,0] = t1
    t2 = (V[k,0])
    T2[k,0] = t2
print()
print('Iteração =',Iter)
print()
print(V)
print()

t3 = max(abs(T1))
t4 = max(abs(T2))
t5=t3/t4
if t5<T:
    break
else:
    X=V
    for i in range(15,18):
        I1[i-15,0]=S1[i-15,0]/(3*X[i,0])
        V1[i-15,0]=X[i,0]
    H1=np.dot(C1,V1)
    for j in range(15,18):
        I[i,0]=I1[i-15,0]-H1[i-15,0]


    for i in range(21,24):
        I2[i-21,0]=S2[i-21,0]/(3*X[i,0])
        V2[i-21,0]=X[i,0]
    H2=np.dot(C2,V2)
    for j in range(21,24):
        I[i,0]=I2[i-21,0]-H2[i-21,0]


    for i in range(27,29):
        I3[i-27,0]=S3[i-27,0]/(3*X[i,0])
        V3[i-27,0]=X[i,0]
    H3=np.dot(C3,V3)
    for j in range(27,29):
        I[i,0]=I3[i-27,0]-H3[i-27,0]
```

```
            for i in range(33,36):
                I4[i-33,0]=S4[i-33,0]/(3*X[i,0])
                V4[i-33,0]=X[i,0]
            H4=np.dot(C4,V4)
            for j in range(33,36):
                I[i,0]=I4[i-33,0]-H4[i-33,0]


            for i in range(39,42):
                I5[i-39,0]=S5[i-39,0]/(3*X[i,0])
                V5[i-39,0]=X[i,0]
            H5=np.dot(C5,V5)
            for j in range(39,42):
                I[i,0]=I5[i-39,0]-H5[i-39,0]


            for i in range(45,48):
                I6[i-45,0]=S6[i-45,0]/(3*X[i,0])
                V6[i-45,0]=X[i,0]
            H6=np.dot(C6,V6)
            for j in range(45,48):
                I[i,0]=I6[i-45,0]-H6[i-45,0]


            for i in range(51,54):
                I7[i-51,0]=S7[i-51,0]/(3*X[i,0])
                V7[i-51,0]=X[i,0]
            H7=np.dot(C7,V7)
            for j in range(51,54):
                I[i,0]=I7[i-51,0]-H7[i-51,0]

        if cont1==soma1:
            cont1=0

print()
print('Vetor V')
print()
print(V)
```

```
print()
print('Iterações =',Iter)

fim = time.process_time()
print()
print('Tempo de Processamento %f' % (fim - inicio))
print()
print(psutil.virtual_memory())

sys.stdout.flush()
sys.exit()
```

A.1.2 Integrado ao Método de Gauss-Seidel

```
import numpy as np
import time
import psuti
import sys

sys.stdout=open("1csrgs.txt","a")

inicio = time.perf_counter()

n=54

aux = np.loadtxt('inputys1.txt', dtype='complex')
aux = aux.reshape((1,n,n))

YS = np.zeros((n,n),dtype=complex)

for i in range(n):
    for j in range(n):
        YS[i,j] = aux[0][i][j]

T3 = np.zeros((n,1),dtype=complex)
for i in range(n):
    T3[i,0] = YS[i,i]
```

```
soma11=0
for i in range(n):
    for j in range(n):
        if YS[i,j] != 0:
            soma11 = soma11 + 1

AN = np.zeros((1,soma11),dtype=complex)
JA = np.zeros((1,soma11))
IA = np.zeros((1,n+1))
IA[0,0] = 0
h=0
z=0
for i in range(n):
    for j in range(n):
        if YS[i,j] != 0:
            AN[0,h] = YS[i,j]
            JA[0,h] = j
            h=h+1
            z=z+1
        IA[0,i+1] = z

aux1 = np.loadtxt('inputi1.txt', dtype='complex')
aux1 = aux1.reshape((1,n,1))

I = np.zeros((n,1),dtype=complex)

for i in range(n):
    for j in range(1):
        I[i,j] = aux1[0][i][j]

aux2 = np.loadtxt('inputx1.txt', dtype='complex')
aux2 = aux2.reshape((1,n,1))

V = np.zeros((n,1),dtype=complex)

for i in range(n):
    for j in range(1):
        V[i,j] = aux2[0][i][j]
```

```
n1=3

aux2 = np.loadtxt('C1.txt', dtype='complex')
aux3 = aux2.reshape((1,n1,n1))

C1 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C1[i,j] = aux3[0][i][j]

S1=np.zeros((n1,1),dtype=complex)
S1[0,0]=complex(19000000,-13500000)
S1[1,0]=complex(2191342.961,23204482.67)
S1[2,0]=complex(-21191342.94,-9704482.688)

aux4 = np.loadtxt('C2.txt', dtype='complex')
aux5 = aux4.reshape((1,n1,n1))

C2 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C2[i,j] = aux5[0][i][j]

S2=np.zeros((n1,1),dtype=complex)
S2[0,0]=complex(10000000,-6500000)
S2[1,0]=complex(629165.115,11910254.04)
S2[2,0]=complex(-10629165.12,-5410254.047)

aux6 = np.loadtxt('C3.txt', dtype='complex')
aux7 = aux6.reshape((1,n1,n1))

C3 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C3[i,j] = aux7[0][i][j]
```

```
S3=np.zeros((n1,1),dtype=complex)
S3[0,0]=complex(15000000,-11000000)
S3[1,0]=complex(2026279.443,18490381.06)
S3[2,0]=complex(-17026279.44,-7490381.069)


aux8 = np.loadtxt('C4.txt', dtype='complex')
aux9 = aux8.reshape((1,n1,n1))

C4 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C4[i,j] = aux9[0][i][j]

S4=np.zeros((n1,1),dtype=complex)
S4[0,0]=complex(5000000,-3500000)
S4[1,0]=complex(531088.9134,6080127.019)
S4[2,0]=complex(-5531088.913,-2580127.019)


aux10 = np.loadtxt('C5.txt', dtype='complex')
aux11 = aux10.reshape((1,n1,n1))

C5 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C5[i,j] = aux11[0][i][j]

S5=np.zeros((n1,1),dtype=complex)
S5[0,0]=complex(5000000,-1500000)
S5[1,0]=complex(-1200961.897,5080127.018)
S5[2,0]=complex(-3799038.103,-3580127.021)

aux12 = np.loadtxt('C6.txt', dtype='complex')
aux13 = aux12.reshape((1,n1,n1))
```

```
C6 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C6[i,j] = aux13[0][i][j]

S6=np.zeros((n1,1),dtype=complex)
S6[0,0]=complex(4500000,-2000000)
S6[1,0]=complex(-517949.192,4897114.317)
S6[2,0]=complex(-4424633.127,-2161624.78)

aux14 = np.loadtxt('C7.txt', dtype='complex')
aux15 = aux14.reshape((1,n1,n1))

C7 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C7[i,j] = aux15[0][i][j]

S7=np.zeros((n1,1),dtype=complex)
S7[0,0]=complex(10000000,-7000000)
S7[1,0]=complex(1062177.834,12160254.04)
S7[2,0]=complex(-11062177.83,-5160254.039)


I1=np.zeros((n1,1),dtype=complex)
V1=np.zeros((n1,1),dtype=complex)
I2=np.zeros((n1,1),dtype=complex)
V2=np.zeros((n1,1),dtype=complex)
I3=np.zeros((n1,1),dtype=complex)
V3=np.zeros((n1,1),dtype=complex)
I4=np.zeros((n1,1),dtype=complex)
V4=np.zeros((n1,1),dtype=complex)
I5=np.zeros((n1,1),dtype=complex)
V5=np.zeros((n1,1),dtype=complex)
I6=np.zeros((n1,1),dtype=complex)
V6=np.zeros((n1,1),dtype=complex)
I7=np.zeros((n1,1),dtype=complex)
```

```
V7=np.zeros((n1,1),dtype=complex)

for i in range(15,18):
    I1[i-15,0]=S1[i-15,0]/(3*V[i,0])
    V1[i-15,0]=V[i,0]
H1=np.dot(C1,V1)
for j in range(15,18):
    I[i,0]=I1[i-15,0]-H1[i-15,0]


for i in range(21,24):
    I2[i-21,0]=S2[i-21,0]/(3*V[i,0])
    V2[i-21,0]=V[i,0]
H2=np.dot(C2,V2)
for j in range(21,24):
    I[i,0]=I2[i-21,0]-H2[i-21,0]


for i in range(27,29):
    I3[i-27,0]=S3[i-27,0]/(3*V[i,0])
    V3[i-27,0]=V[i,0]
H3=np.dot(C3,V3)
for j in range(27,29):
    I[i,0]=I3[i-27,0]-H3[i-27,0]


for i in range(33,36):
    I4[i-33,0]=S4[i-33,0]/(3*V[i,0])
    V4[i-33,0]=V[i,0]
H4=np.dot(C4,V4)
for j in range(33,36):
    I[i,0]=I4[i-33,0]-H4[i-33,0]


for i in range(39,42):
    I5[i-39,0]=S5[i-39,0]/(3*V[i,0])
    V5[i-39,0]=V[i,0]
H5=np.dot(C5,V5)
for j in range(39,42):
```

```
    I[i,0]=I5[i-39,0]-H5[i-39,0]


for i in range(45,48):
    I6[i-45,0]=S6[i-45,0]/(3*V[i,0])
    V6[i-45,0]=V[i,0]
H6=np.dot(C6,V6)
for j in range(45,48):
    I[i,0]=I6[i-45,0]-H6[i-45,0]


for i in range(51,54):
    I7[i-51,0]=S7[i-51,0]/(3*V[i,0])
    V7[i-51,0]=V[i,0]
H7=np.dot(C7,V7)
for j in range(51,54):
    I[i,0]=I7[i-51,0]-H7[i-51,0]


t = 1
T=10**(-t)
IterMax = 50

V8 = np.zeros((n,1), dtype=complex)
T1 = np.zeros((n,1), dtype=complex)
T2 = np.zeros((n,1), dtype=complex)

print('Gauss-Seidel - CSR')
Iter=0
X=V

cont1=0
while Iter<IterMax:
    Iter=Iter+1
    for k in range(n):
      d=int(IA[0,k+1]-IA[0,k])
      soma = 0
      soma1 = 1
      for k1 in range(d):
```

```
        d1=int(JA[0,cont1])
        if (k > d1):
            soma=soma+(AN[0,cont1]*V[d1,0])
        elif (k < d1):
            soma1=soma1+(AN[0,cont1]*X[d1,0])
        cont1=cont1+1
        if cont1==soma11:
            cont1=0
    V8[k,0]=(1/T3[k,0])*(I[k,0]-soma1-soma)
    t1 = (V8[k,0]-X[k,0])
    T1[k,0] = t1
    t2 = (V8[k,0])
    T2[k,0] = t2
print()
print('Iteração = ',Iter)
print()
print(V8)
print()
t3 = max(abs(T1))
t4 = max(abs(T2))
t5=t3/t4
if t5<T:
    break
else:
    V=X
    X=V8
    for i in range(15,18):
        I1[i-15,0]=S1[i-15,0]/(3*V8[i,0])
        V1[i-15,0]=V8[i,0]
    H1=np.dot(C1,V1)
    for j in range(15,18):
        I[i,0]=I1[i-15,0]-H1[i-15,0]


    for i in range(21,24):
        I2[i-21,0]=S2[i-21,0]/(3*V8[i,0])
        V2[i-21,0]=V8[i,0]
    H2=np.dot(C2,V2)
    for j in range(21,24):
```

```
    I[i,0]=I2[i-21,0]-H2[i-21,0]


for i in range(27,29):
    I3[i-27,0]=S3[i-27,0]/(3*V8[i,0])
    V3[i-27,0]=V8[i,0]
H3=np.dot(C3,V3)
for j in range(27,29):
    I[i,0]=I3[i-27,0]-H3[i-27,0]


for i in range(33,36):
    I4[i-33,0]=S4[i-33,0]/(3*V8[i,0])
    V4[i-33,0]=V8[i,0]
H4=np.dot(C4,V4)
for j in range(33,36):
    I[i,0]=I4[i-33,0]-H4[i-33,0]


for i in range(39,42):
    I5[i-39,0]=S5[i-39,0]/(3*V8[i,0])
    V5[i-39,0]=V8[i,0]
H5=np.dot(C5,V5)
for j in range(39,42):
    I[i,0]=I5[i-39,0]-H5[i-39,0]


for i in range(45,48):
    I6[i-45,0]=S6[i-45,0]/(3*V8[i,0])
    V6[i-45,0]=V8[i,0]
H6=np.dot(C6,V6)
for j in range(45,48):
    I[i,0]=I6[i-45,0]-H6[i-45,0]


for i in range(51,54):
    I7[i-51,0]=S7[i-51,0]/(3*V8[i,0])
    V7[i-51,0]=V8[i,0]
H7=np.dot(C7,V7)
```

```
        for j in range(51,54):
            I[i,0]=I7[i-51,0]-H7[i-51,0]

    if cont1==soma11:
        cont1=0

print()
print('Matriz V')
print()
print(V8)
print()
print('Iterações =',Iter)


fim = time.perf_counter()
print()
print('Tempo de Processamento %f' % (fim - inicio))
print()
print(psutil.virtual_memory())

sys.stdout.flush()
sys.exit()
```

A.2 Método CSC

A.2.1 Integrado ao Método de Jacobi

```
import numpy as np
import time
import psutil
import sys

sys.stdout=open("1cscjac.txt","a")

inicio = time.process_time()
```

```
n=54

aux = np.loadtxt('inputys1.txt', dtype='complex')
aux = aux.reshape((1,n,n))

YS = np.zeros((n,n),dtype=complex)

for i in range(n):
    for j in range(n):
        YS[i,j] = aux[0][i][j]

T3 = np.zeros((n,1),dtype=complex)
for i in range(n):
    T3[i,0] = YS[i,i]

soma1=0
for i in range(n):
    for j in range(n):
        if YS[i,j] != 0:
            soma1 = soma1 + 1

AN = np.zeros((1,soma1),dtype=complex)
JA = np.zeros((1,soma1))
IA = np.zeros((1,n+1))
IA[0,0] = 0
h=0
z=0
for j in range(n):
    for i in range(n):
        if YS[i,j] != 0:
            AN[0,h] = YS[i,j]
            JA[0,h] = i
            h=h+1
            z=z+1
        IA[0,j+1] = z


aux1 = np.loadtxt('inputi1.txt', dtype='complex')
aux1 = aux1.reshape((1,n,1))
```

```
I = np.zeros((n,1),dtype=complex)

for i in range(n):
    for j in range(1):
        I[i,j] = aux1[0][i][j]


aux2 = np.loadtxt('inputx1.txt', dtype='complex')
aux2 = aux2.reshape((1,n,1))

V8 = np.zeros((n,1),dtype=complex)

for i in range(n):
    for j in range(1):
        V8[i,j] = aux2[0][i][j]

n1=3

aux2 = np.loadtxt('C1.txt', dtype='complex')
aux3 = aux2.reshape((1,n1,n1))

C1 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C1[i,j] = aux3[0][i][j]

S1=np.zeros((n1,1),dtype=complex)
S1[0,0]=complex(19000000,-13500000)
S1[1,0]=complex(2191342.961,23204482.67)
S1[2,0]=complex(-21191342.94,-9704482.688)

aux4 = np.loadtxt('C2.txt', dtype='complex')
aux5 = aux4.reshape((1,n1,n1))

C2 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
```

```
    for j in range(n1):
        C2[i,j] = aux5[0][i][j]

S2=np.zeros((n1,1),dtype=complex)
S2[0,0]=complex(10000000,-6500000)
S2[1,0]=complex(629165.115,11910254.04)
S2[2,0]=complex(-10629165.12,-5410254.047)

aux6 = np.loadtxt('C3.txt', dtype='complex')
aux7 = aux6.reshape((1,n1,n1))

C3 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C3[i,j] = aux7[0][i][j]

S3=np.zeros((n1,1),dtype=complex)
S3[0,0]=complex(15000000,-11000000)
S3[1,0]=complex(2026279.443,18490381.06)
S3[2,0]=complex(-17026279.44,-7490381.069)


aux8 = np.loadtxt('C4.txt', dtype='complex')
aux9 = aux8.reshape((1,n1,n1))

C4 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C4[i,j] = aux9[0][i][j]

S4=np.zeros((n1,1),dtype=complex)
S4[0,0]=complex(5000000,-3500000)
S4[1,0]=complex(531088.9134,6080127.019)
S4[2,0]=complex(-5531088.913,-2580127.019)


aux10 = np.loadtxt('C5.txt', dtype='complex')
```

```
aux11 = aux10.reshape((1,n1,n1))

C5 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C5[i,j] = aux11[0][i][j]

S5=np.zeros((n1,1),dtype=complex)
S5[0,0]=complex(5000000,-1500000)
S5[1,0]=complex(-1200961.897,5080127.018)
S5[2,0]=complex(-3799038.103,-3580127.021)

aux12 = np.loadtxt('C6.txt', dtype='complex')
aux13 = aux12.reshape((1,n1,n1))

C6 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C6[i,j] = aux13[0][i][j]

S6=np.zeros((n1,1),dtype=complex)
S6[0,0]=complex(4500000,-2000000)
S6[1,0]=complex(-517949.192,4897114.317)
S6[2,0]=complex(-4424633.127,-2161624.78)

aux14 = np.loadtxt('C7.txt', dtype='complex')
aux15 = aux14.reshape((1,n1,n1))

C7 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C7[i,j] = aux15[0][i][j]

S7=np.zeros((n1,1),dtype=complex)
S7[0,0]=complex(10000000,-7000000)
S7[1,0]=complex(1062177.834,12160254.04)
```

```
S7[2,0]=complex(-11062177.83,-5160254.039)


I1=np.zeros((n1,1),dtype=complex)
V1=np.zeros((n1,1),dtype=complex)
I2=np.zeros((n1,1),dtype=complex)
V2=np.zeros((n1,1),dtype=complex)
I3=np.zeros((n1,1),dtype=complex)
V3=np.zeros((n1,1),dtype=complex)
I4=np.zeros((n1,1),dtype=complex)
V4=np.zeros((n1,1),dtype=complex)
I5=np.zeros((n1,1),dtype=complex)
V5=np.zeros((n1,1),dtype=complex)
I6=np.zeros((n1,1),dtype=complex)
V6=np.zeros((n1,1),dtype=complex)
I7=np.zeros((n1,1),dtype=complex)
V7=np.zeros((n1,1),dtype=complex)

for i in range(15,18):
    I1[i-15,0]=S1[i-15,0]/(3*V8[i,0])
    V1[i-15,0]=V8[i,0]
H1=np.dot(C1,V1)
for j in range(15,18):
    I[i,0]=I1[i-15,0]-H1[i-15,0]


for i in range(21,24):
    I2[i-21,0]=S2[i-21,0]/(3*V8[i,0])
    V2[i-21,0]=V8[i,0]
H2=np.dot(C2,V2)
for j in range(21,24):
    I[i,0]=I2[i-21,0]-H2[i-21,0]


for i in range(27,29):
    I3[i-27,0]=S3[i-27,0]/(3*V8[i,0])
    V3[i-27,0]=V8[i,0]
H3=np.dot(C3,V3)
for j in range(27,29):
```

```
    I[i,0]=I3[i-27,0]-H3[i-27,0]


for i in range(33,36):
    I4[i-33,0]=S4[i-33,0]/(3*V8[i,0])
    V4[i-33,0]=V8[i,0]
H4=np.dot(C4,V4)
for j in range(33,36):
    I[i,0]=I4[i-33,0]-H4[i-33,0]


for i in range(39,42):
    I5[i-39,0]=S5[i-39,0]/(3*V8[i,0])
    V5[i-39,0]=V8[i,0]
H5=np.dot(C5,V5)
for j in range(39,42):
    I[i,0]=I5[i-39,0]-H5[i-39,0]


for i in range(45,48):
    I6[i-45,0]=S6[i-45,0]/(3*V8[i,0])
    V6[i-45,0]=V8[i,0]
H6=np.dot(C6,V6)
for j in range(45,48):
    I[i,0]=I6[i-45,0]-H6[i-45,0]


for i in range(51,54):
    I7[i-51,0]=S7[i-51,0]/(3*V8[i,0])
    V7[i-51,0]=V8[i,0]
H7=np.dot(C7,V7)
for j in range(51,54):
    I[i,0]=I7[i-51,0]-H7[i-51,0]


t = 1
T=10**-t
IterMax = 50
```

```
X = np.zeros((n,1),dtype=complex)
T1 = np.zeros((n,1), dtype=complex)
T2 = np.zeros((n,1), dtype=complex)

print('Jacobi-CSC')
print()
Iter=0
X=V8

cont1=0
while Iter<=IterMax:
    Iter=Iter+1
    V = np.zeros((n,1),dtype=complex)
    for k in range(n):
        d=int(IA[0,k+1]-IA[0,k])
        soma=0
        for k1 in range(d):
            d1=int(JA[0,cont1])
            if k != d1:
                soma=soma+(X[d1,0]*AN[0,cont1])
                cont1=cont1+1
            else:
                cont1=cont1+1
                continue
        V[k,0]=(I[k,0]-soma)/T3[k,0]
        t1 = (V[k,0]-X[k,0])
        T1[k,0] = t1
        t2 = (V[k,0])
        T2[k,0] = t2
    print()
    print('Iteração =',Iter)
    print()
    print(V)
    print()

    t3 = max(abs(T1))
    t4 = max(abs(T2))
    t5=t3/t4
```

```
if t5<T:
    break
else:
    X=V
    for i in range(15,18):
        I1[i-15,0]=S1[i-15,0]/(3*X[i,0])
        V1[i-15,0]=X[i,0]
    H1=np.dot(C1,V1)
    for j in range(15,18):
        I[i,0]=I1[i-15,0]-H1[i-15,0]


    for i in range(21,24):
        I2[i-21,0]=S2[i-21,0]/(3*X[i,0])
        V2[i-21,0]=X[i,0]
    H2=np.dot(C2,V2)
    for j in range(21,24):
        I[i,0]=I2[i-21,0]-H2[i-21,0]


    for i in range(27,29):
        I3[i-27,0]=S3[i-27,0]/(3*X[i,0])
        V3[i-27,0]=X[i,0]
    H3=np.dot(C3,V3)
    for j in range(27,29):
        I[i,0]=I3[i-27,0]-H3[i-27,0]


    for i in range(33,36):
        I4[i-33,0]=S4[i-33,0]/(3*X[i,0])
        V4[i-33,0]=X[i,0]
    H4=np.dot(C4,V4)
    for j in range(33,36):
        I[i,0]=I4[i-33,0]-H4[i-33,0]


    for i in range(39,42):
        I5[i-39,0]=S5[i-39,0]/(3*X[i,0])
        V5[i-39,0]=X[i,0]
```

```
            H5=np.dot(C5,V5)
            for j in range(39,42):
                I[i,0]=I5[i-39,0]-H5[i-39,0]


            for i in range(45,48):
                I6[i-45,0]=S6[i-45,0]/(3*X[i,0])
                V6[i-45,0]=X[i,0]
            H6=np.dot(C6,V6)
            for j in range(45,48):
                I[i,0]=I6[i-45,0]-H6[i-45,0]


            for i in range(51,54):
                I7[i-51,0]=S7[i-51,0]/(3*X[i,0])
                V7[i-51,0]=X[i,0]
            H7=np.dot(C7,V7)
            for j in range(51,54):
                I[i,0]=I7[i-51,0]-H7[i-51,0]

        if cont1==soma1:
            cont1=0


print()
print('Matriz V')
print()
print(V)
print()
print('Iterações =',Iter)


fim = time.process_time()
print()
print('Tempo de Processamento %f' % (fim - inicio))
print()
print(psutil.virtual_memory())

sys.stdout.flush()
```

```
sys.exit()
```

A.2.2 Integrado ao Método de Gauss-Seidel

```
import numpy as np
import time
import psutil
import sys

sys.stdout=open("1cscgs.txt","a")

inicio = time.perf_counter()

n=54

aux = np.loadtxt('inputys1.txt', dtype='complex')
aux = aux.reshape((1,n,n))

YS = np.zeros((n,n),dtype=complex)

for i in range(n):
    for j in range(n):
        YS[i,j] = aux[0][i][j]

T3 = np.zeros((n,1),dtype=complex)
for i in range(n):
    T3[i,0] = YS[i,i]

soma11=0
for i in range(n):
    for j in range(n):
        if YS[i,j] != 0:
            soma11 = soma11 + 1

AN = np.zeros((1,soma11),dtype=complex)
JA = np.zeros((1,soma11))
```

```
IA = np.zeros((1,n+1))
IA[0,0] = 0
h=0
z=0
for j in range(n):
    for i in range(n):
        if YS[i,j] != 0:
            AN[0,h] = YS[i,j]
            JA[0,h] = i
            h=h+1
            z=z+1
        IA[0,j+1] = z

aux1 = np.loadtxt('inputi1.txt', dtype='complex')
aux1 = aux1.reshape((1,n,1))

I = np.zeros((n,1),dtype=complex)

for i in range(n):
    for j in range(1):
        I[i,j] = aux1[0][i][j]

aux2 = np.loadtxt('inputx1.txt', dtype='complex')
aux2 = aux2.reshape((1,n,1))

V = np.zeros((n,1),dtype=complex)

for i in range(n):
    for j in range(1):
        V[i,j] = aux2[0][i][j]

n1=3

aux2 = np.loadtxt('C1.txt', dtype='complex')
aux3 = aux2.reshape((1,n1,n1))

C1 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
```

```
    for j in range(n1):
        C1[i,j] = aux3[0][i][j]

S1=np.zeros((n1,1),dtype=complex)
S1[0,0]=complex(19000000,-13500000)
S1[1,0]=complex(2191342.961,23204482.67)
S1[2,0]=complex(-21191342.94,-9704482.688)

aux4 = np.loadtxt('C2.txt', dtype='complex')
aux5 = aux4.reshape((1,n1,n1))

C2 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C2[i,j] = aux5[0][i][j]

S2=np.zeros((n1,1),dtype=complex)
S2[0,0]=complex(10000000,-6500000)
S2[1,0]=complex(629165.115,11910254.04)
S2[2,0]=complex(-10629165.12,-5410254.047)

aux6 = np.loadtxt('C3.txt', dtype='complex')
aux7 = aux6.reshape((1,n1,n1))

C3 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C3[i,j] = aux7[0][i][j]

S3=np.zeros((n1,1),dtype=complex)
S3[0,0]=complex(15000000,-11000000)
S3[1,0]=complex(2026279.443,18490381.06)
S3[2,0]=complex(-17026279.44,-7490381.069)


aux8 = np.loadtxt('C4.txt', dtype='complex')
aux9 = aux8.reshape((1,n1,n1))
```

```
C4 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C4[i,j] = aux9[0][i][j]

S4=np.zeros((n1,1),dtype=complex)
S4[0,0]=complex(5000000,-3500000)
S4[1,0]=complex(531088.9134,6080127.019)
S4[2,0]=complex(-5531088.913,-2580127.019)


aux10 = np.loadtxt('C5.txt', dtype='complex')
aux11 = aux10.reshape((1,n1,n1))

C5 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C5[i,j] = aux11[0][i][j]

S5=np.zeros((n1,1),dtype=complex)
S5[0,0]=complex(5000000,-1500000)
S5[1,0]=complex(-1200961.897,5080127.018)
S5[2,0]=complex(-3799038.103,-3580127.021)

aux12 = np.loadtxt('C6.txt', dtype='complex')
aux13 = aux12.reshape((1,n1,n1))

C6 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C6[i,j] = aux13[0][i][j]

S6=np.zeros((n1,1),dtype=complex)
S6[0,0]=complex(4500000,-2000000)
S6[1,0]=complex(-517949.192,4897114.317)
```

```
S6[2,0]=complex(-4424633.127,-2161624.78)

aux14 = np.loadtxt('C7.txt', dtype='complex')
aux15 = aux14.reshape((1,n1,n1))

C7 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C7[i,j] = aux15[0][i][j]

S7=np.zeros((n1,1),dtype=complex)
S7[0,0]=complex(10000000,-7000000)
S7[1,0]=complex(1062177.834,12160254.04)
S7[2,0]=complex(-11062177.83,-5160254.039)


I1=np.zeros((n1,1),dtype=complex)
V1=np.zeros((n1,1),dtype=complex)
I2=np.zeros((n1,1),dtype=complex)
V2=np.zeros((n1,1),dtype=complex)
I3=np.zeros((n1,1),dtype=complex)
V3=np.zeros((n1,1),dtype=complex)
I4=np.zeros((n1,1),dtype=complex)
V4=np.zeros((n1,1),dtype=complex)
I5=np.zeros((n1,1),dtype=complex)
V5=np.zeros((n1,1),dtype=complex)
I6=np.zeros((n1,1),dtype=complex)
V6=np.zeros((n1,1),dtype=complex)
I7=np.zeros((n1,1),dtype=complex)
V7=np.zeros((n1,1),dtype=complex)

for i in range(15,18):
    I1[i-15,0]=S1[i-15,0]/(3*V[i,0])
    V1[i-15,0]=V[i,0]
H1=np.dot(C1,V1)
for j in range(15,18):
    I[i,0]=I1[i-15,0]-H1[i-15,0]
```

```
for i in range(21,24):
    I2[i-21,0]=S2[i-21,0]/(3*V[i,0])
    V2[i-21,0]=V[i,0]
H2=np.dot(C2,V2)
for j in range(21,24):
    I[i,0]=I2[i-21,0]-H2[i-21,0]


for i in range(27,29):
    I3[i-27,0]=S3[i-27,0]/(3*V[i,0])
    V3[i-27,0]=V[i,0]
H3=np.dot(C3,V3)
for j in range(27,29):
    I[i,0]=I3[i-27,0]-H3[i-27,0]


for i in range(33,36):
    I4[i-33,0]=S4[i-33,0]/(3*V[i,0])
    V4[i-33,0]=V[i,0]
H4=np.dot(C4,V4)
for j in range(33,36):
    I[i,0]=I4[i-33,0]-H4[i-33,0]


for i in range(39,42):
    I5[i-39,0]=S5[i-39,0]/(3*V[i,0])
    V5[i-39,0]=V[i,0]
H5=np.dot(C5,V5)
for j in range(39,42):
    I[i,0]=I5[i-39,0]-H5[i-39,0]


for i in range(45,48):
    I6[i-45,0]=S6[i-45,0]/(3*V[i,0])
    V6[i-45,0]=V[i,0]
H6=np.dot(C6,V6)
for j in range(45,48):
    I[i,0]=I6[i-45,0]-H6[i-45,0]
```

```
for i in range(51,54):
    I7[i-51,0]=S7[i-51,0]/(3*V[i,0])
    V7[i-51,0]=V[i,0]
H7=np.dot(C7,V7)
for j in range(51,54):
    I[i,0]=I7[i-51,0]-H7[i-51,0]


t = 1
T=10**(-t)
IterMax = 50

V8 = np.zeros((n,1), dtype=complex)
T1 = np.zeros((n,1), dtype=complex)
T2 = np.zeros((n,1), dtype=complex)

print('Gauss-Seidel - CSC')
Iter=0
X=V

cont1=0
while Iter<IterMax:
    Iter=Iter+1
    for k in range(n):
      d=int(IA[0,k+1]-IA[0,k])
      soma = 0
      soma1 = 1
      for k1 in range(d):
          d1=int(JA[0,cont1])
          if (k > d1):
              soma=soma+(AN[0,cont1]*V[d1,0])
          elif (k < d1):
              soma1=soma1+(AN[0,cont1]*X[d1,0])
          cont1=cont1+1
          if cont1==soma11:
              cont1=0
      V8[k,0]=(1/T3[k,0])*(I[k,0]-soma1-soma)
```

```
  t1 = (V8[k,0]-X[k,0])
  T1[k,0] = t1
  t2 = (V8[k,0])
  T2[k,0] = t2
print()
print('Iteração = ',Iter)
print()
print(V8)
print()
t3 = max(abs(T1))
t4 = max(abs(T2))
t5=t3/t4
if t5<T:
    break
else:
    V=X
    X=V8
    for i in range(15,18):
        I1[i-15,0]=S1[i-15,0]/(3*V8[i,0])
        V1[i-15,0]=V8[i,0]
    H1=np.dot(C1,V1)
    for j in range(15,18):
        I[i,0]=I1[i-15,0]-H1[i-15,0]


    for i in range(21,24):
        I2[i-21,0]=S2[i-21,0]/(3*V8[i,0])
        V2[i-21,0]=V8[i,0]
    H2=np.dot(C2,V2)
    for j in range(21,24):
        I[i,0]=I2[i-21,0]-H2[i-21,0]


    for i in range(27,29):
        I3[i-27,0]=S3[i-27,0]/(3*V8[i,0])
        V3[i-27,0]=V8[i,0]
    H3=np.dot(C3,V3)
    for j in range(27,29):
        I[i,0]=I3[i-27,0]-H3[i-27,0]
```

```
            for i in range(33,36):
                I4[i-33,0]=S4[i-33,0]/(3*V8[i,0])
                V4[i-33,0]=V8[i,0]
            H4=np.dot(C4,V4)
            for j in range(33,36):
                I[i,0]=I4[i-33,0]-H4[i-33,0]


            for i in range(39,42):
                I5[i-39,0]=S5[i-39,0]/(3*V8[i,0])
                V5[i-39,0]=V8[i,0]
            H5=np.dot(C5,V5)
            for j in range(39,42):
                I[i,0]=I5[i-39,0]-H5[i-39,0]


            for i in range(45,48):
                I6[i-45,0]=S6[i-45,0]/(3*V8[i,0])
                V6[i-45,0]=V8[i,0]
            H6=np.dot(C6,V6)
            for j in range(45,48):
                I[i,0]=I6[i-45,0]-H6[i-45,0]


            for i in range(51,54):
                I7[i-51,0]=S7[i-51,0]/(3*V8[i,0])
                V7[i-51,0]=V8[i,0]
            H7=np.dot(C7,V7)
            for j in range(51,54):
                I[i,0]=I7[i-51,0]-H7[i-51,0]

        if cont1==soma11:
            cont1=0

print()
print('Matriz V')
print()
```

```
print(V8)
print()
print('Iterações =',Iter)

fim = time.perf_counter()
print()
print('Tempo de Processamento %f' % (fim - inicio))
print()
print(psutil.virtual_memory())

sys.stdout.flush()
sys.exit()
```

A.3 Método CSV

A.3.1 Integrado ao Método de Jacobi

```
import numpy as np
import time
import psutil
import sys

sys.stdout=open("1csvjac.txt","a")

inicio = time.process_time()

n=54

aux = np.loadtxt('inputys1.txt', dtype='complex')
aux = aux.reshape((1,n,n))

YS = np.zeros((n,n),dtype=complex)

for i in range(n):
    for j in range(n):
        YS[i,j] = aux[0][i][j]
```

```
T3 = np.zeros((n,1),dtype=complex)
for i in range(n):
    T3[i,0] = YS[i,i]

soma1=0
for i in range(n):
    for j in range(n):
        if YS[i,j] != 0:
            soma1 = soma1 + 1

JN = np.zeros((1,n))
for i in range(n):
    for j in range(n):
        if YS[i,j] != 0:
            JN[0,i]=j
            break

AA = np.zeros((1,soma1),dtype=complex)
IA = np.zeros((1,soma1))
NZ = np.zeros((1,n))
h=0
z=0
for i in range(n):
    r=0
    for j in range(n):
        z=z+1
        if YS[i,j] != 0:
            AA[0,h] = YS[i,j]
            IA[0,h] = z
            h=h+1
            r=r+1
            z=0
    NZ[0,i]=r


aux1 = np.loadtxt('inputi1.txt', dtype='complex')
aux1 = aux1.reshape((1,n,1))

I = np.zeros((n,1),dtype=complex)
```

```
for i in range(n):
    for j in range(1):
        I[i,j] = aux1[0][i][j]


aux2 = np.loadtxt('inputx1.txt', dtype='complex')
aux2 = aux2.reshape((1,n,1))

V8 = np.zeros((n,1),dtype=complex)

for i in range(n):
    for j in range(1):
        V8[i,j] = aux2[0][i][j]

n1=3

aux2 = np.loadtxt('C1.txt', dtype='complex')
aux3 = aux2.reshape((1,n1,n1))

C1 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C1[i,j] = aux3[0][i][j]

S1=np.zeros((n1,1),dtype=complex)
S1[0,0]=complex(19000000,-13500000)
S1[1,0]=complex(2191342.961,23204482.67)
S1[2,0]=complex(-21191342.94,-9704482.688)

aux4 = np.loadtxt('C2.txt', dtype='complex')
aux5 = aux4.reshape((1,n1,n1))

C2 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C2[i,j] = aux5[0][i][j]
```

```
S2=np.zeros((n1,1),dtype=complex)
S2[0,0]=complex(10000000,-6500000)
S2[1,0]=complex(629165.115,11910254.04)
S2[2,0]=complex(-10629165.12,-5410254.047)

aux6 = np.loadtxt('C3.txt', dtype='complex')
aux7 = aux6.reshape((1,n1,n1))

C3 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C3[i,j] = aux7[0][i][j]

S3=np.zeros((n1,1),dtype=complex)
S3[0,0]=complex(15000000,-11000000)
S3[1,0]=complex(2026279.443,18490381.06)
S3[2,0]=complex(-17026279.44,-7490381.069)


aux8 = np.loadtxt('C4.txt', dtype='complex')
aux9 = aux8.reshape((1,n1,n1))

C4 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C4[i,j] = aux9[0][i][j]

S4=np.zeros((n1,1),dtype=complex)
S4[0,0]=complex(5000000,-3500000)
S4[1,0]=complex(531088.9134,6080127.019)
S4[2,0]=complex(-5531088.913,-2580127.019)


aux10 = np.loadtxt('C5.txt', dtype='complex')
aux11 = aux10.reshape((1,n1,n1))
```

```
C5 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C5[i,j] = aux11[0][i][j]

S5=np.zeros((n1,1),dtype=complex)
S5[0,0]=complex(5000000,-1500000)
S5[1,0]=complex(-1200961.897,5080127.018)
S5[2,0]=complex(-3799038.103,-3580127.021)

aux12 = np.loadtxt('C6.txt', dtype='complex')
aux13 = aux12.reshape((1,n1,n1))

C6 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C6[i,j] = aux13[0][i][j]

S6=np.zeros((n1,1),dtype=complex)
S6[0,0]=complex(4500000,-2000000)
S6[1,0]=complex(-517949.192,4897114.317)
S6[2,0]=complex(-4424633.127,-2161624.78)

aux14 = np.loadtxt('C7.txt', dtype='complex')
aux15 = aux14.reshape((1,n1,n1))

C7 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C7[i,j] = aux15[0][i][j]

S7=np.zeros((n1,1),dtype=complex)
S7[0,0]=complex(10000000,-7000000)
S7[1,0]=complex(1062177.834,12160254.04)
S7[2,0]=complex(-11062177.83,-5160254.039)
```

```
I1=np.zeros((n1,1),dtype=complex)
V1=np.zeros((n1,1),dtype=complex)
I2=np.zeros((n1,1),dtype=complex)
V2=np.zeros((n1,1),dtype=complex)
I3=np.zeros((n1,1),dtype=complex)
V3=np.zeros((n1,1),dtype=complex)
I4=np.zeros((n1,1),dtype=complex)
V4=np.zeros((n1,1),dtype=complex)
I5=np.zeros((n1,1),dtype=complex)
V5=np.zeros((n1,1),dtype=complex)
I6=np.zeros((n1,1),dtype=complex)
V6=np.zeros((n1,1),dtype=complex)
I7=np.zeros((n1,1),dtype=complex)
V7=np.zeros((n1,1),dtype=complex)

for i in range(15,18):
    I1[i-15,0]=S1[i-15,0]/(3*V8[i,0])
    V1[i-15,0]=V8[i,0]
H1=np.dot(C1,V1)
for j in range(15,18):
    I[i,0]=I1[i-15,0]-H1[i-15,0]


for i in range(21,24):
    I2[i-21,0]=S2[i-21,0]/(3*V8[i,0])
    V2[i-21,0]=V8[i,0]
H2=np.dot(C2,V2)
for j in range(21,24):
    I[i,0]=I2[i-21,0]-H2[i-21,0]


for i in range(27,29):
    I3[i-27,0]=S3[i-27,0]/(3*V8[i,0])
    V3[i-27,0]=V8[i,0]
H3=np.dot(C3,V3)
for j in range(27,29):
    I[i,0]=I3[i-27,0]-H3[i-27,0]
```

```
for i in range(33,36):
    I4[i-33,0]=S4[i-33,0]/(3*V8[i,0])
    V4[i-33,0]=V8[i,0]
H4=np.dot(C4,V4)
for j in range(33,36):
    I[i,0]=I4[i-33,0]-H4[i-33,0]


for i in range(39,42):
    I5[i-39,0]=S5[i-39,0]/(3*V8[i,0])
    V5[i-39,0]=V8[i,0]
H5=np.dot(C5,V5)
for j in range(39,42):
    I[i,0]=I5[i-39,0]-H5[i-39,0]


for i in range(45,48):
    I6[i-45,0]=S6[i-45,0]/(3*V8[i,0])
    V6[i-45,0]=V8[i,0]
H6=np.dot(C6,V6)
for j in range(45,48):
    I[i,0]=I6[i-45,0]-H6[i-45,0]


for i in range(51,54):
    I7[i-51,0]=S7[i-51,0]/(3*V8[i,0])
    V7[i-51,0]=V8[i,0]
H7=np.dot(C7,V7)
for j in range(51,54):
    I[i,0]=I7[i-51,0]-H7[i-51,0]


t = 1
T=10**-t
IterMax = 50


X = np.zeros((n,1),dtype=complex)
```

```
T1 = np.zeros((n,1), dtype=complex)
T2 = np.zeros((n,1), dtype=complex)

print('Jacobi-CSV')
Iter=0
X=V8

cont1=0
while Iter<=IterMax:
    Iter=Iter+1
    V = np.zeros((n,1),dtype=complex)
    for k in range(n):
        d=int(NZ[0,k])
        soma=0
        d2=int(JN[0,k])
        cont=0
        for k1 in range(d):
            if AA[0,cont1] != T3[k,0]:
                d1=int(IA[0,cont1])
                if k1!=0:
                    if d1!=1:
                        d3=int(IA[0,cont1-1])
                        d4=abs(d1-d3)
                        k2=cont+d4
                        cont=int(k2)
                        if cont>=54:
                            cont=cont-54
                        soma=soma+(AA[0,cont1]*X[cont,0])
                        cont=cont+1
                        cont1=cont1+1
                    else:
                       soma=soma+(AA[0,cont1]*X[cont,0])
                       cont=cont+1
                       cont1=cont1+1
                else:
                   cont=d2
                   soma=soma+(AA[0,cont1]*X[cont,0])
                   cont=cont+1
                   cont1=cont1+1
```

```
            else:
                d1=int(IA[0,cont1])
                if k1==0:
                    cont=d2
                    cont=cont+1
                    cont1=cont1+1
                    continue
                if d1==1:
                   cont=cont+1
                   cont1=cont1+1
                   continue
                else:
                    d3=int(IA[0,cont1-1])
                    d4=abs(d1-d3)
                    k2=cont+d4
                    cont=int(k2)
                    cont=cont+1
                    cont1=cont1+1
                    continue
        V[k,0]=(I[k,0]-soma)/T3[k,0]
        t1 = (V[k,0]-X[k,0])
        T1[k,0] = t1
        t2 = (V[k,0])
        T2[k,0] = t2
    print()
    print('Iteração =',Iter)
    print()
    print(V)
    print()

    t3 = max(abs(T1))
    t4 = max(abs(T2))
    t5=t3/t4
    if t5<T:
        break
    else:
        X=V
        for i in range(15,18):
            I1[i-15,0]=S1[i-15,0]/(3*X[i,0])
```

```
    V1[i-15,0]=X[i,0]
H1=np.dot(C1,V1)
for j in range(15,18):
    I[i,0]=I1[i-15,0]-H1[i-15,0]


for i in range(21,24):
    I2[i-21,0]=S2[i-21,0]/(3*X[i,0])
    V2[i-21,0]=X[i,0]
H2=np.dot(C2,V2)
for j in range(21,24):
    I[i,0]=I2[i-21,0]-H2[i-21,0]


for i in range(27,29):
    I3[i-27,0]=S3[i-27,0]/(3*X[i,0])
    V3[i-27,0]=X[i,0]
H3=np.dot(C3,V3)
for j in range(27,29):
    I[i,0]=I3[i-27,0]-H3[i-27,0]


for i in range(33,36):
    I4[i-33,0]=S4[i-33,0]/(3*X[i,0])
    V4[i-33,0]=X[i,0]
H4=np.dot(C4,V4)
for j in range(33,36):
    I[i,0]=I4[i-33,0]-H4[i-33,0]


for i in range(39,42):
    I5[i-39,0]=S5[i-39,0]/(3*X[i,0])
    V5[i-39,0]=X[i,0]
H5=np.dot(C5,V5)
for j in range(39,42):
    I[i,0]=I5[i-39,0]-H5[i-39,0]


for i in range(45,48):
```

```
            I6[i-45,0]=S6[i-45,0]/(3*X[i,0])
            V6[i-45,0]=X[i,0]
        H6=np.dot(C6,V6)
        for j in range(45,48):
            I[i,0]=I6[i-45,0]-H6[i-45,0]


        for i in range(51,54):
            I7[i-51,0]=S7[i-51,0]/(3*X[i,0])
            V7[i-51,0]=X[i,0]
        H7=np.dot(C7,V7)
        for j in range(51,54):
            I[i,0]=I7[i-51,0]-H7[i-51,0]

    if cont1==soma1:
        cont1=0


print()
print('Matriz V')
print()
print(V)
print()
print('Iterações =',Iter)

fim = time.process_time()
print()
print('Tempo de Processamento %f' % (fim - inicio))
print()
print(psutil.virtual_memory())

sys.stdout.flush()
sys.exit()
```

A.3.2 Integrado ao Método de Gauss-Seidel

```
import numpy as np
import time
import psutil
import sys

sys.stdout=open("1csvgs.txt","a")

inicio = time.process_time()

n=54

aux = np.loadtxt('inputys1.txt', dtype='complex')
aux = aux.reshape((1,n,n))

YS = np.zeros((n,n),dtype=complex)

for i in range(n):
    for j in range(n):
        YS[i,j] = aux[0][i][j]

T3 = np.zeros((n,1),dtype=complex)
for i in range(n):
    T3[i,0] = YS[i,i]

soma11=0
for i in range(n):
    for j in range(n):
        if YS[i,j] != 0:
            soma11 = soma11 + 1

JN = np.zeros((1,n))
for i in range(n):
    for j in range(n):
        if YS[i,j] != 0:
            JN[0,i]=j
            break
```

```
AA = np.zeros((1,soma11),dtype=complex)
IA = np.zeros((1,soma11))
NZ = np.zeros((1,n))
h=0
z=0
for i in range(n):
    r=0
    for j in range(n):
        z=z+1
        if YS[i,j] != 0:
            AA[0,h] = YS[i,j]
            IA[0,h] = z
            h=h+1
            r=r+1
            z=0
    NZ[0,i]=r


aux1 = np.loadtxt('inputi1.txt', dtype='complex')
aux1 = aux1.reshape((1,n,1))

I = np.zeros((n,1),dtype=complex)

for i in range(n):
    for j in range(1):
        I[i,j] = aux1[0][i][j]


aux2 = np.loadtxt('inputx1.txt', dtype='complex')
aux2 = aux2.reshape((1,n,1))

V = np.zeros((n,1),dtype=complex)

for i in range(n):
    for j in range(1):
        V[i,j] = aux2[0][i][j]
```

```
n1=3

aux2 = np.loadtxt('C1.txt', dtype='complex')
aux3 = aux2.reshape((1,n1,n1))

C1 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C1[i,j] = aux3[0][i][j]

S1=np.zeros((n1,1),dtype=complex)
S1[0,0]=complex(19000000,-13500000)
S1[1,0]=complex(2191342.961,23204482.67)
S1[2,0]=complex(-21191342.94,-9704482.688)

aux4 = np.loadtxt('C2.txt', dtype='complex')
aux5 = aux4.reshape((1,n1,n1))

C2 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C2[i,j] = aux5[0][i][j]

S2=np.zeros((n1,1),dtype=complex)
S2[0,0]=complex(10000000,-6500000)
S2[1,0]=complex(629165.115,11910254.04)
S2[2,0]=complex(-10629165.12,-5410254.047)

aux6 = np.loadtxt('C3.txt', dtype='complex')
aux7 = aux6.reshape((1,n1,n1))

C3 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C3[i,j] = aux7[0][i][j]
```

```
S3=np.zeros((n1,1),dtype=complex)
S3[0,0]=complex(15000000,-11000000)
S3[1,0]=complex(2026279.443,18490381.06)
S3[2,0]=complex(-17026279.44,-7490381.069)


aux8 = np.loadtxt('C4.txt', dtype='complex')
aux9 = aux8.reshape((1,n1,n1))

C4 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C4[i,j] = aux9[0][i][j]

S4=np.zeros((n1,1),dtype=complex)
S4[0,0]=complex(5000000,-3500000)
S4[1,0]=complex(531088.9134,6080127.019)
S4[2,0]=complex(-5531088.913,-2580127.019)


aux10 = np.loadtxt('C5.txt', dtype='complex')
aux11 = aux10.reshape((1,n1,n1))

C5 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C5[i,j] = aux11[0][i][j]

S5=np.zeros((n1,1),dtype=complex)
S5[0,0]=complex(5000000,-1500000)
S5[1,0]=complex(-1200961.897,5080127.018)
S5[2,0]=complex(-3799038.103,-3580127.021)

aux12 = np.loadtxt('C6.txt', dtype='complex')
aux13 = aux12.reshape((1,n1,n1))

C6 = np.zeros((n1,n1),dtype=complex)
```

```
for i in range(n1):
    for j in range(n1):
        C6[i,j] = aux13[0][i][j]

S6=np.zeros((n1,1),dtype=complex)
S6[0,0]=complex(4500000,-2000000)
S6[1,0]=complex(-517949.192,4897114.317)
S6[2,0]=complex(-4424633.127,-2161624.78)

aux14 = np.loadtxt('C7.txt', dtype='complex')
aux15 = aux14.reshape((1,n1,n1))

C7 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C7[i,j] = aux15[0][i][j]

S7=np.zeros((n1,1),dtype=complex)
S7[0,0]=complex(10000000,-7000000)
S7[1,0]=complex(1062177.834,12160254.04)
S7[2,0]=complex(-11062177.83,-5160254.039)


I1=np.zeros((n1,1),dtype=complex)
V1=np.zeros((n1,1),dtype=complex)
I2=np.zeros((n1,1),dtype=complex)
V2=np.zeros((n1,1),dtype=complex)
I3=np.zeros((n1,1),dtype=complex)
V3=np.zeros((n1,1),dtype=complex)
I4=np.zeros((n1,1),dtype=complex)
V4=np.zeros((n1,1),dtype=complex)
I5=np.zeros((n1,1),dtype=complex)
V5=np.zeros((n1,1),dtype=complex)
I6=np.zeros((n1,1),dtype=complex)
V6=np.zeros((n1,1),dtype=complex)
I7=np.zeros((n1,1),dtype=complex)
V7=np.zeros((n1,1),dtype=complex)
```

```
for i in range(15,18):
    I1[i-15,0]=S1[i-15,0]/(3*V[i,0])
    V1[i-15,0]=V[i,0]
H1=np.dot(C1,V1)
for j in range(15,18):
    I[i,0]=I1[i-15,0]-H1[i-15,0]


for i in range(21,24):
    I2[i-21,0]=S2[i-21,0]/(3*V[i,0])
    V2[i-21,0]=V[i,0]
H2=np.dot(C2,V2)
for j in range(21,24):
    I[i,0]=I2[i-21,0]-H2[i-21,0]


for i in range(27,29):
    I3[i-27,0]=S3[i-27,0]/(3*V[i,0])
    V3[i-27,0]=V[i,0]
H3=np.dot(C3,V3)
for j in range(27,29):
    I[i,0]=I3[i-27,0]-H3[i-27,0]


for i in range(33,36):
    I4[i-33,0]=S4[i-33,0]/(3*V[i,0])
    V4[i-33,0]=V[i,0]
H4=np.dot(C4,V4)
for j in range(33,36):
    I[i,0]=I4[i-33,0]-H4[i-33,0]


for i in range(39,42):
    I5[i-39,0]=S5[i-39,0]/(3*V[i,0])
    V5[i-39,0]=V[i,0]
H5=np.dot(C5,V5)
for j in range(39,42):
    I[i,0]=I5[i-39,0]-H5[i-39,0]
```

```
for i in range(45,48):
    I6[i-45,0]=S6[i-45,0]/(3*V[i,0])
    V6[i-45,0]=V[i,0]
H6=np.dot(C6,V6)
for j in range(45,48):
    I[i,0]=I6[i-45,0]-H6[i-45,0]


for i in range(51,54):
    I7[i-51,0]=S7[i-51,0]/(3*V[i,0])
    V7[i-51,0]=V[i,0]
H7=np.dot(C7,V7)
for j in range(51,54):
    I[i,0]=I7[i-51,0]-H7[i-51,0]

t = 1
T=10**-t
IterMax = 50

V8 = np.zeros((n,1), dtype=complex)
X = np.zeros((n,1),dtype=complex)
T1 = np.zeros((n,1), dtype=complex)
T2 = np.zeros((n,1), dtype=complex)

print('Gauss-Seidel - CSV')
Iter=0
X=V

cont1=0
while Iter<=IterMax:
    Iter=Iter+1
    for k in range(n):
        d=int(NZ[0,k])
        soma=0
        soma1=0
        d2=int(JN[0,k])
        cont=0
```

```
    for k1 in range(d):
        d1=int(IA[0,cont1])
        if cont1>=soma11:
            cont1=0
        if k1!=0:
            if d1!=1:
                d3=int(IA[0,cont1-1])
                d4=abs(d1-d3)
                k2=cont+d4
                cont=int(k2)
                if cont>53:
                    cont=0
                if (k > cont):
                    soma=soma+(AA[0,cont1]*V[cont,0])
                elif (k < cont):
                    soma1=soma1+(AA[0,cont1]*X[cont,0])
                cont=cont+1
                cont1=cont1+1
            else:
                if (k > cont):
                    soma=soma+(AA[0,cont1]*V[cont,0])
                elif (k < cont):
                    soma1=soma1+(AA[0,cont1]*X[cont,0])
                cont=cont+1
                cont1=cont1+1
        else:
            cont=d2
            if (k > cont):
               soma=soma+(AA[0,cont1]*V[cont,0])
            elif (k < cont):
               soma1=soma1+(AA[0,cont1]*X[cont,0])
            cont=cont+1
            cont1=cont1+1
    V8[k,0]=(1/T3[k,0])*(I[k,0]-soma1-soma)
    t1 = (V8[k,0]-X[k,0])
    T1[k,0] = t1
    t2 = (V8[k,0])
    T2[k,0] = t2
print()
```

```
print('Iteração =',Iter)
print()
print(V8)
print()

t3 = max(abs(T1))
t4 = max(abs(T2))
t5=t3/t4
if t5<T:
    break
else:
    V=X
    X=V8
    for i in range(15,18):
        I1[i-15,0]=S1[i-15,0]/(3*V8[i,0])
        V1[i-15,0]=V8[i,0]
    H1=np.dot(C1,V1)
    for j in range(15,18):
        I[i,0]=I1[i-15,0]-H1[i-15,0]


    for i in range(21,24):
        I2[i-21,0]=S2[i-21,0]/(3*V8[i,0])
        V2[i-21,0]=V8[i,0]
    H2=np.dot(C2,V2)
    for j in range(21,24):
        I[i,0]=I2[i-21,0]-H2[i-21,0]


    for i in range(27,29):
        I3[i-27,0]=S3[i-27,0]/(3*V8[i,0])
        V3[i-27,0]=V8[i,0]
    H3=np.dot(C3,V3)
    for j in range(27,29):
        I[i,0]=I3[i-27,0]-H3[i-27,0]


    for i in range(33,36):
        I4[i-33,0]=S4[i-33,0]/(3*V8[i,0])
```

```
            V4[i-33,0]=V8[i,0]
        H4=np.dot(C4,V4)
        for j in range(33,36):
            I[i,0]=I4[i-33,0]-H4[i-33,0]


        for i in range(39,42):
            I5[i-39,0]=S5[i-39,0]/(3*V8[i,0])
            V5[i-39,0]=V8[i,0]
        H5=np.dot(C5,V5)
        for j in range(39,42):
            I[i,0]=I5[i-39,0]-H5[i-39,0]


        for i in range(45,48):
            I6[i-45,0]=S6[i-45,0]/(3*V8[i,0])
            V6[i-45,0]=V8[i,0]
        H6=np.dot(C6,V6)
        for j in range(45,48):
            I[i,0]=I6[i-45,0]-H6[i-45,0]


        for i in range(51,54):
            I7[i-51,0]=S7[i-51,0]/(3*V8[i,0])
            V7[i-51,0]=V8[i,0]
        H7=np.dot(C7,V7)
        for j in range(51,54):
            I[i,0]=I7[i-51,0]-H7[i-51,0]

    if cont1==soma11:
        cont1=0


print()
print('Matriz V')
print()
print(V8)
print()
print('Iterações =',Iter)
```

```
fim = time.process_time()
print()
print('Tempo de Processamento %f' % (fim - inicio))
print()
print(psutil.virtual_memory())

sys.stdout.flush()
sys.exit()
```

A.4 Método Skyline

A.4.1 Integrado ao Método de Jacobi

```
import numpy as np
import time
import psutil
import sys

sys.stdout=open("1skyjac.txt","a")

inicio = time.process_time()

n=54

aux = np.loadtxt('inputys1.txt', dtype='complex')
aux = aux.reshape((1,n,n))

YS = np.zeros((n,n),dtype=complex)

for i in range(n):
    for j in range(n):
        YS[i,j] = aux[0][i][j]

T3 = np.zeros((n,1),dtype=complex)
```

```
for i in range(n):
    T3[i,0] = YS[i,i]

soma1=0
for i in range(n):
    for j in range(n):
        if YS[i,j] != 0:
            soma1 = soma1 + 1


FCOL = np.zeros((1,n))
LCOL = np.zeros((1,n))
RW = np.zeros((1,n+1))
h=0
z=0

for i in range(n):
    for j in range(n):
        if YS[i,j] != 0:
            FCOL[0,i] = j
            break

for i in range(n):
    for j in range(n-1,-1,-1):
        if YS[i,j] != 0:
            LCOL[0,i] = j
            break

for i in range(n):
    b = int(FCOL[0,i])
    c = int(LCOL[0,i])
    z = (c-b)+1
    h=h+z

VAL = np.zeros((1,h),dtype=complex)
r=0
for i in range(n):
    b = int(FCOL[0,i])
    c = int(LCOL[0,i])
```

```
    for j in range(b,c+1,1):
        VAL[0,r] = YS[i,j]
        r=r+1
    RW[0,i+1] = r


aux1 = np.loadtxt('inputi1.txt', dtype='complex')
aux1 = aux1.reshape((1,n,1))

I = np.zeros((n,1),dtype=complex)

for i in range(n):
    for j in range(1):
        I[i,j] = aux1[0][i][j]


aux2 = np.loadtxt('inputx1.txt', dtype='complex')
aux2 = aux2.reshape((1,n,1))

V8 = np.zeros((n,1),dtype=complex)

for i in range(n):
    for j in range(1):
        V8[i,j] = aux2[0][i][j]

n1=3

aux2 = np.loadtxt('C1.txt', dtype='complex')
aux3 = aux2.reshape((1,n1,n1))

C1 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C1[i,j] = aux3[0][i][j]

S1=np.zeros((n1,1),dtype=complex)
S1[0,0]=complex(19000000,-13500000)
S1[1,0]=complex(2191342.961,23204482.67)
```

```
S1[2,0]=complex(-21191342.94,-9704482.688)

aux4 = np.loadtxt('C2.txt', dtype='complex')
aux5 = aux4.reshape((1,n1,n1))

C2 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C2[i,j] = aux5[0][i][j]

S2=np.zeros((n1,1),dtype=complex)
S2[0,0]=complex(10000000,-6500000)
S2[1,0]=complex(629165.115,11910254.04)
S2[2,0]=complex(-10629165.12,-5410254.047)

aux6 = np.loadtxt('C3.txt', dtype='complex')
aux7 = aux6.reshape((1,n1,n1))

C3 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C3[i,j] = aux7[0][i][j]

S3=np.zeros((n1,1),dtype=complex)
S3[0,0]=complex(15000000,-11000000)
S3[1,0]=complex(2026279.443,18490381.06)
S3[2,0]=complex(-17026279.44,-7490381.069)


aux8 = np.loadtxt('C4.txt', dtype='complex')
aux9 = aux8.reshape((1,n1,n1))

C4 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C4[i,j] = aux9[0][i][j]
```

```
S4=np.zeros((n1,1),dtype=complex)
S4[0,0]=complex(5000000,-3500000)
S4[1,0]=complex(531088.9134,6080127.019)
S4[2,0]=complex(-5531088.913,-2580127.019)


aux10 = np.loadtxt('C5.txt', dtype='complex')
aux11 = aux10.reshape((1,n1,n1))

C5 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C5[i,j] = aux11[0][i][j]

S5=np.zeros((n1,1),dtype=complex)
S5[0,0]=complex(5000000,-1500000)
S5[1,0]=complex(-1200961.897,5080127.018)
S5[2,0]=complex(-3799038.103,-3580127.021)

aux12 = np.loadtxt('C6.txt', dtype='complex')
aux13 = aux12.reshape((1,n1,n1))

C6 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C6[i,j] = aux13[0][i][j]

S6=np.zeros((n1,1),dtype=complex)
S6[0,0]=complex(4500000,-2000000)
S6[1,0]=complex(-517949.192,4897114.317)
S6[2,0]=complex(-4424633.127,-2161624.78)

aux14 = np.loadtxt('C7.txt', dtype='complex')
aux15 = aux14.reshape((1,n1,n1))

C7 = np.zeros((n1,n1),dtype=complex)
```

```
for i in range(n1):
    for j in range(n1):
        C7[i,j] = aux15[0][i][j]

S7=np.zeros((n1,1),dtype=complex)
S7[0,0]=complex(10000000,-7000000)
S7[1,0]=complex(1062177.834,12160254.04)
S7[2,0]=complex(-11062177.83,-5160254.039)


I1=np.zeros((n1,1),dtype=complex)
V1=np.zeros((n1,1),dtype=complex)
I2=np.zeros((n1,1),dtype=complex)
V2=np.zeros((n1,1),dtype=complex)
I3=np.zeros((n1,1),dtype=complex)
V3=np.zeros((n1,1),dtype=complex)
I4=np.zeros((n1,1),dtype=complex)
V4=np.zeros((n1,1),dtype=complex)
I5=np.zeros((n1,1),dtype=complex)
V5=np.zeros((n1,1),dtype=complex)
I6=np.zeros((n1,1),dtype=complex)
V6=np.zeros((n1,1),dtype=complex)
I7=np.zeros((n1,1),dtype=complex)
V7=np.zeros((n1,1),dtype=complex)

for i in range(15,18):
    I1[i-15,0]=S1[i-15,0]/(3*V8[i,0])
    V1[i-15,0]=V8[i,0]
H1=np.dot(C1,V1)
for j in range(15,18):
    I[i,0]=I1[i-15,0]-H1[i-15,0]


for i in range(21,24):
    I2[i-21,0]=S2[i-21,0]/(3*V8[i,0])
    V2[i-21,0]=V8[i,0]
H2=np.dot(C2,V2)
for j in range(21,24):
```

```
    I[i,0]=I2[i-21,0]-H2[i-21,0]


for i in range(27,29):
    I3[i-27,0]=S3[i-27,0]/(3*V8[i,0])
    V3[i-27,0]=V8[i,0]
H3=np.dot(C3,V3)
for j in range(27,29):
    I[i,0]=I3[i-27,0]-H3[i-27,0]


for i in range(33,36):
    I4[i-33,0]=S4[i-33,0]/(3*V8[i,0])
    V4[i-33,0]=V8[i,0]
H4=np.dot(C4,V4)
for j in range(33,36):
    I[i,0]=I4[i-33,0]-H4[i-33,0]


for i in range(39,42):
    I5[i-39,0]=S5[i-39,0]/(3*V8[i,0])
    V5[i-39,0]=V8[i,0]
H5=np.dot(C5,V5)
for j in range(39,42):
    I[i,0]=I5[i-39,0]-H5[i-39,0]


for i in range(45,48):
    I6[i-45,0]=S6[i-45,0]/(3*V8[i,0])
    V6[i-45,0]=V8[i,0]
H6=np.dot(C6,V6)
for j in range(45,48):
    I[i,0]=I6[i-45,0]-H6[i-45,0]


for i in range(51,54):
    I7[i-51,0]=S7[i-51,0]/(3*V8[i,0])
    V7[i-51,0]=V8[i,0]
H7=np.dot(C7,V7)
```

```
for j in range(51,54):
    I[i,0]=I7[i-51,0]-H7[i-51,0]


t = 1
T=10**-t
IterMax = 50

X = np.zeros((n,1),dtype=complex)
T1 = np.zeros((n,1), dtype=complex)
T2 = np.zeros((n,1), dtype=complex)

print('Jacobi-SKYLINE')
Iter=0
X=V8

cont1=0
while Iter<=IterMax:
    Iter=Iter+1
    V = np.zeros((n,1),dtype=complex)
    for k in range(n):
        d=int(RW[0,k+1]-RW[0,k])
        soma=0
        soma3=0
        d1=int(FCOL[0,k])
        for k1 in range(d):
            if d1+soma3 != k:
                soma=soma+(VAL[0,cont1]*X[d1+soma3,0])
                cont1=cont1+1
                soma3=soma3+1
            else:
                cont1=cont1+1
                soma3=soma3+1
                continue
        V[k,0]=(I[k,0]-soma)/T3[k,0]
        t1 = (V[k,0]-X[k,0])
        T1[k,0] = t1
        t2 = (V[k,0])
        T2[k,0] = t2
```

```
print()
print('Iteração =',Iter)
print()
print(V)
print()

t3 = max(abs(T1))
t4 = max(abs(T2))
t5=t3/t4
if t5<T:
    break
else:
    X=V
    for i in range(15,18):
        I1[i-15,0]=S1[i-15,0]/(3*X[i,0])
        V1[i-15,0]=X[i,0]
    H1=np.dot(C1,V1)
    for j in range(15,18):
        I[i,0]=I1[i-15,0]-H1[i-15,0]


    for i in range(21,24):
        I2[i-21,0]=S2[i-21,0]/(3*X[i,0])
        V2[i-21,0]=X[i,0]
    H2=np.dot(C2,V2)
    for j in range(21,24):
        I[i,0]=I2[i-21,0]-H2[i-21,0]


    for i in range(27,29):
        I3[i-27,0]=S3[i-27,0]/(3*X[i,0])
        V3[i-27,0]=X[i,0]
    H3=np.dot(C3,V3)
    for j in range(27,29):
        I[i,0]=I3[i-27,0]-H3[i-27,0]


    for i in range(33,36):
        I4[i-33,0]=S4[i-33,0]/(3*X[i,0])
```

```
            V4[i-33,0]=X[i,0]
        H4=np.dot(C4,V4)
        for j in range(33,36):
            I[i,0]=I4[i-33,0]-H4[i-33,0]


        for i in range(39,42):
            I5[i-39,0]=S5[i-39,0]/(3*X[i,0])
            V5[i-39,0]=X[i,0]
        H5=np.dot(C5,V5)
        for j in range(39,42):
            I[i,0]=I5[i-39,0]-H5[i-39,0]


        for i in range(45,48):
            I6[i-45,0]=S6[i-45,0]/(3*X[i,0])
            V6[i-45,0]=X[i,0]
        H6=np.dot(C6,V6)
        for j in range(45,48):
            I[i,0]=I6[i-45,0]-H6[i-45,0]


        for i in range(51,54):
            I7[i-51,0]=S7[i-51,0]/(3*X[i,0])
            V7[i-51,0]=X[i,0]
        H7=np.dot(C7,V7)
        for j in range(51,54):
            I[i,0]=I7[i-51,0]-H7[i-51,0]

    if cont1==990:
        cont1=0

print()
print('Matriz V')
print()
print(V)
print()
print('Iterações =',Iter)
```

```
fim = time.process_time()
print()
print('Tempo de Processamento %f' % (fim - inicio))
print()
print(psutil.virtual_memory())

sys.stdout.flush()
sys.exit()
```

A.4.2 Integrado ao Método de Gauss-Seidel

```
import numpy as np
import time
import psutil
import sys

sys.stdout=open("1skygs.txt","a")

inicio = time.process_time()

n=54

aux = np.loadtxt('inputys1.txt', dtype='complex')
aux = aux.reshape((1,n,n))

YS = np.zeros((n,n),dtype=complex)

for i in range(n):
    for j in range(n):
        YS[i,j] = aux[0][i][j]

T3 = np.zeros((n,1),dtype=complex)
for i in range(n):
    T3[i,0] = YS[i,i]


FCOL = np.zeros((1,n))
```

```
LCOL = np.zeros((1,n))
RW = np.zeros((1,n+1))
h=0
z=0

for i in range(n):
    for j in range(n):
        if YS[i,j] != 0:
            FCOL[0,i] = j
            break

for i in range(n):
    for j in range(n-1,1,-1):
        if YS[i,j] != 0:
            LCOL[0,i] = j
            break

for i in range(n):
    b = int(FCOL[0,i])
    c = int(LCOL[0,i])
    z = (c-b)+1
    h=h+z

VAL = np.zeros((1,h),dtype=complex)
r=0
for i in range(n):
    b = int(FCOL[0,i])
    c = int(LCOL[0,i])
    for j in range(b,c+1,1):
        VAL[0,r] = YS[i,j]
        r=r+1
    RW[0,i+1] = r


aux1 = np.loadtxt('inputi1.txt', dtype='complex')
aux1 = aux1.reshape((1,n,1))

I = np.zeros((n,1),dtype=complex)
```

```
for i in range(n):
    for j in range(1):
        I[i,j] = aux1[0][i][j]


aux2 = np.loadtxt('inputx1.txt', dtype='complex')
aux2 = aux2.reshape((1,n,1))

V = np.zeros((n,1),dtype=complex)

for i in range(n):
    for j in range(1):
        V[i,j] = aux2[0][i][j]

n1=3

aux2 = np.loadtxt('C1.txt', dtype='complex')
aux3 = aux2.reshape((1,n1,n1))

C1 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C1[i,j] = aux3[0][i][j]

S1=np.zeros((n1,1),dtype=complex)
S1[0,0]=complex(19000000,-13500000)
S1[1,0]=complex(2191342.961,23204482.67)
S1[2,0]=complex(-21191342.94,-9704482.688)

aux4 = np.loadtxt('C2.txt', dtype='complex')
aux5 = aux4.reshape((1,n1,n1))

C2 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C2[i,j] = aux5[0][i][j]
```

```
S2=np.zeros((n1,1),dtype=complex)
S2[0,0]=complex(10000000,-6500000)
S2[1,0]=complex(629165.115,11910254.04)
S2[2,0]=complex(-10629165.12,-5410254.047)

aux6 = np.loadtxt('C3.txt', dtype='complex')
aux7 = aux6.reshape((1,n1,n1))

C3 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C3[i,j] = aux7[0][i][j]

S3=np.zeros((n1,1),dtype=complex)
S3[0,0]=complex(15000000,-11000000)
S3[1,0]=complex(2026279.443,18490381.06)
S3[2,0]=complex(-17026279.44,-7490381.069)


aux8 = np.loadtxt('C4.txt', dtype='complex')
aux9 = aux8.reshape((1,n1,n1))

C4 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C4[i,j] = aux9[0][i][j]

S4=np.zeros((n1,1),dtype=complex)
S4[0,0]=complex(5000000,-3500000)
S4[1,0]=complex(531088.9134,6080127.019)
S4[2,0]=complex(-5531088.913,-2580127.019)


aux10 = np.loadtxt('C5.txt', dtype='complex')
aux11 = aux10.reshape((1,n1,n1))

C5 = np.zeros((n1,n1),dtype=complex)
```

```
for i in range(n1):
    for j in range(n1):
        C5[i,j] = aux11[0][i][j]

S5=np.zeros((n1,1),dtype=complex)
S5[0,0]=complex(5000000,-1500000)
S5[1,0]=complex(-1200961.897,5080127.018)
S5[2,0]=complex(-3799038.103,-3580127.021)

aux12 = np.loadtxt('C6.txt', dtype='complex')
aux13 = aux12.reshape((1,n1,n1))

C6 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C6[i,j] = aux13[0][i][j]

S6=np.zeros((n1,1),dtype=complex)
S6[0,0]=complex(4500000,-2000000)
S6[1,0]=complex(-517949.192,4897114.317)
S6[2,0]=complex(-4424633.127,-2161624.78)

aux14 = np.loadtxt('C7.txt', dtype='complex')
aux15 = aux14.reshape((1,n1,n1))

C7 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C7[i,j] = aux15[0][i][j]

S7=np.zeros((n1,1),dtype=complex)
S7[0,0]=complex(10000000,-7000000)
S7[1,0]=complex(1062177.834,12160254.04)
S7[2,0]=complex(-11062177.83,-5160254.039)
```

```
I1=np.zeros((n1,1),dtype=complex)
V1=np.zeros((n1,1),dtype=complex)
I2=np.zeros((n1,1),dtype=complex)
V2=np.zeros((n1,1),dtype=complex)
I3=np.zeros((n1,1),dtype=complex)
V3=np.zeros((n1,1),dtype=complex)
I4=np.zeros((n1,1),dtype=complex)
V4=np.zeros((n1,1),dtype=complex)
I5=np.zeros((n1,1),dtype=complex)
V5=np.zeros((n1,1),dtype=complex)
I6=np.zeros((n1,1),dtype=complex)
V6=np.zeros((n1,1),dtype=complex)
I7=np.zeros((n1,1),dtype=complex)
V7=np.zeros((n1,1),dtype=complex)

for i in range(15,18):
    I1[i-15,0]=S1[i-15,0]/(3*V[i,0])
    V1[i-15,0]=V[i,0]
H1=np.dot(C1,V1)
for j in range(15,18):
    I[i,0]=I1[i-15,0]-H1[i-15,0]


for i in range(21,24):
    I2[i-21,0]=S2[i-21,0]/(3*V[i,0])
    V2[i-21,0]=V[i,0]
H2=np.dot(C2,V2)
for j in range(21,24):
    I[i,0]=I2[i-21,0]-H2[i-21,0]


for i in range(27,29):
    I3[i-27,0]=S3[i-27,0]/(3*V[i,0])
    V3[i-27,0]=V[i,0]
H3=np.dot(C3,V3)
for j in range(27,29):
    I[i,0]=I3[i-27,0]-H3[i-27,0]
```

```
for i in range(33,36):
    I4[i-33,0]=S4[i-33,0]/(3*V[i,0])
    V4[i-33,0]=V[i,0]
H4=np.dot(C4,V4)
for j in range(33,36):
    I[i,0]=I4[i-33,0]-H4[i-33,0]


for i in range(39,42):
    I5[i-39,0]=S5[i-39,0]/(3*V[i,0])
    V5[i-39,0]=V[i,0]
H5=np.dot(C5,V5)
for j in range(39,42):
    I[i,0]=I5[i-39,0]-H5[i-39,0]


for i in range(45,48):
    I6[i-45,0]=S6[i-45,0]/(3*V[i,0])
    V6[i-45,0]=V[i,0]
H6=np.dot(C6,V6)
for j in range(45,48):
    I[i,0]=I6[i-45,0]-H6[i-45,0]


for i in range(51,54):
    I7[i-51,0]=S7[i-51,0]/(3*V[i,0])
    V7[i-51,0]=V[i,0]
H7=np.dot(C7,V7)
for j in range(51,54):
    I[i,0]=I7[i-51,0]-H7[i-51,0]


t = 1
T=10**-t
IterMax = 50


X = np.zeros((n,1),dtype=complex)
T1 = np.zeros((n,1), dtype=complex)
```

```
T2 = np.zeros((n,1), dtype=complex)

print('Gauss-Seidel - SKYLINE')
Iter=0
X=V

V8 = np.zeros((n,1), dtype=complex)

cont1=0
while Iter<IterMax:
    Iter=Iter+1
    for k in range(n):
        d=int(RW[0,k+1]-RW[0,k])
        soma=0
        soma1=0
        soma3=0
        d1=int(FCOL[0,k])
        for k1 in range(d):
            if (k > d1+soma3):
                soma=soma+(VAL[0,cont1]*V[d1+soma3,0])
            elif (k < d1+soma3):
                soma1=soma1+(VAL[0,cont1]*X[d1+soma3,0])
            cont1=cont1+1
            soma3=soma3+1
        V8[k,0]=(1/T3[k,0])*(I[k,0]-soma1-soma)
        t1 = (V8[k,0]-X[k,0])
        T1[k,0] = t1
        t2 = (V8[k,0])
        T2[k,0] = t2
    print()
    print('Iteração =',Iter)
    print()
    print(V8)
    print()

    t3 = max(abs(T1))
    t4 = max(abs(T2))
    t5=t3/t4
    if t5<T:
```

```
        break
    else:
        V=X
        X=V8
        for i in range(15,18):
            I1[i-15,0]=S1[i-15,0]/(3*V8[i,0])
            V1[i-15,0]=V8[i,0]
        H1=np.dot(C1,V1)
        for j in range(15,18):
            I[i,0]=I1[i-15,0]-H1[i-15,0]


        for i in range(21,24):
            I2[i-21,0]=S2[i-21,0]/(3*V8[i,0])
            V2[i-21,0]=V8[i,0]
        H2=np.dot(C2,V2)
        for j in range(21,24):
            I[i,0]=I2[i-21,0]-H2[i-21,0]


        for i in range(27,29):
            I3[i-27,0]=S3[i-27,0]/(3*V8[i,0])
            V3[i-27,0]=V8[i,0]
        H3=np.dot(C3,V3)
        for j in range(27,29):
            I[i,0]=I3[i-27,0]-H3[i-27,0]


        for i in range(33,36):
            I4[i-33,0]=S4[i-33,0]/(3*V8[i,0])
            V4[i-33,0]=V8[i,0]
        H4=np.dot(C4,V4)
        for j in range(33,36):
            I[i,0]=I4[i-33,0]-H4[i-33,0]


        for i in range(39,42):
            I5[i-39,0]=S5[i-39,0]/(3*V8[i,0])
            V5[i-39,0]=V8[i,0]
```

```
            H5=np.dot(C5,V5)
            for j in range(39,42):
                I[i,0]=I5[i-39,0]-H5[i-39,0]


            for i in range(45,48):
                I6[i-45,0]=S6[i-45,0]/(3*V8[i,0])
                V6[i-45,0]=V8[i,0]
            H6=np.dot(C6,V6)
            for j in range(45,48):
                I[i,0]=I6[i-45,0]-H6[i-45,0]


            for i in range(51,54):
                I7[i-51,0]=S7[i-51,0]/(3*V8[i,0])
                V7[i-51,0]=V8[i,0]
            H7=np.dot(C7,V7)
            for j in range(51,54):
                I[i,0]=I7[i-51,0]-H7[i-51,0]

        if cont1==990:
            cont1=0


print()
print('Matriz V')
print()
print(V8)
print()
print('Iterações =',Iter)


fim = time.process_time()
print()
print('Tempo de Processamento %f' % (fim - inicio))
print()
print(psutil.virtual_memory())

sys.stdout.flush()
```

```
sys.exit()
```

A.5 Método DFA2

A.5.1 Integrado ao Método de Jacobi

```
import numpy as np
import time
import psutil
import sys

sys.stdout=open("1dfa2jac.txt","a")

inicio = time.process_time()

n=54

aux = np.loadtxt('inputys1.txt', dtype='complex')
aux = aux.reshape((1,n,n))

YS = np.zeros((n,n),dtype=complex)

for i in range(n):
    for j in range(n):
        YS[i,j] = aux[0][i][j]

T3 = np.zeros((n,1),dtype=complex)
for i in range(n):
    T3[i,0] = YS[i,i]

soma1=0
for i in range(n):
    for j in range(n):
        if YS[i,j] != 0:
           soma1=soma1+1

fat=soma1/n
```

```
fat1=round(fat)

q1=n*fat1

a3=int(q1)
VAL1 = np.zeros((1,a3),dtype=complex)
soma2=0
for i in range(n):
    for j in range(n):
        if YS[i,j] != 0:
           VAL1[0,soma2] = YS[i,j]
           soma2=soma2+1

ax1 = VAL1.reshape((1,n,9))
VAL = np.zeros((n,9),dtype=complex)
for i in range(n):
    for j in range(9):
        VAL[i,j] = ax1[0][i][j]


ROWID = np.zeros((1,n))
INDEX = np.zeros((1,n))
OFFSET = np.zeros((1,n))
NZ = np.zeros((1,n))

JA=np.zeros((1,soma1))
z=0
h1=0
for i in range(n):
    h=0
    for j in range(n):
        if YS[i,j] != 0:
            h = h + 1
            JA[0,h1] = j
            h1=h1+1
    NZ[0,i]=h
    ROWID[0,i] = z
    z=z+1
```

```
INDEX = np.zeros((1,n))
OFFSET = np.zeros((1,n))

INDEX[0,0] = 0
OFFSET[0,0] = 0
c1=0
d=0
for i in range(1,n):
    c=int(NZ[0,i-1])
    c1=c1+c
    OFFSET[0,i] = c1%fat1
    if c>1:
        INDEX[0,i] = d+1
        d=d+1
    else:
        INDEX[0,i] = INDEX[0,i-1]

aux1 = np.loadtxt('inputi1.txt', dtype='complex')
aux1 = aux1.reshape((1,n,1))

I = np.zeros((n,1),dtype=complex)

for i in range(n):
    for j in range(1):
        I[i,j] = aux1[0][i][j]


aux2 = np.loadtxt('inputx1.txt', dtype='complex')
aux2 = aux2.reshape((1,n,1))

V = np.zeros((n,1),dtype=complex)

for i in range(n):
    for j in range(1):
        V[i,j] = aux2[0][i][j]

n1=3
```

```
aux2 = np.loadtxt('C1.txt', dtype='complex')
aux3 = aux2.reshape((1,n1,n1))

C1 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C1[i,j] = aux3[0][i][j]

S1=np.zeros((n1,1),dtype=complex)
S1[0,0]=complex(19000000,-13500000)
S1[1,0]=complex(2191342.961,23204482.67)
S1[2,0]=complex(-21191342.94,-9704482.688)

aux4 = np.loadtxt('C2.txt', dtype='complex')
aux5 = aux4.reshape((1,n1,n1))

C2 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C2[i,j] = aux5[0][i][j]

S2=np.zeros((n1,1),dtype=complex)
S2[0,0]=complex(10000000,-6500000)
S2[1,0]=complex(629165.115,11910254.04)
S2[2,0]=complex(-10629165.12,-5410254.047)

aux6 = np.loadtxt('C3.txt', dtype='complex')
aux7 = aux6.reshape((1,n1,n1))

C3 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C3[i,j] = aux7[0][i][j]

S3=np.zeros((n1,1),dtype=complex)
S3[0,0]=complex(15000000,-11000000)
```

```
S3[1,0]=complex(2026279.443,18490381.06)
S3[2,0]=complex(-17026279.44,-7490381.069)


aux8 = np.loadtxt('C4.txt', dtype='complex')
aux9 = aux8.reshape((1,n1,n1))

C4 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C4[i,j] = aux9[0][i][j]

S4=np.zeros((n1,1),dtype=complex)
S4[0,0]=complex(5000000,-3500000)
S4[1,0]=complex(531088.9134,6080127.019)
S4[2,0]=complex(-5531088.913,-2580127.019)


aux10 = np.loadtxt('C5.txt', dtype='complex')
aux11 = aux10.reshape((1,n1,n1))

C5 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C5[i,j] = aux11[0][i][j]

S5=np.zeros((n1,1),dtype=complex)
S5[0,0]=complex(5000000,-1500000)
S5[1,0]=complex(-1200961.897,5080127.018)
S5[2,0]=complex(-3799038.103,-3580127.021)

aux12 = np.loadtxt('C6.txt', dtype='complex')
aux13 = aux12.reshape((1,n1,n1))

C6 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
```

```
    for j in range(n1):
        C6[i,j] = aux13[0][i][j]

S6=np.zeros((n1,1),dtype=complex)
S6[0,0]=complex(4500000,-2000000)
S6[1,0]=complex(-517949.192,4897114.317)
S6[2,0]=complex(-4424633.127,-2161624.78)

aux14 = np.loadtxt('C7.txt', dtype='complex')
aux15 = aux14.reshape((1,n1,n1))

C7 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C7[i,j] = aux15[0][i][j]

S7=np.zeros((n1,1),dtype=complex)
S7[0,0]=complex(10000000,-7000000)
S7[1,0]=complex(1062177.834,12160254.04)
S7[2,0]=complex(-11062177.83,-5160254.039)


I1=np.zeros((n1,1),dtype=complex)
V1=np.zeros((n1,1),dtype=complex)
I2=np.zeros((n1,1),dtype=complex)
V2=np.zeros((n1,1),dtype=complex)
I3=np.zeros((n1,1),dtype=complex)
V3=np.zeros((n1,1),dtype=complex)
I4=np.zeros((n1,1),dtype=complex)
V4=np.zeros((n1,1),dtype=complex)
I5=np.zeros((n1,1),dtype=complex)
V5=np.zeros((n1,1),dtype=complex)
I6=np.zeros((n1,1),dtype=complex)
V6=np.zeros((n1,1),dtype=complex)
I7=np.zeros((n1,1),dtype=complex)
V7=np.zeros((n1,1),dtype=complex)

for i in range(15,18):
```

```
    I1[i-15,0]=S1[i-15,0]/(3*V[i,0])
    V1[i-15,0]=V[i,0]
H1=np.dot(C1,V1)
for j in range(15,18):
    I[i,0]=I1[i-15,0]-H1[i-15,0]


for i in range(21,24):
    I2[i-21,0]=S2[i-21,0]/(3*V[i,0])
    V2[i-21,0]=V[i,0]
H2=np.dot(C2,V2)
for j in range(21,24):
    I[i,0]=I2[i-21,0]-H2[i-21,0]


for i in range(27,29):
    I3[i-27,0]=S3[i-27,0]/(3*V[i,0])
    V3[i-27,0]=V[i,0]
H3=np.dot(C3,V3)
for j in range(27,29):
    I[i,0]=I3[i-27,0]-H3[i-27,0]


for i in range(33,36):
    I4[i-33,0]=S4[i-33,0]/(3*V[i,0])
    V4[i-33,0]=V[i,0]
H4=np.dot(C4,V4)
for j in range(33,36):
    I[i,0]=I4[i-33,0]-H4[i-33,0]


for i in range(39,42):
    I5[i-39,0]=S5[i-39,0]/(3*V[i,0])
    V5[i-39,0]=V[i,0]
H5=np.dot(C5,V5)
for j in range(39,42):
    I[i,0]=I5[i-39,0]-H5[i-39,0]
```

```
for i in range(45,48):
    I6[i-45,0]=S6[i-45,0]/(3*V[i,0])
    V6[i-45,0]=V[i,0]
H6=np.dot(C6,V6)
for j in range(45,48):
    I[i,0]=I6[i-45,0]-H6[i-45,0]


for i in range(51,54):
    I7[i-51,0]=S7[i-51,0]/(3*V[i,0])
    V7[i-51,0]=V[i,0]
H7=np.dot(C7,V7)
for j in range(51,54):
    I[i,0]=I7[i-51,0]-H7[i-51,0]


t = 1
T=10**-t
IterMax = 50


VAL2=np.zeros((1,468),dtype=complex)
cont=0
for i in range(n):
    for j in range(9):
        if VAL[i,j] != 0:
            VAL2[0,cont] = VAL[i,j]
            cont=cont+1


X = np.zeros((n,1),dtype=complex)
T1 = np.zeros((n,1), dtype=complex)
T2 = np.zeros((n,1), dtype=complex)

print('Jacobi-DFA2')
Iter=0
X=V

cont1=0
```

```
while Iter<=IterMax:
    Iter=Iter+1
    V = np.zeros((n,1),dtype=complex)
    for k in range(n):
        d=int(NZ[0,k])
        soma=0
        for k1 in range(d):
            d1=int(JA[0,cont1])
            if k != d1:
                soma=soma+(VAL2[0,cont1]*X[d1,0])
                cont1=cont1+1
            else:
                cont1=cont1+1
                continue
        V[k,0]=(I[k,0]-soma)/T3[k,0]
        t1 = (V[k,0]-X[k,0])
        T1[k,0] = t1
        t2 = (V[k,0])
        T2[k,0] = t2
    print()
    print('Iteração =',Iter)
    print()
    print(V)
    print()

    t3 = max(abs(T1))
    t4 = max(abs(T2))
    t5=t3/t4
    if t5<T:
        break
    else:
        X=V
        for i in range(15,18):
            I1[i-15,0]=S1[i-15,0]/(3*X[i,0])
            V1[i-15,0]=X[i,0]
        H1=np.dot(C1,V1)
        for j in range(15,18):
            I[i,0]=I1[i-15,0]-H1[i-15,0]
```

```
for i in range(21,24):
    I2[i-21,0]=S2[i-21,0]/(3*X[i,0])
    V2[i-21,0]=X[i,0]
H2=np.dot(C2,V2)
for j in range(21,24):
    I[i,0]=I2[i-21,0]-H2[i-21,0]


for i in range(27,29):
    I3[i-27,0]=S3[i-27,0]/(3*X[i,0])
    V3[i-27,0]=X[i,0]
H3=np.dot(C3,V3)
for j in range(27,29):
    I[i,0]=I3[i-27,0]-H3[i-27,0]


for i in range(33,36):
    I4[i-33,0]=S4[i-33,0]/(3*X[i,0])
    V4[i-33,0]=X[i,0]
H4=np.dot(C4,V4)
for j in range(33,36):
    I[i,0]=I4[i-33,0]-H4[i-33,0]


for i in range(39,42):
    I5[i-39,0]=S5[i-39,0]/(3*X[i,0])
    V5[i-39,0]=X[i,0]
H5=np.dot(C5,V5)
for j in range(39,42):
    I[i,0]=I5[i-39,0]-H5[i-39,0]


for i in range(45,48):
    I6[i-45,0]=S6[i-45,0]/(3*X[i,0])
    V6[i-45,0]=X[i,0]
H6=np.dot(C6,V6)
for j in range(45,48):
    I[i,0]=I6[i-45,0]-H6[i-45,0]
```

```
        for i in range(51,54):
            I7[i-51,0]=S7[i-51,0]/(3*X[i,0])
            V7[i-51,0]=X[i,0]
        H7=np.dot(C7,V7)
        for j in range(51,54):
            I[i,0]=I7[i-51,0]-H7[i-51,0]

    if cont1==soma1:
        cont1=0


print()
print('Matriz V')
print()
print(V)
print()
print('Iterações =',Iter)


fim = time.process_time()
print()
print('Tempo de Processamento %f' % (fim - inicio))
print()
print(psutil.virtual_memory())


sys.stdout.flush()
sys.exit()
```

A.5.2 Integrado ao Método de Gauss-Seidel

```
import numpy as np
import time
import psutil
import sys
```

```
sys.stdout=open("1dfa2gs.txt","a")

inicio = time.process_time()

n=54

aux = np.loadtxt('inputys1.txt', dtype='complex')
aux = aux.reshape((1,n,n))

YS = np.zeros((n,n),dtype=complex)

for i in range(n):
    for j in range(n):
        YS[i,j] = aux[0][i][j]

T3 = np.zeros((n,1),dtype=complex)
for i in range(n):
    T3[i,0] = YS[i,i]

soma11=0
for i in range(n):
    for j in range(n):
        if YS[i,j] != 0:
           soma11=soma11+1

fat=soma11/n
fat1=round(fat)

q1=n*fat1

a3=int(q1)
VAL1 = np.zeros((1,a3),dtype=complex)
soma2=0
for i in range(n):
    for j in range(n):
        if YS[i,j] != 0:
           VAL1[0,soma2] = YS[i,j]
           soma2=soma2+1
```

```
ax1 = VAL1.reshape((1,n,9))
VAL = np.zeros((n,9),dtype=complex)
for i in range(n):
    for j in range(9):
        VAL[i,j] = ax1[0][i][j]


NZ = np.zeros((1,n))
JA=np.zeros((1,soma11))
z=0
h1=0
for i in range(n):
    h=0
    for j in range(n):
        if YS[i,j] != 0:
            h = h + 1
            JA[0,h1] = j
            h1=h1+1
    NZ[0,i]=h

INDEX = np.zeros((1,n))
OFFSET = np.zeros((1,n))

INDEX[0,0] = 0
OFFSET[0,0] = 0
c1=0
d=0
for i in range(1,n):
    c=int(NZ[0,i-1])
    c1=c1+c
    OFFSET[0,i] = c1%fat1
    if c>1:
        INDEX[0,i] = d+1
        d=d+1
    else:
        INDEX[0,i] = INDEX[0,i-1]

aux1 = np.loadtxt('inputi1.txt', dtype='complex')
```

```
aux1 = aux1.reshape((1,n,1))

I = np.zeros((n,1),dtype=complex)

for i in range(n):
    for j in range(1):
        I[i,j] = aux1[0][i][j]


aux2 = np.loadtxt('inputx1.txt', dtype='complex')
aux2 = aux2.reshape((1,n,1))

V = np.zeros((n,1),dtype=complex)

for i in range(n):
    for j in range(1):
        V[i,j] = aux2[0][i][j]

n1=3

aux2 = np.loadtxt('C1.txt', dtype='complex')
aux3 = aux2.reshape((1,n1,n1))

C1 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C1[i,j] = aux3[0][i][j]

S1=np.zeros((n1,1),dtype=complex)
S1[0,0]=complex(19000000,-13500000)
S1[1,0]=complex(2191342.961,23204482.67)
S1[2,0]=complex(-21191342.94,-9704482.688)

aux4 = np.loadtxt('C2.txt', dtype='complex')
aux5 = aux4.reshape((1,n1,n1))

C2 = np.zeros((n1,n1),dtype=complex)
```

```
for i in range(n1):
    for j in range(n1):
        C2[i,j] = aux5[0][i][j]

S2=np.zeros((n1,1),dtype=complex)
S2[0,0]=complex(10000000,-6500000)
S2[1,0]=complex(629165.115,11910254.04)
S2[2,0]=complex(-10629165.12,-5410254.047)

aux6 = np.loadtxt('C3.txt', dtype='complex')
aux7 = aux6.reshape((1,n1,n1))

C3 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C3[i,j] = aux7[0][i][j]

S3=np.zeros((n1,1),dtype=complex)
S3[0,0]=complex(15000000,-11000000)
S3[1,0]=complex(2026279.443,18490381.06)
S3[2,0]=complex(-17026279.44,-7490381.069)


aux8 = np.loadtxt('C4.txt', dtype='complex')
aux9 = aux8.reshape((1,n1,n1))

C4 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C4[i,j] = aux9[0][i][j]

S4=np.zeros((n1,1),dtype=complex)
S4[0,0]=complex(5000000,-3500000)
S4[1,0]=complex(531088.9134,6080127.019)
S4[2,0]=complex(-5531088.913,-2580127.019)
```

```
aux10 = np.loadtxt('C5.txt', dtype='complex')
aux11 = aux10.reshape((1,n1,n1))

C5 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C5[i,j] = aux11[0][i][j]

S5=np.zeros((n1,1),dtype=complex)
S5[0,0]=complex(5000000,-1500000)
S5[1,0]=complex(-1200961.897,5080127.018)
S5[2,0]=complex(-3799038.103,-3580127.021)

aux12 = np.loadtxt('C6.txt', dtype='complex')
aux13 = aux12.reshape((1,n1,n1))

C6 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C6[i,j] = aux13[0][i][j]

S6=np.zeros((n1,1),dtype=complex)
S6[0,0]=complex(4500000,-2000000)
S6[1,0]=complex(-517949.192,4897114.317)
S6[2,0]=complex(-4424633.127,-2161624.78)

aux14 = np.loadtxt('C7.txt', dtype='complex')
aux15 = aux14.reshape((1,n1,n1))

C7 = np.zeros((n1,n1),dtype=complex)

for i in range(n1):
    for j in range(n1):
        C7[i,j] = aux15[0][i][j]

S7=np.zeros((n1,1),dtype=complex)
S7[0,0]=complex(10000000,-7000000)
```

```
S7[1,0]=complex(1062177.834,12160254.04)
S7[2,0]=complex(-11062177.83,-5160254.039)


I1=np.zeros((n1,1),dtype=complex)
V1=np.zeros((n1,1),dtype=complex)
I2=np.zeros((n1,1),dtype=complex)
V2=np.zeros((n1,1),dtype=complex)
I3=np.zeros((n1,1),dtype=complex)
V3=np.zeros((n1,1),dtype=complex)
I4=np.zeros((n1,1),dtype=complex)
V4=np.zeros((n1,1),dtype=complex)
I5=np.zeros((n1,1),dtype=complex)
V5=np.zeros((n1,1),dtype=complex)
I6=np.zeros((n1,1),dtype=complex)
V6=np.zeros((n1,1),dtype=complex)
I7=np.zeros((n1,1),dtype=complex)
V7=np.zeros((n1,1),dtype=complex)

for i in range(15,18):
    I1[i-15,0]=S1[i-15,0]/(3*V[i,0])
    V1[i-15,0]=V[i,0]
H1=np.dot(C1,V1)
for j in range(15,18):
    I[i,0]=I1[i-15,0]-H1[i-15,0]


for i in range(21,24):
    I2[i-21,0]=S2[i-21,0]/(3*V[i,0])
    V2[i-21,0]=V[i,0]
H2=np.dot(C2,V2)
for j in range(21,24):
    I[i,0]=I2[i-21,0]-H2[i-21,0]


for i in range(27,29):
    I3[i-27,0]=S3[i-27,0]/(3*V[i,0])
    V3[i-27,0]=V[i,0]
H3=np.dot(C3,V3)
```

```
for j in range(27,29):
    I[i,0]=I3[i-27,0]-H3[i-27,0]


for i in range(33,36):
    I4[i-33,0]=S4[i-33,0]/(3*V[i,0])
    V4[i-33,0]=V[i,0]
H4=np.dot(C4,V4)
for j in range(33,36):
    I[i,0]=I4[i-33,0]-H4[i-33,0]


for i in range(39,42):
    I5[i-39,0]=S5[i-39,0]/(3*V[i,0])
    V5[i-39,0]=V[i,0]
H5=np.dot(C5,V5)
for j in range(39,42):
    I[i,0]=I5[i-39,0]-H5[i-39,0]


for i in range(45,48):
    I6[i-45,0]=S6[i-45,0]/(3*V[i,0])
    V6[i-45,0]=V[i,0]
H6=np.dot(C6,V6)
for j in range(45,48):
    I[i,0]=I6[i-45,0]-H6[i-45,0]


for i in range(51,54):
    I7[i-51,0]=S7[i-51,0]/(3*V[i,0])
    V7[i-51,0]=V[i,0]
H7=np.dot(C7,V7)
for j in range(51,54):
    I[i,0]=I7[i-51,0]-H7[i-51,0]


t = 1
T=10**-t
IterMax = 50
```

```
VAL2=np.zeros((1,468),dtype=complex)
cont=0
for i in range(n):
    for j in range(9):
        if VAL[i,j] != 0:
            VAL2[0,cont] = VAL[i,j]
            cont=cont+1

V8 = np.zeros((n,1), dtype=complex)
X = np.zeros((n,1),dtype=complex)
T1 = np.zeros((n,1), dtype=complex)
T2 = np.zeros((n,1), dtype=complex)

print('Gauss-Seidel - DFA2')
Iter=0
X=V

cont1=0
while Iter<=IterMax:
    Iter=Iter+1
    for k in range(n):
        d=int(NZ[0,k])
        soma=0
        soma1=0
        for k1 in range(d):
            d1=int(JA[0,cont1])
            if (k > d1):
                soma=soma+(VAL2[0,cont1]*V[d1,0])
            elif (k < d1):
                soma1=soma1+(VAL2[0,cont1]*X[d1,0])
            cont1=cont1+1
            continue
        V8[k,0]=(I[k,0]-soma1-soma)/T3[k,0]
        t1 = (V8[k,0]-X[k,0])
        T1[k,0] = t1
        t2 = (V8[k,0])
        T2[k,0] = t2
```

```
print()
print('Iteração =',Iter)
print()
print(V8)
print()

t3 = max(abs(T1))
t4 = max(abs(T2))
t5=t3/t4
if t5<T:
    break
else:
    V=X
    X=V8
    for i in range(15,18):
        I1[i-15,0]=S1[i-15,0]/(3*V8[i,0])
        V1[i-15,0]=V8[i,0]
    H1=np.dot(C1,V1)
    for j in range(15,18):
        I[i,0]=I1[i-15,0]-H1[i-15,0]


    for i in range(21,24):
        I2[i-21,0]=S2[i-21,0]/(3*V8[i,0])
        V2[i-21,0]=V8[i,0]
    H2=np.dot(C2,V2)
    for j in range(21,24):
        I[i,0]=I2[i-21,0]-H2[i-21,0]


    for i in range(27,29):
        I3[i-27,0]=S3[i-27,0]/(3*V8[i,0])
        V3[i-27,0]=V8[i,0]
    H3=np.dot(C3,V3)
    for j in range(27,29):
        I[i,0]=I3[i-27,0]-H3[i-27,0]


    for i in range(33,36):
```

```
            I4[i-33,0]=S4[i-33,0]/(3*V8[i,0])
            V4[i-33,0]=V8[i,0]
        H4=np.dot(C4,V4)
        for j in range(33,36):
            I[i,0]=I4[i-33,0]-H4[i-33,0]


        for i in range(39,42):
            I5[i-39,0]=S5[i-39,0]/(3*V8[i,0])
            V5[i-39,0]=V8[i,0]
        H5=np.dot(C5,V5)
        for j in range(39,42):
            I[i,0]=I5[i-39,0]-H5[i-39,0]


        for i in range(45,48):
            I6[i-45,0]=S6[i-45,0]/(3*V8[i,0])
            V6[i-45,0]=V8[i,0]
        H6=np.dot(C6,V6)
        for j in range(45,48):
            I[i,0]=I6[i-45,0]-H6[i-45,0]


        for i in range(51,54):
            I7[i-51,0]=S7[i-51,0]/(3*V8[i,0])
            V7[i-51,0]=V8[i,0]
        H7=np.dot(C7,V7)
        for j in range(51,54):
            I[i,0]=I7[i-51,0]-H7[i-51,0]

    if cont1==soma11:
        cont1=0


print()
print('Matriz V')
print()
print(V8)
print()
```

```
print('Iterações =',Iter)


fim = time.process_time()
print()
print('Tempo de Processamento %f' % (fim - inicio))
print()
print(psutil.virtual_memory())


sys.stdout.flush()
sys.exit()
```

Apêndice B

Tempos Aferidos

B.1 Tempos Aferidos - Sistema Equilibrado (Sem GD)

Tabela B.1: Tempos Aferidos - Sistema Equilibrado (Sem GD)

Tempos Aferidos (s)											
Execução	Métodos										
	DSS	Método 1	Método 2	Método 3	Método 4	Método 5	Método 6	Método 7	Método 8	Método 9	Método 10
1	0,171875	0,03125	0,056393	0,03125	0,055851	0,03125	0,056393	0,0625	0,046875	0,0625	0,0625
2	0,015625	0,078125	0,034379	0,09375	0,027639	0,046875	0,034379	0,078125	0,0625	0,09375	0,046875
3	0,03125	0,09375	0,023436	0,09375	0,059145	0,046875	0,083402	0,0625	0,0625	0,078125	0,078125
4	0,03125	0,046875	0,024265	0,09375	0,065987	0,039226	0,0625	0,09375	0,078125	0,09375	0,078125
5	0,015625	0,0625	0,025258	0,09375	0,079951	0,30555	0,0625	0,046875	0,078125	0,09375	0,078125
6	0,046875	0,078125	0,05816	0,09375	0,065226	0,046875	0,0625	0,0625	0,0625	0,078125	0,015625
7	0,015625	0,078125	0,024234	0,09375	0,021708	0,03125	0,0625	0,03125	0,03125	0,0625	0,03125
8	0,078125	0,109375	0,057583	0,09375	0,073876	0,078125	0,0625	0,0625	0,078125	0,078125	0,03125
9	0,015625	0,078125	0,021582	0,046875	0,077766	0,03125	0,07462	0,09375	0,0625	0,0625	0,03125
10	0,078125	0,09375	0,024036	0,109375	0,024725	0,0625	0,077125	0,046875	0,078125	0,09375	0,0625
11	0,0625	0,109375	0,079379	0,109375	0,024332	0,046875	0,073546	0,046875	0,03125	0,046875	0,078125
12	0,0625	0,0625	0,080812	0,109375	0,073546	0,09375	0,058405	0,0625	0,0625	0,078125	0,0625
13	0,03125	0,0625	0,021556	0,0625	0,058405	0,09375	0,02494	0,0625	0,078125	0,0625	0,0625
14	0,0625	0,03125	0,083592	0,109375	0,023642	0,0625	0,034626	0,078125	0,078125	0,09375	0,03125
15	0,015625	0,078125	0,09162	0,09375	0,024066	0,0625	0,070015	0,0625	0,078125	0,09375	0,078125
16	0,109375	0,03125	0,081091	0,0625	0,058325	0,046875	0,058325	0,109375	0,078125	0,0625	0,078125
17	0,03125	0,0625	0,078738	0,109375	0,073159	0,09375	0,073159	0,09375	0,078125	0,078125	0,0625
18	0,0625	0,0625	0,061141	0,046875	0,075058	0,09375	0,084694	0,09375	0,03125	0,078125	0,09375
19	0,0625	0,015625	0,076132	0,109375	0,027524	0,078125	0,085333	0,078125	0,0625	0,078125	0,046875
20	0,03125	0,0625	0,070014	0,109375	0,059859	0,046875	0,059859	0,09375	0,0625	0,09375	0,09375
21	0,046875	0,0625	0,059158	0,09375	0,024411	0,09375	0,079512	0,078125	0,046875	0,09375	0,09375
22	0,125	0,0625	0,071934	0,078125	0,074473	0,078125	0,074473	0,078125	0,078125	0,09375	0,078125
23	0,0625	0,078125	0,024994	0,09375	0,022259	0,057583	0,024715	0,0625	0,078125	0,03125	0,046875
24	0,109375	0,03125	0,022402	0,0625	0,024715	0,095739	0,025558	0,078125	0,078125	0,078125	0,09375
25	0,078125	0,015625	0,058776	0,046875	0,025558	0,088983	0,078125	0,09375	0,0625	0,03125	0,078125
26	0,09375	0,0625	0,024793	0,078125	0,034626	0,080812	0,09375	0,109375	0,078125	0,046875	0,0625
27	0,09375	0,0625	0,024793	0,078125	0,034626	0,080812	0,09375	0,109375	0,078125	0,046875	0,0625
28	0,125	0,0625	0,054939	0,078125	0,070015	0,092405	0,078125	0,09375	0,046875	0,078125	0,078125
29	0,09375	0,0625	0,026064	0,0625	0,024252	0,079512	0,091176	0,09375	0,078125	0,078125	0,078125
30	0,0625	0,0625	0,035139	0,015625	0,02424	0,074473	0,09375	0,09375	0,046875	0,078125	0,078125

Tabela B.2: Tempos Aferidos - Sistema Equilibrado (Sem GD)

Tempos Aferidos (s)											
Execução	Métodos										
	DSS	Método 1	Método 2	Método 3	Método 4	Método 5	Método 6	Método 7	Método 8	Método 9	Método 10
31	0,140625	0,0625	0,022588	0,015625	0,02313	0,015625	0,015625	0,03125	0,015625	0,015625	0,015625
32	0,03125	0,015625	0,024025	0,03125	0,022864	0,03125	0,015625	0,03125	0,015625	0,015625	0,015625
33	0,03125	0,03125	0,026182	0,015625	0,02418	0,03125	0,015625	0,03125	0,03125	0,03125	0,03125
34	0,03125	0,03125	0,02398	0,015625	0,024776	0,015625	0,03125	0,03125	0,015625	0,015625	0,015625
35	0,015625	0,046875	0,023496	0,046875	0,023707	0,03125	0,015625	0,046875	0,015625	0,03125	0,015625
36	0,015625	0,015625	0,025727	0,03125	0,02423	0,03125	0,03125	0,03125	0,015625	0,03125	0,015625
37	0,078125	0,015625	0,02531	0,03125	0,024039	0,03125	0,015625	0,046875	0,03125	0,03125	0,03125
38	0,0625	0,015625	0,024882	0,03125	0,022563	0,03125	0,046875	0,015625	0,03125	0,03125	0,03125
39	0,03125	0,015625	0,023588	0,03125	0,023324	0,015625	0,046875	0,03125	0,015625	0,046875	0,015625
40	0,015625	0,03125	0,023861	0,03125	0,026811	0,046875	0,015625	0,046875	0,03125	0,03125	0,03125
41	0,109375	0,015625	0,020525	0,015625	0,021515	0,015625	0,03125	0,0625	0,03125	0,015625	0,03125
42	0,015625	0,03125	0,021436	0,03125	0,021466	0,015625	0,015625	0,03125	0,015625	0,03125	0,015625
43	0,046875	0,03125	0,021232	0,03125	0,029229	0,03125	0,03125	0,015625	0,015625	0,03125	0,015625
44	0,09375	0,015625	0,024163	0,03125	0,024643	0,03125	0,015625	0,03125	0,015625	0,03125	0,03125
45	0,078125	0,03125	0,023916	0,03125	0,02356	0,03125	0,015625	0,046875	0,015625	0,03125	0,015625
46	0,09375	0,046875	0,023329	0,03125	0,021874	0,03125	0,03125	0,03125	0,015625	0,03125	0,015625
47	0,015625	0,03125	0,023884	0,015625	0,022542	0,015625	0,015625	0,03125	0,03125	0,015625	0,015625
48	0,03125	0,015625	0,022009	0,03125	0,022373	0,03125	0,015625	0,03125	0,03125	0,03125	0,015625
49	0,09375	0,03125	0,023893	0,046875	0,027711	0,03125	0,015625	0,03125	0,03125	0,03125	0,015625
50	0,09375	0,015625	0,026392	0,03125	0,023344	0,046875	0,015625	0,046875	0,015625	0,03125	0,03125
51	0,03125	0,03125	0,022611	0,015625	0,023579	0,03125	0,015625	0,03125	0,015625	0,03125	0,03125
52	0,109375	0,015625	0,024393	0,03125	0,023667	0,03125	0,015625	0,046875	0,015625	0,03125	0,03125
53	0,078125	0,03125	0,023102	0,015625	0,024505	0,03125	0,015625	0,015625	0,03125	0,03125	0,015625
54	0,0625	0,03125	0,028224	0,015625	0,023194	0,015625	0,015625	0,03125	0,03125	0,03125	0,015625
55	0,125	0,046875	0,022498	0,015625	0,022337	0,03125	0,015625	0,046875	0,03125	0,03125	0,015625
56	0,09375	0,03125	0,022623	0,015625	0,023737	0,03125	0,015625	0,0625	0,015625	0,03125	0,03125
57	0,140625	0,03125	0,023921	0,015625	0,021783	0,015625	0,015625	0,03125	0,015625	0,03125	0,015625
57	0,0625	0,03125	0,025144	0,015625	0,023899	0,03125	0,015625	0,015625	0,015625	0,015625	0,015625
59	0,078125	0,03125	0,020889	0,03125	0,026218	0,03125	0,015625	0,03125	0,03125	0,03125	0,015625
60	0,078125	0,046875	0,027162	0,03125	0,023974	0,046875	0,015625	0,046875	0,015625	0,03125	0,03125

B.2 Tempos Aferidos - Sistema Equilibrado (Com GD)

Tabela B.3: Tempos Aferidos - Sistema Equilibrado (Com GD)

Tempos Aferidos (s)											
Execução	Métodos										
	DSS	Método 1	Método 2	Método 3	Método 4	Método 5	Método 6	Método 7	Método 8	Método 9	Método 10
1	0,0625	0,03125	0,060299	0,046875	0,078079	0,015625	0,0625	0,03125	0,078125	0,03125	0,109375
2	0,015625	0,03125	0,025197	0,015625	0,027455	0,03125	0,078125	0,03125	0,09375	0,03125	0,046875
3	0,015625	0,015625	0,075888	0,046875	0,058393	0,03125	0,078125	0,015625	0,09375	0,015625	0,078125
4	0,015625	0,015625	0,092282	0,03125	0,089817	0,03125	0,0625	0,03125	0,078125	0,03125	0,078125
5	0,015625	0,03125	0,092411	0,03125	0,092126	0,03125	0,09375	0,046875	0,078125	0,015625	0,09375
6	0,03125	0,015625	0,092	0,03125	0,091568	0,046875	0,0625	0,03125	0,09375	0,03125	0,078125
7	0,03125	0,046875	0,088943	0,03125	0,093533	0,03125	0,03125	0,015625	0,078125	0,046875	0,09375
8	0,015625	0,03125	0,088911	0,03125	0,089309	0,03125	0,015625	0,03125	0,078125	0,015625	0,078125
9	0,015625	0,046875	0,064428	0,03125	0,094535	0,03125	0,0625	0,03125	0,078125	0,03125	0,09375
10	0,03125	0,03125	0,086763	0,015625	0,091758	0,046875	0,078125	0,03125	0,078125	0,03125	0,09375
11	0,015625	0,03125	0,084416	0,03125	0,087011	0,015625	0,078125	0,015625	0,09375	0,03125	0,09375
12	0,015625	0,03125	0,088843	0,03125	0,093225	0,03125	0,078125	0,046875	0,0625	0,046875	0,0625
13	0,015625	0,03125	0,090984	0,03125	0,059573	0,03125	0,078125	0,03125	0,078125	0,03125	0,078125
14	0,015625	0,015625	0,087523	0,015625	0,09433	0,03125	0,078125	0,015625	0,109375	0,015625	0,078125
15	0,03125	0,015625	0,07594	0,03125	0,083557	0,03125	0,078125	0,015625	0,03125	0,03125	0,09375
16	0,03125	0,03125	0,056241	0,015625	0,059787	0,03125	0,078125	0,03125	0,0625	0,03125	0,078125
17	0,03125	0,03125	0,058269	0,03125	0,094871	0,015625	0,09375	0,03125	0,015625	0,015625	0,109375
18	0,015625	0,03125	0,082018	0,03125	0,087907	0,015625	0,078125	0,015625	0,078125	0,015625	0,03125
19	0,03125	0,03125	0,082819	0,03125	0,071593	0,015625	0,09375	0,03125	0,078125	0,03125	0,0625
20	0,015625	0,015625	0,058164	0,03125	0,074407	0,03125	0,078125	0,046875	0,078125	0,03125	0,015625
21	0,015625	0,015625	0,082859	0,015625	0,081054	0,03125	0,078125	0,046875	0,09375	0,03125	0,03125
22	0,03125	0,03125	0,087909	0,015625	0,058736	0,03125	0,078125	0,03125	0,09375	0,03125	0,078125
23	0,015625	0,03125	0,080746	0,015625	0,092278	0,046875	0,078125	0,03125	0,0625	0,015625	0,078125
24	0,03125	0,015625	0,084144	0,03125	0,058661	0,046875	0,078125	0,03125	0,109375	0,015625	0,0625
25	0,046875	0,03125	0,08026	0,015625	0,059255	0,015625	0,03125	0,03125	0,078125	0,03125	0,0625
26	0,03125	0,03125	0,078992	0,03125	0,092213	0,015625	0,078125	0,03125	0,09375	0,03125	0,078125
27	0,03125	0,03125	0,072395	0,03125	0,085183	0,015625	0,078125	0,03125	0,078125	0,03125	0,0625
28	0,03125	0,015625	0,092873	0,03125	0,086755	0,015625	0,0625	0,046875	0,078125	0,046875	0,046875
29	0,03125	0,03125	0,057467	0,03125	0,092651	0,03125	0,078125	0,03125	0,078125	0,015625	0,078125
30	0,03125	0,015625	0,087901	0,015625	0,082272	0,015625	0,09375	0,03125	0,078125	0,015625	0,078125

Tabela B.4: Tempos Aferidos - Sistema Equilibrado (Com GD)

Tempos Aferidos (s)											
Execução	Métodos										
	DSS	Método 1	Método 2	Método 3	Método 4	Método 5	Método 6	Método 7	Método 8	Método 9	Método 10
31	0,03125	0,046875	0,081975	0,03125	0,079708	0,03125	0,0625	0,03125	0,09375	0,015625	0,09375
32	0,03125	0,015625	0,082689	0,03125	0,092391	0,03125	0,046875	0,03125	0,078125	0,03125	0,09375
33	0,046875	0,03125	0,070811	0,03125	0,058587	0,03125	0,078125	0,03125	0,078125	0,046875	0,078125
34	0,03125	0,015625	0,084333	0,03125	0,092063	0,015625	0,09375	0,03125	0,09375	0,03125	0,09375
35	0,046875	0,03125	0,081042	0,03125	0,098044	0,03125	0,078125	0,03125	0,078125	0,046875	0,078125
36	0,046875	0,015625	0,076707	0,03125	0,08795	0,03125	0,078125	0,03125	0,09375	0,03125	0,0625
37	0,015625	0,015625	0,082849	0,046875	0,086326	0,046875	0,078125	0,03125	0,078125	0,03125	0,078125
38	0,015625	0,03125	0,026812	0,03125	0,058838	0,03125	0,0625	0,03125	0,09375	0,015625	0,078125
39	0,03125	0,03125	0,072509	0,03125	0,083081	0,015625	0,078125	0,03125	0,09375	0,03125	0,0625
40	0,046875	0,109375	0,077399	0,046875	0,065724	0,03125	0,078125	0,046875	0,09375	0,03125	0,0625
41	0,03125	0,046875	0,092292	0,09375	0,078635	0,125	0,0625	0,125	0,09375	0,046875	0,09375
42	0,03125	0,09375	0,092039	0,09375	0,086521	0,09375	0,078125	0,03125	0,078125	0,046875	0,078125
43	0,046875	0,046875	0,076682	0,078125	0,066978	0,03125	0,078125	0,078125	0,09375	0,09375	0,078125
44	0,046875	0,0625	0,075895	0,078125	0,054568	0,09375	0,09375	0,078125	0,0625	0,09375	0,03125
45	0,0625	0,0625	0,090628	0,0625	0,058898	0,0625	0,09375	0,125	0,09375	0,09375	0,078125
46	0,046875	0,078125	0,081166	0,109375	0,07989	0,078125	0,09375	0,125	0,078125	0,109375	0,078125
47	0,046875	0,046875	0,03511	0,03125	0,086096	0,078125	0,078125	0,109375	0,09375	0,125	0,0625
48	0,046875	0,09375	0,02511	0,0625	0,063966	0,078125	0,09375	0,125	0,078125	0,09375	0,078125
49	0,046875	0,09375	0,065648	0,09375	0,056395	0,109375	0,09375	0,078125	0,0625	0,109375	0,078125
50	0,03125	0,046875	0,043788	0,03125	0,092387	0,109375	0,109375	0,109375	0,0625	0,109375	0,078125
51	0,03125	0,09375	0,081638	0,046875	0,092883	0,109375	0,109375	0,046875	0,09375	0,046875	0,0625
52	0,0625	0,078125	0,0916	0,03125	0,092035	0,09375	0,078125	0,125	0,109375	0,109375	0,0625
53	0,0625	0,09375	0,079689	0,046875	0,068251	0,0625	0,0625	0,109375	0,078125	0,09375	0,09375
54	0,046875	0,125	0,057738	0,09375	0,071307	0,09375	0,0625	0,09375	0,09375	0,03125	0,078125
55	0,046875	0,125	0,085671	0,09375	0,077618	0,09375	0,046875	0,09375	0,109375	0,03125	0,078125
56	0,046875	0,125	0,057941	0,09375	0,082044	0,109375	0,09375	0,078125	0,078125	0,03125	0,0625
57	0,046875	0,109375	0,06869	0,09375	0,02703	0,125	0,09375	0,109375	0,09375	0,03125	0,078125
58	0,03125	0,125	0,079629	0,09375	0,083035	0,109375	0,078125	0,09375	0,09375	0,09375	0,078125
59	0,03125	0,109375	0,072572	0,09375	0,024847	0,125	0,078125	0,078125	0,109375	0,078125	0,0625
60	0,03125	0,09375	0,073695	0,09375	0,038705	0,078125	0,125	0,09375	0,109375	0,09375	0,078125

B.3 Tempos Aferidos - Sistema Desequilibrado (Sem GD)

Tabela B.5: Tempos Aferidos - Sistema Desequilibrado (Sem GD)

Tempos Aferidos (s)											
Execução	Métodos										
	DSS	Método 1	Método 2	Método 3	Método 4	Método 5	Método 6	Método 7	Método 8	Método 9	Método 10
1	0,125	0,265625	0,047601	0,234375	0,14444	0,21875	0,046875	0,140625	0,03125	0,28125	0,046875
2	0,03125	0,296875	0,059546	0,296875	0,090126	0,296875	0,078125	0,15625	0,09375	0,296875	0,078125
3	0,046875	0,171875	0,088762	0,328125	0,076681	0,25	0,09375	0,265625	0,09375	0,3125	0,09375
4	0,015625	0,390625	0,098407	0,265625	0,078315	0,171875	0,078125	0,203125	0,078125	0,25	0,078125
5	0,03125	0,375	0,033674	0,453125	0,080803	0,234375	0,078125	0,21875	0,09375	0,234375	0,09375
6	0,015625	0,421875	0,091505	0,296875	0,039194	0,234375	0,09375	0,21875	0,09375	0,34375	0,109375
7	0,046875	0,3125	0,091373	0,21875	0,087258	0,34375	0,109375	0,234375	0,09375	0,171875	0,078125
8	0,046875	0,265625	0,059119	0,328125	0,081855	0,171875	0,09375	0,25	0,109375	0,21875	0,09375
9	0,03125	0,265625	0,084557	0,171875	0,094574	0,25	0,09375	0,203125	0,078125	0,234375	0,109375
10	0,046875	0,34375	0,024811	0,234375	0,086977	0,1875	0,09375	0,265625	0,09375	0,265625	0,09375
11	0,03125	0,40625	0,024976	0,203125	0,087222	0,21875	0,078125	0,265625	0,046875	0,390625	0,078125
12	0,03125	0,3125	0,025537	0,140625	0,079098	0,203125	0,0625	0,078125	0,015625	0,28125	0,03125
13	0,03125	0,34375	0,076767	0,265625	0,081632	0,1875	0,09375	0,15625	0,015625	0,15625	0,078125
14	0,046875	0,28125	0,090369	0,171875	0,058927	0,234375	0,078125	0,265625	0,03125	0,234375	0,0625
15	0,078125	0,28125	0,079	0,171875	0,060304	0,328125	0,078125	0,234375	0,078125	0,203125	0,046875
16	0,015625	0,40625	0,077375	0,34375	0,05884	0,390625	0,109375	0,21875	0,09375	0,328125	0,078125
17	0,03125	0,296875	0,081765	0,25	0,088066	0,265625	0,109375	0,28125	0,109375	0,171875	0,09375
18	0,0625	0,203125	0,078923	0,21875	0,059105	0,203125	0,046875	0,171875	0,078125	0,25	0,015625
19	0,046875	0,3125	0,092461	0,453125	0,089304	0,3125	0,03125	0,1875	0,09375	0,296875	0,03125
20	0,046875	0,203125	0,058995	0,34375	0,091197	0,234375	0,0625	0,34375	0,0625	0,15625	0,03125
21	0,109375	0,125	0,058583	0,140625	0,090071	0,21875	0,078125	0,421875	0,078125	0,28125	0,078125
22	0,0625	0,109375	0,080964	0,296875	0,086469	0,40625	0,078125	0,109375	0,078125	0,296875	0,078125
23	0,046875	0,15625	0,057482	0,265625	0,077921	0,359375	0,015625	0,171875	0,09375	0,21875	0,078125
24	0,09375	0,15625	0,071077	0,265625	0,084874	0,1875	0,09375	0,34375	0,078125	0,265625	0,0625
25	0,0625	0,21875	0,031329	0,265625	0,072211	0,203125	0,09375	0,203125	0,109375	0,21875	0,09375
26	0,078125	0,140625	0,080718	0,21875	0,040073	0,34375	0,0625	0,171875	0,09375	0,328125	0,078125
27	0,109375	0,140625	0,092826	0,203125	0,074142	0,390625	0,09375	0,15625	0,0625	0,21875	0,109375
28	0,09375	0,140625	0,037022	0,1875	0,075141	0,25	0,09375	0,25	0,078125	0,203125	0,078125
29	0,046875	0,125	0,075447	0,1875	0,081314	0,390625	0,09375	0,15625	0,078125	0,453125	0,0625
30	0,09375	0,234375	0,092286	0,15625	0,086692	0,25	0,09375	0,265625	0,109375	0,1875	0,09375

Tabela B.6: Tempos Aferidos - Sistema Desequilibrado (Sem GD)

Tempos Aferidos (s)											
Execução	Métodos										
	DSS	Método 1	Método 2	Método 3	Método 4	Método 5	Método 6	Método 7	Método 8	Método 9	Método 10
31	0,03125	0,140625	0,033094	0,171875	0,077769	0,234375	0,09375	0,25	0,0625	0,28125	0,078125
32	0,078125	0,140625	0,091443	0,296875	0,081904	0,171875	0,046875	0,125	0,078125	0,375	0,078125
33	0,0625	0,15625	0,077773	0,203125	0,082437	0,265625	0,0625	0,125	0,078125	0,3125	0,109375
34	0,0625	0,140625	0,095298	0,203125	0,058762	0,234375	0,09375	0,125	0,0625	0,296875	0,03125
35	0,078125	0,171875	0,096063	0,359375	0,088837	0,328125	0,09375	0,28125	0,09375	0,34375	0,078125
36	0,1875	0,15625	0,083438	0,3125	0,091315	0,40625	0,09375	0,25	0,109375	0,34375	0,078125
37	0,125	0,15625	0,062116	0,21875	0,025087	0,265625	0,078125	0,203125	0,09375	0,28125	0,09375
38	0,0625	0,109375	0,041398	0,34375	0,039124	0,28125	0,078125	0,34375	0,078125	0,265625	0,0625
39	0,140625	0,140625	0,030605	0,28125	0,036029	0,28125	0,078125	0,1875	0,0625	0,375	0,078125
40	0,046875	0,125	0,03922	0,25	0,059003	0,234375	0,078125	0,234375	0,109375	0,140625	0,078125
41	0,09375	0,21875	0,078915	0,140625	0,038953	0,3125	0,0625	0,234375	0,03125	0,171875	0,03125
42	0,078125	0,125	0,087091	0,125	0,072521	0,359375	0,078125	0,28125	0,015625	0,25	0,078125
43	0,140625	0,140625	0,081885	0,125	0,058929	0,296875	0,09375	0,296875	0,03125	0,203125	0,078125
44	0,140625	0,140625	0,081991	0,171875	0,075868	0,359375	0,015625	0,359375	0,078125	0,21875	0,09375
45	0,09375	0,171875	0,072343	0,359375	0,085726	0,265625	0,09375	0,28125	0,03125	0,4375	0,078125
46	0,078125	0,171875	0,078753	0,234375	0,08141	0,21875	0,0625	0,25	0,09375	0,28125	0,078125
47	0,046875	0,15625	0,030522	0,21875	0,081849	0,1875	0,078125	0,328125	0,09375	0,296875	0,078125
48	0,0625	0,1875	0,082289	0,28125	0,087365	0,234375	0,109375	0,21875	0,09375	0,21875	0,078125
49	0,03125	0,140625	0,071694	0,375	0,091057	0,25	0,078125	0,3125	0,078125	0,46875	0,09375
50	0,09375	0,15625	0,077517	0,453125	0,051877	0,265625	0,09375	0,328125	0,078125	0,25	0,078125
51	0,0625	0,140625	0,093817	0,28125	0,058969	0,328125	0,078125	0,203125	0,078125	0,3125	0,09375
52	0,109375	0,125	0,092424	0,359375	0,088172	0,21875	0,109375	0,328125	0,09375	0,421875	0,109375
53	0,0625	0,140625	0,078942	0,234375	0,058349	0,265625	0,078125	0,234375	0,09375	0,25	0,078125
54	0,140625	0,21875	0,075571	0,1875	0,08489	0,21875	0,046875	0,21875	0,109375	0,265625	0,09375
55	0,203125	0,125	0,086541	0,3125	0,069233	0,265625	0,078125	0,21875	0,09375	0,28125	0,078125
56	0,15625	0,375	0,086469	0,203125	0,079163	0,328125	0,09375	0,234375	0,09375	0,203125	0,09375
57	0,140625	0,3125	0,08121	0,234375	0,089923	0,265625	0,078125	0,28125	0,09375	0,328125	0,03125
58	0,15625	0,375	0,090376	0,1875	0,078852	0,28125	0,109375	0,125	0,0625	0,421875	0,09375
59	0,15625	0,328125	0,089152	0,390625	0,074745	0,28125	0,09375	0,4375	0,078125	0,296875	0,09375
60	0,1875	0,328125	0,091669	0,296875	0,078526	0,265625	0,09375	0,3125	0,09375	0,296875	0,078125

B.4 Tempos Aferidos - Sistema Desequilibrado (Com GD)

Tabela B.7: Tempos Aferidos - Sistema Desequilibrado (Com GD)

Tempos Aferidos (s)											
Execução	Métodos										
	DSS	Método 1	Método 2	Método 3	Método 4	Método 5	Método 6	Método 7	Método 8	Método 9	Método 10
1	0,125	0,265625	0,027077	0,203125	0,105919	0,3125	0,0625	0,125	0,109375	0,109375	0,078125
2	0,03125	0,21875	0,090388	0,3125	0,083861	0,140625	0,0625	0,3125	0,09375	0,125	0,09375
3	0,046875	0,1875	0,086309	0,171875	0,073113	0,140625	0,0625	0,296875	0,109375	0,109375	0,09375
4	0,015625	0,21875	0,053811	0,265625	0,077027	0,25	0,046875	0,1875	0,09375	0,109375	0,078125
5	0,03125	0,359375	0,086099	0,296875	0,075983	0,171875	0,0625	0,203125	0,0625	0,109375	0,09375
6	0,015625	0,203125	0,058324	0,296875	0,074769	0,328125	0,015625	0,296875	0,046875	0,125	0,078125
7	0,046875	0,25	0,084943	0,203125	0,059785	0,3125	0,015625	0,25	0,078125	0,125	0,078125
8	0,046875	0,125	0,070761	0,21875	0,081911	0,28125	0,015625	0,390625	0,078125	0,109375	0,03125
9	0,03125	0,109375	0,078065	0,25	0,082272	0,234375	0,03125	0,359375	0,078125	0,125	0,078125
10	0,046875	0,171875	0,085729	0,1875	0,07795	0,21875	0,0625	0,265625	0,078125	0,125	0,046875
11	0,03125	0,3125	0,034781	0,1875	0,069328	0,234375	0,0625	0,21875	0,078125	0,140625	0,015625
12	0,03125	0,296875	0,065033	0,25	0,08885	0,171875	0,09375	0,265625	0,09375	0,109375	0,046875
13	0,03125	0,265625	0,096883	0,21875	0,092107	0,296875	0,09375	0,25	0,09375	0,109375	0,03125
14	0,046875	0,125	0,085436	0,296875	0,081362	0,28125	0,09375	0,28125	0,03125	0,109375	0,0625
15	0,078125	0,328125	0,093004	0,234375	0,093583	0,1875	0,0625	0,34375	0,109375	0,140625	0,046875
16	0,015625	0,3125	0,091012	0,171875	0,094456	0,296875	0,09375	0,203125	0,09375	0,140625	0,078125
17	0,03125	0,21875	0,0882	0,28125	0,059864	0,25	0,09375	0,171875	0,078125	0,109375	0,078125
18	0,0625	0,21875	0,097482	0,359375	0,08481	0,21875	0,109375	0,3125	0,09375	0,140625	0,0625
19	0,046875	0,296875	0,058531	0,234375	0,068234	0,234375	0,09375	0,375	0,078125	0,109375	0,046875
20	0,046875	0,203125	0,07954	0,390625	0,09563	0,34375	0,109375	0,171875	0,09375	0,125	0,0625
21	0,109375	0,234375	0,069239	0,28125	0,075791	0,21875	0,03125	0,109375	0,09375	0,125	0,09375
22	0,0625	0,25	0,072583	0,34375	0,086873	0,375	0,09375	0,203125	0,078125	0,140625	0,078125
23	0,046875	0,125	0,074061	0,265625	0,08722	0,28125	0,09375	0,265625	0,078125	0,125	0,046875
24	0,09375	0,328125	0,086704	0,21875	0,058563	0,203125	0,09375	0,171875	0,109375	0,140625	0,046875
25	0,0625	0,125	0,060184	0,34375	0,047982	0,34375	0,09375	0,203125	0,09375	0,109375	0,09375
26	0,078125	0,359375	0,091113	0,21875	0,055637	0,328125	0,078125	0,171875	0,09375	0,109375	0,09375
27	0,109375	0,21875	0,068059	0,25	0,083309	0,375	0,0625	0,234375	0,046875	0,109375	0,078125
28	0,09375	0,3125	0,092176	0,125	0,075942	0,1875	0,078125	0,21875	0,09375	0,125	0,09375
29	0,046875	0,25	0,092371	0,359375	0,077473	0,28125	0,078125	0,21875	0,109375	0,140625	0,09375
30	0,09375	0,265625	0,078123	0,28125	0,03619	0,21875	0,109375	0,34375	0,109375	0,125	0,09375

Tabela B.8: Tempos Aferidos - Sistema Desequilibrado (Com GD)

Tempos Aferidos (s)											
Execução	Métodos										
	DSS	Método 1	Método 2	Método 3	Método 4	Método 5	Método 6	Método 7	Método 8	Método 9	Método 10
31	0,03125	0,328125	0,075384	0,125	0,079656	0,375	0,078125	0,15625	0,109375	0,109375	0,015625
32	0,078125	0,171875	0,072339	0,1875	0,080566	0,28125	0,09375	0,125	0,09375	0,140625	0,078125
33	0,0625	0,4375	0,079604	0,234375	0,079904	0,171875	0,078125	0,125	0,03125	0,125	0,09375
34	0,0625	0,28125	0,087234	0,203125	0,058683	0,21875	0,078125	0,203125	0,078125	0,109375	0,0625
35	0,078125	0,328125	0,081924	0,265625	0,091834	0,234375	0,09375	0,171875	0,09375	0,140625	0,0625
36	0,1875	0,40625	0,058373	0,21875	0,073749	0,140625	0,078125	0,328125	0,09375	0,125	0,09375
37	0,125	0,140625	0,078314	0,359375	0,076331	0,140625	0,078125	0,234375	0,09375	0,140625	0,046875
38	0,0625	0,375	0,064126	0,1875	0,083924	0,125	0,09375	0,21875	0,09375	0,125	0,09375
39	0,140625	0,265625	0,05839	0,25	0,067535	0,3125	0,09375	0,21875	0,09375	0,140625	0,078125
40	0,046875	0,171875	0,079205	0,328125	0,024861	0,203125	0,078125	0,28125	0,0625	0,140625	0,078125
41	0,09375	0,171875	0,073564	0,234375	0,024868	0,203125	0,0625	0,15625	0,09375	0,109375	0,078125
42	0,078125	0,21875	0,080576	0,28125	0,035923	0,203125	0,078125	0,203125	0,015625	0,140625	0,09375
43	0,140625	0,3125	0,087337	0,203125	0,070125	0,1875	0,09375	0,25	0,09375	0,140625	0,078125
44	0,140625	0,15625	0,088881	0,28125	0,03629	0,234375	0,078125	0,15625	0,09375	0,109375	0,046875
45	0,09375	0,171875	0,058835	0,203125	0,074585	0,234375	0,078125	0,234375	0,09375	0,125	0,078125
46	0,078125	0,4375	0,058635	0,265625	0,075673	0,296875	0,078125	0,234375	0,09375	0,125	0,109375
47	0,046875	0,1875	0,058556	0,265625	0,072954	0,234375	0,109375	0,203125	0,0625	0,125	0,078125
48	0,0625	0,375	0,038406	0,203125	0,094432	0,265625	0,09375	0,1875	0,09375	0,109375	0,0625
49	0,03125	0,25	0,071017	0,1875	0,092579	0,234375	0,015625	0,171875	0,078125	0,125	0,0625
50	0,09375	0,25	0,070153	0,265625	0,085489	0,171875	0,03125	0,265625	0,078125	0,109375	0,078125
51	0,0625	0,1875	0,080822	0,234375	0,074847	0,203125	0,03125	0,203125	0,09375	0,125	0,046875
52	0,109375	0,28125	0,075972	0,234375	0,085958	0,1875	0,0625	0,21875	0,09375	0,109375	0,03125
53	0,0625	0,171875	0,080787	0,21875	0,097053	0,234375	0,0625	0,171875	0,078125	0,125	0,03125
54	0,140625	0,203125	0,089762	0,265625	0,087753	0,1875	0,0625	0,234375	0,078125	0,125	0,09375
55	0,203125	0,390625	0,071725	0,296875	0,081163	0,21875	0,078125	0,1875	0,0625	0,109375	0,09375
56	0,15625	0,390625	0,036817	0,125	0,081578	0,359375	0,0625	0,28125	0,109375	0,125	0,078125
57	0,140625	0,328125	0,026794	0,203125	0,080277	0,21875	0,0625	0,375	0,09375	0,125	0,109375
58	0,15625	0,1875	0,047159	0,296875	0,058985	0,21875	0,09375	0,171875	0,078125	0,125	0,09375
59	0,15625	0,203125	0,071314	0,296875	0,085274	0,25	0,09375	0,265625	0,09375	0,125	0,03125
60	0,1875	0,21875	0,065984	0,25	0,081299	0,265625	0,078125	0,296875	0,078125	0,140625	0,0625