UNIVERSIDADE FEDERAL DE JUIZ DE FORA
FACULDADE DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Ricardo Proba Fava

Estratégia de Simulação de Transitórios Eletromecânicos com Emprego de Unidades de Simulação Funcional e OpenDSS

Juiz de Fora
2024

Ricardo Proba Fava

Estratégia de Simulação de Transitórios Eletromecânicos com Emprego de Unidades de Simulação Funcional e OpenDSS

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Sistemas de Energia Elétrica

Orientador: Prof. Dr. Marcelo Aroca Tomim

Juiz de Fora

2024

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Proba Fava, Ricardo.
Estratégia de Simulação de Transitórios Eletromecânicos com Emprego de Unidades de Simulação Funcional e OpenDSS / Ricardo Proba Fava. – 2024.
153 f. : il.

Orientador: Marcelo Aroca Tomim
Dissertação (Mestrado) – Universidade Federal de Juiz de Fora, Faculdade de Engenharia. Programa de Pós-Graduação em Engenharia Elétrica, 2024.

1. Cálculo do fluxo de potência. 2. Estabilidade transitória. 3. FMU. 4. Simulação no tempo. 5. OpenDSS. 6. OpenModelica. I. Aroca Tomim, Marcelo, orient. II. Título.

Ricardo Proba Fava

Estratégia de Simulação de Transitórios Eletromecânicos com Emprego de Unidades de Simulação Funcional e OpenDSS

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Sistemas de Energia Elétrica

Aprovada em 9 de maio de 2024.

BANCA EXAMINADORA

Prof. Dr. Marcelo Aroca Tomim - Orientador
Universidade Federal de Juiz de Fora

Prof. Dr. Antonio Felipe da Cunha de Aquino
Universidade Federal de Santa Catarina

Prof. Dr. João Alberto Passos Filho
Universidade Federal de Juiz de Fora

Juiz de Fora, 24/04/2024.

Documento assinado eletronicamente por **Marcelo Aroca Tomim**, **Professor(a)**, em 09/05/2024, às 16:44, conforme horário oficial de Brasília, com fundamento no § 3º do

art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Joao Alberto Passos Filho**, **Professor(a)**, em 09/05/2024, às 16:44, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Antonio Felipe da Cunha de Aquino**, **Usuário Externo**, em 09/05/2024, às 16:45, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no Portal do SEI-Ufjf (www2.ufjf.br/SEI) através do ícone Conferência de Documentos, informando o código verificador **1789946** e o código CRC **190B1CF9**.

AGRADECIMENTOS

Agradeço às pessoas que contribuíram, direta ou indiretamente, para eu conseguir concluir minha dissertação de mestrado.

A meus pais, Fátima e Vanderli, que sempre investiram e acreditaram em mim. A carreira acadêmica é um momento que requer máxima dedicação, e o apoio deles foi fundamental.

A meus amigos e professores da Universidade Federal de Juiz de Fora (UFJF) e do Programa de Pós-Graduação em Engenharia Elétrica (PPEE) que conviveram comigo durante a minha trajetória acadêmica. Sempre serei grato pela troca de informações e de conhecimento.

Ao meu orientador Marcelo Aroca Tomim, que me apresentou o tema deste trabalho e, através de suas contribuições e recomendações, sempre priorizou a qualidade e a melhoria do mesmo.

À parceria entre a Petrobras e a UFJF, por meio do projeto RED - Recursos Energéticos Distribuídos, que tornou possível a realização deste trabalho.

"*A mente que se abre a uma nova ideia jamais voltará ao seu tamanho original.*"
Albert Einstein

RESUMO

Esta dissertação apresenta a implementação de um método alternado implícito para solucionar um problema de estabilidade transitória que envolve um sistema elétrico de potência de grande porte. Para isso, o sistema é dividido em três elementos (ou subsistemas): a rede elétrica de transmissão, os geradores elétricos dinâmicos (bem como os equipamentos dinâmicos associados) e a rede de distribuição. Considerando o desmembramento proposto, cada um dos subsistemas pode ser modelado em domínios distintos, como por exemplo domínios da frequência e do tempo. Além disso, torna-se possível atribuir ferramentas computacionais e métodos apropriados de solução para cada subsistema. Como resultado, pode-se esperar que a simulação ofereça uma maior flexibilidade na modelagem, possibilidade de reutilização dos modelos e potenciais ganhos de desempenho computacional. O primeiro subsistema, correspondente à rede de transmissão e suas cargas (sejam lineares ou não lineares), é representado por sua sequência positiva através da linguagem *Python*, onde é implementado o método de solução nodal. Para o segundo subsistema, correspondente às máquinas elétricas, deve-se considerar a sua natureza dinâmica e, portanto, a ferramenta computacional adotada é a *Functional Mock-up Interface* (FMI), que consiste em uma interface padronizada e gratuita que visa o acoplamento entre modelos matemáticos e simuladores. Os modelos, ou bibliotecas, que adotam a interface FMI são chamados *Functional Mock-up Units* (FMU), e podem ser classificados como dois tipos distintos: *Co-Simulation* (CS) e *Model Exchange* (ME). Este estudo adota a FMU do tipo ME, cuja solução deve ser realizada por um solucionador externo baseado no método de integração trapezoidal. O terceiro e último subsistema, correspondente à rede de distribuição, é representado através de um módulo baseado no programa OpenDSS, que por sua vez consiste em um simulador de sistemas elétricos de distribuição e Recursos Energéticos Distribuídos (RED). Para a comunicação em *Python* entre a rede de transmissão e os outros subsistemas, os modelos de máquinas síncronas são implementados pelo programa OMEdit, encapsulados em FMUs por um compilador em Modelica e importados para o ambiente em *Python* pela biblioteca PyFMI, enquanto o subsistema de distribuição é importado pelo módulo OpenDSSDirect. Definidos os subsistemas, os resultados obtidos pela simulação proposta com FMU do tipo ME coincidem de forma satisfatória com os obtidos pelo ANATEM, um programa de simulação dinâmica no domínio no tempo e análise de transitórios eletromecânicos.

Palavras-chave: Cálculo do fluxo de potência; Estabilidade transitória; FMU; Simulação no tempo; OpenDSS; OpenModelica.

ABSTRACT

This master thesis presents the implementation of an implicit alternating method to solve a transient stability problem involving a large electrical power system. To achieve this, the system is split into three elements (or subsystems): the electrical transmission network, the dynamic electrical generators (as well as the associated dynamic equipment) and the distribution network. Considering the proposed dismemberment, each one of the subsystems can be modeled in different domains, such as frequency and time domains. Furthermore, it's possible to assign appropriate computational tools and solution methods to each subsystem. As a result, the simulation is expected to offer greater modeling flexibility, potential for model reuse, and computational performance improvement. The first subsystem, corresponding to the transmission network and its loads (whether linear or non-linear), is represented by its equivalent positive sequence circuit using Python language, where the nodal solution method is implemented. For the second subsystem, corresponding to electrical machines, its dynamic nature must be considered and, therefore, the adopted computational tool is the Functional Mock-up Interface (FMI), which consists of a standardized and free interface that aims at coupling between mathematical models and simulators. The models, or libraries, that adopt the FMI interface are called Functional Mock-up Units (FMU), and can be classified as two different types: Co-Simulation (CS) and Model Exchange (ME). This study adopts the ME-type FMU, whose solution must be performed by an external solver based on the trapezoidal integration method. The third and final subsystem, corresponding to the distribution network, is represented through a package based on the OpenDSS, which consists of a simulator of electrical distribution systems and Distributed Energy Resources (RED). For Python-based communication between the transmission network and the other subsystems, synchronous machine models are implemented by the software OMEdit, encapsulated into FMUs by a Modelica compiler and imported into the Python environment by PyFMI library, while the distribution subsystem is imported by the OpenDSSDirect package. With the defined subsystems, the results obtained by the proposed simulation with an ME-type FMU match satisfactorily with those obtained by ANATEM, a program of time-domain dynamic simulation and analysis of electromechanical transients.

Keywords: Power flow calculation; Transient Stability; FMU; Time simulation; OpenDSS; OpenModelica.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

ANAREDE	Programa de Análise de Redes Elétricas
ANATEM	Programa de Análise de Transitórios Eletromecânicos
ANEEL	Agência Nacional de Energia Elétrica
API	*Application Programming Interface*
AVR	*Automatic Voltage Regulator*
BEN	Balanço Energético Nacional
CEPEL	Centro de Pesquisa em Energia Elétrica
COM	*Component Object Model*
CS	*Co-Simulation*
DLL	*Dynamic-Link Library*
EPRI	*Electric Power Research Institute*
FMI	*Functional Mockup Interface*
FMU	*Functional Mock-up Interface*
GD	Geração Distribuída
IEEE	*Institute of Electrical and Electronics Engineers*
LT	Linha de Transmissão
ME	*Model Exchange*
MMGD	Micro e Minigeração Distribuída
NIAE	*Normalized Integral Absolute Error*
OMEdit	*OpenModelica Connection Editor*
ONS	Operador Nacional do Sistema Elétrico
OpenDSS	*Open Distribution System Simulator*^{TM}
PAC	Ponto de Acoplamento Comum
PSS	*Power System Stabilizer*
RAP	Relatório de Análise de Perturbação
RED	Recursos Energéticos Distribuídos
SIN	Sistema Interligado Nacional
UFJF	Universidade Federal de Juiz de Fora
VBA	*Visual Basic for Applications*

LISTA DE SÍMBOLOS

δ	Ângulo da tensão interna do modelo de máquinas
Δt	Passo de integração da simulação no tempo
ε	Tolerância de convergência
ω	Desvio de velocidade angular do modelo de máquinas
ω_b	Desvio de velocidade angular nominal do modelo de máquinas
θ	Ângulo de fase da tensão nodal da rede de transmissão
A,C	Coeficientes da parcela de corrente constante do modelo de carga ZIP
B,D	Coeficientes da parcela de impedância constante do modelo de carga ZIP
d	Contador de iterações do OpenDSS
D_m	Constante de atrito viscoso, ou amortecimento, do modelo de máquinas
H	Constante de inércia do modelo de máquinas
$\mathbf{i}$	Vetor de injeções de correntes nodais das barras rede de transmissão
$\mathbf{i}_{dss}$	Vetor de injeções de correntes nodais das barras do OpenDSS
$\mathbf{i}_g$	Vetor de correntes de Norton dos geradores
$\mathbf{i}_l$	Vetor de correntes de carga da rede de transmissão
$\overline{I}_{adj}$	Corrente de ajuste de Norton da parcela de potência constante do modelo de carga ZIP
$\overline{I}_{adj,dss}$	Corrente de ajuste de Norton do OpenDSS
$\overline{I}_c$	Corrente da parcela de corrente constante do modelo de carga ZIP
$\overline{I}_{pac,dss}$	Corrente absorvida pelo PAC do OpenDSS
$\overline{I}_{pac,nr}$	Corrente absorvida pelo PAC encontrada pelo método de *Newton-Raphson*
$\overline{I}_{pac,rede}$	Corrente absorvida pelo PAC da rede de transmissão
i	Contador de iterações do conjunto modelo-rede
$\mathbf{J}$	Matriz jacobiana do método de *Newton-Raphson*
P_0, Q_0	Potências de regime permanente do modelo de máquinas
P_e	Potência elétrica do modelo de máquinas
P_m	Potência mecânica do modelo de máquinas
r	Contador de iterações da solução da rede de transmissão
$\overline{S}_{adj,dss}$	Potência de ajuste de Norton do OpenDSS
t	Instante de tempo atual
t_0	Instante de tempo inicial
t_e	Instante de evento
t_e^-	Instante imediatamente antes do evento
t_e^+	Instante imediatamente após o evento
t_f	Instante de tempo final
$\mathbf{v}$	Vetor de tensões nodais da rede de transmissão
$\mathbf{v}_{dss}$	Vetor de tensões nodais do OpenDSS
V_{mn}	Tensão limite de modelo de carga da rede de transmissão

$\overline{V}_t$	Tensão nodal da rede de transmissão
$\overline{V}_0$	Tensão nominal de regime permanente
$\overline{V}_{pac,dss}$	Tensão nodal do PAC do OpenDSS
$\overline{V}_{pac,rede}$	Tensão nodal do PAC da rede de transmissão
$\overline{V}_{pac,nr}$	Tensão do PAC encontrada pelo método de *Newton-Raphson*
$\mathbf{x}$	Vetor de variáveis de estado
$\mathbf{x}_h$	Vetor dos termos históricos das variáveis de estado
$\mathbf{Y}$	Matriz de admitâncias nodais da rede de transmissão
$\mathbf{Y}_{dss}$	Matriz de admitâncias nodais do OpenDSS
$\overline{Y}_{prim}$	Matriz de admitâncias nodais primitiva do OpenDSS
$\overline{y}_0$	Admitância de Norton da parcela de potência constante do modelo de carga ZIP da rede de transmissão
$\overline{y}_{0,dss}$	Admitância de Norton de regime permanente do OpenDSS
$\overline{y}_{dss}$	Admitância de Norton do OpenDSS
$\overline{y}_g$	Admitância de Norton do modelo de máquinas
$\overline{y}_i$	Admitância da parcela de impedância constante do modelo de carga ZIP
$\overline{y}_{mn}$	Admitância da parcela de impedância constante para baixas tensões do modelo de carga ZIP
$\overline{Z}_{tr}$	Impedância do transformador de conexão entre a rede de transmissão e de distribuição

SUMÁRIO

1 INTRODUÇÃO

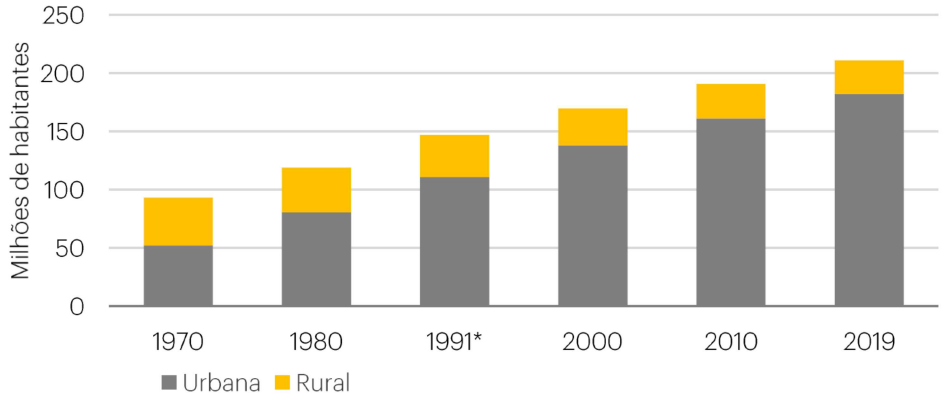
O presente trabalho é desenvolvido em torno da cossimulação eletromecânica de sistemas elétricos de potência, tendo em vista o crescimento da adoção de Recursos Energéticos Distribuídos (RED) em âmbito nacional e mundial. Neste capítulo introdutório, a Seção 1.1 apresenta uma breve contextualização acerca do cenário elétrico brasileiro e de alguns desafios enfrentados no Sistema Interligado Nacional (SIN) nos últimos anos. Em seguida, a Seção 1.2 apresenta os objetivos que motivaram a elaboração desta dissertação. Com os objetivos em mente, a Seção 1.3 realiza uma revisão bibliográfica de trabalhos que nortearam as pesquisas e as implementações computacionais desta dissertação, enquanto a Seção 1.4 apresenta o artigo resultante desse estudo. Por fim, a Seção 1.5 descreve a estrutura e o conteúdo dos próximos capítulos deste documento.

1.1 CONTEXTUALIZAÇÃO

Nos últimos 50 anos, tanto a sociedade quanto a economia passaram por uma série de transformações que se refletiram em mudanças nos hábitos de consumo. Em termos demográficos, aconteceram mudanças significativas no ritmo de crescimento populacional. No que diz respeito à economia, aconteceram diversos eventos importantes que impactaram significativamente a atividade econômica, tanto no Brasil quanto no mundo (EMPRESA DE PESQUISA ENERGÉTICA, 2023a).

Em relação à distribuição ao longo do território nacional, a população brasileira passou por um rápido processo de urbanização entre as décadas de 1940 e 1970. De 1970 até 2019, a população brasileira cresceu, em média, 1,7 % ao ano. Em 1970, 56 % da população estava nas cidades. Esse número saltou para 67,7 % em 1980 e chegou a 86,4 % em 2019. O gráfico completo da população brasileira está mostrado na Figura 1.

Figura 1 – Crescimento da população brasileira urbana e rural.



Fonte: (EMPRESA DE PESQUISA ENERGÉTICA, 2023a)

Tabela 1 – Capacidade instalada de geração elétrica de diferentes fontes no Brasil (em [MW]).

Ano	Hídrica	Térmica	Eólica	Solar
2013	86.018	36.528	2.202	5
2014	89.193	37.827	4.888	15
2015	91.650	39.563	7.633	21
2016	96.925	41.275	10.124	24
2017	100.275	41.628	12.283	935
2018	104.139	40.523	14.390	1.798
2019	109.058	41.219	15.378	2.473
2020	109.271	43.057	17.131	3.287
2021	109.350	44.866	20.771	4.632
2022	109.721	46.284	23.744	7.387

Fonte: (EMPRESA DE PESQUISA ENERGÉTICA, 2023b) (Adaptada)

Nesse mesmo período de 1970 e 2019, a matriz energética brasileira apresentou expressivas alterações em sua composição, refletindo os movimentos demográficos caracterizados por uma forte migração campo-cidade, bem como a relevância de pautas ambientais de sustentabilidade do planeta. Esse cenário possibilitou a introdução de novas fontes renováveis (eólica e solar, por exemplo) no território brasileiro.

Para verificar o crescimento das fontes eólica e solar de forma quantitativa e comparativa, são extraídos do relatório do Balanço Energético Nacional (BEN) os dados de capacidade instalada de geração elétrica no Brasil (EMPRESA DE PESQUISA ENERGÉTICA, 2023b), conforme a Tabela 1.

A Tabela 1 apresenta apenas as gerações hídrica, térmica, eólica e solar no período de 2013 até 2022. No entanto, no relatório completo do BEN, os primeiros dados de capacidade instalada de geração hídrica e térmica no Brasil são de 1974, enquanto os de geração eólica e solar passaram a ser apresentados apenas em 1994 e 2010, respectivamente.

Com base nos dados da Tabela 1, pode-se realizar as seguintes análises:

- Em 2014, a capacidade instalada da geração eólica mais do que dobrou em relação a 2013, isto é, foi de 2.202 [MW] para 4.888 [MW]. Desde então, passou a aumentar a uma taxa média de mais de 2.000 [MW] por ano.

- Em 2017, houve o início de um intenso crescimento da capacidade instalada da geração solar. Inicialmente estava em 24 [MW] em 2016, aumentou para 935 [MW] em 2017, aumentou novamente para 1.798 [MW] em 2018 e por último, esteve em 7.387 [MW] em 2022.

Tabela 2 – Capacidade instalada de geração elétrica de Micro e Minigeração Distribuída no Brasil (em [MW]).

Ano	Hídrica	Térmica	Eólica	Solar	Total
2015	1	2	0	13	17
2016	4	11	0	57	72
2017	37	24	10	175	246
2018	59	38	10	562	670
2019	97	63	10	1.992	2.162
2020	23	95	15	4.635	4.768
2021	63	115	15	8.771	8.965
2022	86	156	17	17.066	17.325

Fonte: (EMPRESA DE PESQUISA ENERGÉTICA, 2023b) (Adaptada)

- Enquanto a capacidade instalada das gerações hídrica e térmica passaram por aumentos de menos de 30 % de 2013 até 2022, as capacidades das gerações eólica e solar passaram por aumentos de 978,3 % e 147.640 %, respectivamente, no mesmo intervalo de tempo.

Outros dados que também podem ser extraídos do relatório do BEN correspondem à capacidade instalada de geração elétrica de Micro e Minigeração Distribuída (MMGD) no Brasil, conforme a Tabela 2.

Com base nos dados da Tabela 2, é possível notar um crescimento exponencial, com destaque para a geração solar, que tem representado mais de 90 % da capacidade instalada nacional de MMGD desde 2019. Conforme descrito no relatório completo do BEN, a MMGD teve seu crescimento incentivado por ações regulatórias, tais como a que estabelece a possibilidade de compensação da energia excedente produzida por sistemas de menor porte (*Net Metering*).

Após apresentar os dados da Tabela 1 e da Tabela 2, pode-se citar algumas vantagens associadas ao intenso crescimento da capacidade instalada das fontes eólica e solar, como por exemplo a redução dos impactos ambientais (em relação a fontes de energia tradicionais) e proximidade com as cargas, o que acarreta uma redução das perdas térmicas. No entanto, essas fontes alternativas de energia possuem uma natureza intermitente, o que pode comprometer o nível das tensões da rede e, consequentemente, pode introduzir problemas de estabilidade, controle e qualidade de energia. Além disso, a inércia reduzida compromete a robustez do sistema.

Portanto, estudando as particularidades de diferentes fontes de energia que atuam em conjunto no mesmo sistema elétrico, é necessário que haja representações computacionais específicas, com o objetivo de modelar as propriedades e os comportamentos dinâmicos

intrínsecos de cada tipo de geração existente dentro do sistema de interesse, conforme descrito em (CHAGAS; TOMIM, 2022). Para isso, tem-se a necessidade de ambientes e ferramentas computacionais que permitam o acoplamento de diferentes modelos e domínios em uma mesma simulação de um sistema elétrico. Integrando-se adequadamente múltiplas ferramentas de simulação, é possível simular grandes sistemas de forma heterogênea, resultando em uma plataforma única de simulação que permite a reutilização de ferramentas comprovadamente confiáveis ao longo de anos de desenvolvimento.

Na engenharia de potência, os conceitos de simulação e modelagem são fundamentais para avaliação de recursos como controlabilidade, confiabilidade e operabilidade geral de dispositivos e de sistemas de potência como um todo. Graças a isso, é possível ter a previsão do comportamento de um sistema de potência antes da ocorrência de uma contingência real (curto-circuito, sobrecarga etc), e no estudo dos efeitos das ações de controle necessárias para evitar tais contingências (PALENSKY et al., 2017).

Para ilustrar a complexidade (e também a importância) da modelagem adequada de diferentes fontes de energia, pode-se tomar como exemplo um caso real do Sistema Interligado Nacional (SIN). No dia 15 de agosto de 2023, às 08h30, uma ocorrência no SIN causou a interrupção de 23.368 [MW], aproximadamente 34,5 % da carga total de 67.507 [MW] daquele momento. A perturbação teve início com a abertura do terminal de Quixadá da LT 500 kV Quixadá - Fortaleza II, sem a incidência de curto-circuito no sistema elétrico. O evento provocou a separação elétrica das regiões Norte e Nordeste das regiões Sul, Sudeste/Centro-Oeste, com desligamento de linhas de transmissão entre essas regiões, afetando 25 estados e o Distrito Federal. O restabelecimento total das cargas foi autorizado às 14h49.

Conforme o Relatório de Análise de Perturbação (RAP) encaminhado pelo Operador Nacional do Sistema Elétrico (ONS), a abrupta redução de tensão após a perda de uma única linha de transmissão foi consequência do desempenho dos controles dos parques eólicos e fotovoltaicos no perímetro da LT, em especial no que tange à capacidade de suporte dinâmico de potência reativa. Esse desempenho ficou aquém dos modelos matemáticos fornecidos pelos agentes ao ONS, o que não permitiu identificar os riscos relacionados ao cenário operativo pré-distúrbio que resultou nos desligamentos em cascata (OPERADOR NACIONAL DO SISTEMA ELÉTRICO, 2023).

Sabendo disso, o RAP indicou que algumas das providências a serem tomadas são, por exemplo, a alteração e validação dos modelos dos geradores eólicos e fotovoltaicos (bem como seus controladores) da região do evento, visando assegurar uma reprodução fidedigna (no formato dos programas ANATEM e PSCAD) dos seus desempenhos em campo.

Resumindo esta seção, os dados apresentados do Balanço Energético Nacional evidenciam o aumento de gerações tanto tradicionais quanto intermitentes, enquanto a

contingência apresentada do Sistema Interligado Nacional evidencia a importância da modelagem adequada das fontes de energia. Essa contextualização serve como ponto de partida para os objetivos desta dissertação.

1.2 OBJETIVOS

Com base no panorama do sistema elétrico brasileiro apresentado na seção anterior, o presente trabalho tem como objetivo apresentar a estrutura de um protótipo de programa de estabilidade transitória, cujos elementos e subsistemas são atribuídos a ferramentas computacionais distintas. Para a implementação desse protótipo, o sistema operacional escolhido é o *Linux*, pois possui as vantagens de ser gratuito, seguro e altamente personalizável (em comparação com o *Windows*, por exemplo).

Com a separação proposta, tem-se a vantagem de desmembramento de um sistema complexo em subsistemas menores (PALENSKY et al., 2017), o que acarreta uma especialização da solução dos mesmos, dado que cada subsistema pode ser modelado em domínios distintos, como por exemplo domínios da frequência e do tempo. Consequentemente, pode-se esperar maior flexibilidade na modelagem e potenciais ganhos de desempenho computacional.

Sabendo disso, são considerados inicialmente dois subsistemas: a rede elétrica de transmissão e o modelo de máquinas elétricas. No caso do primeiro subsistema, estabelece-se que a transmissão, bem como suas cargas lineares e não lineares, é representada por sua sequência positiva. Para isso, adota-se a linguagem *Python*, pois se trata de uma linguagem de programação gratuita, aberta para o público e utilizada em diferentes áreas da engenharia.

Para o segundo subsistema, deve-se ter em mente que, ao contrário do primeiro, as máquinas elétricas (assim como outros equipamentos dinâmicos, tais como cargas dinâmicas e elementos FACTS) possuem uma natureza dinâmica. Portanto, a modelagem desse subsistema é realizada através da *Functional Mock-up Interface* (FMI), uma interface padronizada e gratuita que visa o acoplamento entre modelos matemáticos e simuladores. Os modelos, ou bibliotecas, que adotam a interface FMI são chamados de *Functional Mock-up Unit* (FMU), e podem ser de dois tipos: *Co-Simulation* (CS) e *Model Exchange* (ME) (MODELICA ASSOCIATION, 2021). Enquanto o primeiro tipo possui um solucionador embutido, o segundo precisa adotar um solucionador presente no ambiente de simulação.

Após a implementação do subsistema de transmissão em conjunto com o modelo de máquinas elétricas, é introduzido ao protótipo um terceiro subsistema, que é a rede elétrica de distribuição. No protótipo em *Python*, a sua comunicação com o restante do sistema elétrico é realizada pelo módulo `OpenDSSDirect`, que permite importar os dados da distribuição e executar as funções do OpenDSS sem a necessidade de instalação do programa propriamente dito (DUGAN; MONTENEGRO, 2021). Além disso, o subsistema de

distribuição é representado por sua sequência positiva, assim como a transmissão.

1.3 REVISÃO BIBLIOGRÁFICA

Com os objetivos apresentados, realiza-se uma revisão bibliográfica dos principais trabalhos utilizados como base para a elaboração desta dissertação. Nesses trabalhos, foram implementados diferentes modelos matemáticos através de ferramentas computacionais distintas, contribuindo para o estudo de estabilidade transitória de sistemas elétricos de potência.

Em (DOMMEL; SATO, 1972), foi explicado que, após a solução do fluxo de potência de regime permanente, dois conjuntos de equações devem ser resolvidos como uma função do tempo, onde o primeiro conjunto é um sistema de equações algébricas (que descrevem o comportamento de regime permanente da rede incluindo modelagens de cargas estáticas no método nodal), enquanto o segundo é um sistema de equações diferenciais (que descrevem o comportamento dinâmico das máquinas e de seus circuitos de controle). Em seguida, os métodos de solução das equações algébrico-diferenciais foram divididos em três categorias, incluindo o método alternado, onde as equações diferencias são solucionadas através da integração trapezoidal.

Em (THEODORO et al., 2018), foi apresentada uma simulação híbrida conectando dois subsistemas. O primeiro é do tipo EMT (transitórios eletromagnéticos), que contém usinas fotovoltaicas modeladas em instâncias distintas do MATLAB, enquanto o segundo subsistema é do tipo quasi-estático no domínio da frequência, que consiste em uma rede de distribuição modelada pelo OpenDSS. Essa modelagem híbrida se mostrou essencial, uma vez que economizou esforço de processamento computacional e permitiu a reutilização de ferramentas já amplamente testadas.

Em (MOHSENI-BONAB et al., 2020), foi proposta uma plataforma de cossimulação integrada de redes de transmissão e distribuição, onde o programa MATLAB foi utilizado para modelar a rede de transmissão (pacote MATPOWER), enquanto o programa OpenDSS modelou a rede de distribuição. Com duas soluções independentes entre si, o Ponto de Acoplamento Comum entre ambas as redes foi implementada em *Python*. A plataforma proposta também consiste em uma *toolbox* de Algoritmo Genético, utilizada para encontrar a solução ótima de duas funções objetivo, que buscam maximizar a margem de carregamento e minimizar as perdas elétricas. Com a cossimulação aplicada a uma rede de mais de 68.000 nós, os resultados demonstraram a eficiência dos dispositivos de controle disponíveis.

Em (SILVA, L. T. F. W. DA, 2020), modelos matemáticos de componentes de um sistema de geração eólica foram desenvolvidos em linguagem Modelica e compilados em FMUs. Depois, foram integrados a um algoritmo mestre, que coordenou e executou a cossimulação em um ambiente de programação de linguagem *Python*. Nesse ambiente, as simulações foram possíveis graças à divisão do sistema em subsistemas compostos por

aerogeradores e rede elétrica. Resultados evidenciaram tanto a precisão como ganhos computacionais via paralelização.

Em (CHAGAS; TOMIM, 2022), foi destacado que a inserção crescente de Recursos Energéticos Distribuídos em sistemas de potência contribui para que seja necessário um contínuo desenvolvimento de ferramentas de cossimulação. Tendo isso em mente, foi proposta uma cossimulação entre sistemas de transmissão e distribuição baseada em linhas de transmissão fictícias. O sistema de transmissão foi compilado em uma FMU e depois acoplado a redes de distribuição modeladas em OpenDSS, mostrando a possibilidade de interconexão de subsistemas modelados em domínios distintos e solucionados por métodos numéricos diferentes. Além de conseguir resultados satisfatórios, a estratégia proposta apresentou um tempo computacional reduzido em relação à simulação do sistema completo.

1.4 PUBLICAÇÕES DECORRENTES DA DISSERTAÇÃO

Com base nos estudos realizados e apresentados nesta dissertação, foi desenvolvido o seguinte artigo:

- FAVA, R. P., TOMIM, M. A., CHAGAS, I. B. O., "Estratégia de Simulação de Transitórios Eletromecânicos para Sistemas de Transmissão Integrados a FMUs", *X Simpósio Brasileiro de Sistemas Elétricos* (SBSE), 2023.

1.5 ESTRUTURA DA DISSERTAÇÃO

Além do capítulo de introdução, esta dissertação está dividida em mais 5 capítulos.

No Capítulo 2, apresentam-se conceitos gerais de *Functional Mock-up Unit*, bem como as características próprias de cada tipo existente. Em seguida, apresentam-se conceitos específicos da FMU do tipo *Model Exchange* (ME), escolhida para este trabalho. Conhecidas as definições pertinentes para a simulação computacional, utiliza-se um modelo matemático escrito no programa OMEdit para criar uma FMU do tipo ME, que por sua vez é importada para o ambiente *Python* para ser testada numa simulação no tempo. Nessa simulação, o processo de integração é realizado pelo método trapezoidal através de uma função em *Python* criada para este trabalho. Após a validação da simulação, pode-se adotar modelos complexos de máquinas elétricas no OMEdit para que sejam posteriormente encapsulados em FMUs.

No Capítulo 3, descreve-se a forma como as FMUs de modelo de máquinas elétricas se comunicam com a rede de transmissão (representada em *Python*) durante a simulação no tempo. Com essa comunicação, cada subsistema é solucionado separadamente, caracterizando o denominado método alternado do conjunto modelo-rede. Sabendo disso, é descrito que a solução individual do subsistema de transmissão adota o método nodal, onde as cargas elétricas são representadas pela modelagem do tipo ZIP. Em seguida, adiciona-se à

simulação no tempo uma forma de tratar os eventos e contingências que podem acontecer na rede elétrica, como por exemplo um curto-circuito. Por último, o capítulo se encerra com o método alternado na forma de pseudocódigos e funções escritos em *Python*.

No Capítulo 4, é introduzido o programa OpenDSS, utilizado como base para possibilitar a inclusão de redes elétricas de distribuição ao restante do sistema. Com essa inclusão, o método alternado do capítulo anterior é submetido às adaptações necessárias, assim como o cálculo do ponto inicial de operação de regime permanente. Por último, o capítulo se encerra com a leitura e execução do sistema de distribuição na forma de pseudocódigos e funções escritos em *Python*.

Com o protótipo acoplando três subsistemas dentro de uma simulação de sistema elétrico de potência, o Capítulo 5 descreve a execução de simulações no tempo de um sistema de transmissão de 11 barras acoplado a geradores representados por FMUs e um sistema de distribuição de 38 barras. Para a validação da simulação do ambiente *Python*, os resultados obtidos são comparados com os programas ANAREDE (Análise de Redes Elétricas) e ANATEM (Análise de Transitórios Eletromecânicos), ambos desenvolvidos pelo CEPEL (Centro de Pesquisa em Energia Elétrica) e adotados pelo ONS como ferramentas oficiais para estudos de estabilidade.

No Capítulo 6, tem-se as conclusões finais acerca das metodologias apresentadas e implementadas nos capítulos anteriores. Em seguida, são descritas propostas de melhorias e desenvolvimentos subsequentes desta dissertação.

Depois do capítulo de conclusões, esta dissertação possui 6 apêndices.

No Apêndice A, realiza-se uma comparação da precisão computacional dos dois tipos de FMUs disponibilizados pela interface FMI 2.0.3 (isto é, *Co-Simulation* e *Model Exchange*), com o objetivo de avaliar qual é o mais adequado para a simulação de sistemas elétricos desta dissertação.

No Apêndice B, é apresentado o método de integração numérica trapezoidal, um método consolidado na área de engenharia para discretização e solução de equações diferenciais. Neste trabalho, é utilizado para a solução da modelagem dinâmica dos geradores do sistema elétrico.

No Apêndice C, são apresentados os códigos em *Python* correspondentes ao solucionador externo que é utilizado para o processo de integração do modelo de máquinas encapsulado em um arquivo FMU do tipo *Model Exchange*. Esses códigos utilizam como base o método de integração trapezoidal.

No Apêndice D, são apresentados os dados dos sistemas teste utilizados para a execução da simulação no tempo do protótipo do programa de estabilidade. Também são apresentados os dados do AVR e do PSS correspondentes.

No Apêndice E, é apresentado o método iterativo de *Newton-Raphson*, um método

consolidado na área de engenharia para cálculo de regime permanente de sistemas de potência.

Por fim, no Apêndice F, é apresentada a modelagem de carga do tipo ZIP, escolhida para as cargas conectadas ao sistema de transmissão. Consiste em uma modelagem algébrica não linear cujas equações são essenciais para o cálculo iterativo da solução da rede durante a simulação no tempo.

2 FUNDAMENTOS DA SIMULAÇÃO ELETROMECÂNICA COM O EMPREGO DE FMUs

Neste trabalho, para a representação do subsistema elétrico de transmissão (incluindo cargas lineares e não lineares), utiliza-se uma modelagem estática. No entanto, as máquinas elétricas conectadas à rede compõem um subsistema separado, devido à sua natureza dinâmica. Para essa modelagem dinâmica, adota-se a *Functional Mock-up Interface* (FMI), uma interface padronizada e gratuita que visa o acoplamento entre modelos matemáticos e simuladores.

A primeira versão (FMI 1.0) foi iniciada pela Daimler AG e publicada em 2010, com o objetivo de melhorar o intercâmbio de modelos de simulação entre fornecedores e OEMs (*Original Equipment Manufacturer*, ou Fabricante Original do Equipamento) (MODELICA ASSOCIATION, 2021). Atualmente o desenvolvimento dessa interface continua através da participação de outras empresas e institutos de pesquisa, como por exemplo a Bosch, Haldex, Siemens, Saab, entre outras multinacionais das áreas de engenharia e automação.

Com as contínuas melhorias realizadas, a interface atualmente se encontra na versão FMI 3.0, que apresenta, por exemplo, atualizações na conexão de sinais compatíveis e controle de eventos. No entanto, a versão 3.0 é relativamente recente e não possui suporte por parte do OpenModelica. Portanto, este trabalho adota a versão da FMI 2.0.3, que disponibiliza ferramentas consolidadas.

Sabendo disso, este capítulo apresenta uma introdução sobre as propriedades da FMI e a estrutura básica do seu arquivo executável compactado, denominado FMU. Em seguida, apresenta-se uma diferenciação entre os tipos de FMU existentes na versão 2.0.3. Após detalhar as características pertinentes para a sua implementação na simulação eletromecânica, descreve-se uma forma de exportação de uma FMU para que seja possível testar a sua implementação dentro do ambiente de simulação em *Python*. Com a validação do teste, a interface é aplicada ao contexto de modelagem de máquinas elétricas desta dissertação.

2.1 INTRODUÇÃO

O padrão FMI define uma interface a ser implementada por um arquivo executável denominado FMU (*Functional Mockup Unit*). Essa interface padronizada é utilizada através de um ambiente de simulação de forma que uma ou mais instâncias de um modelo específico possam ser criadas e resolvidas individualmente ou em conjunto com outros modelos. Dessa forma, a interface é capaz de utilizar a FMU de forma simplificada dentro de um ambiente de simulação.

A FMU consiste em um arquivo compactado do tipo `.fmu` que contém princi-

palmente arquivos XML e arquivos em linguagem C. Os arquivos do tipo XML contêm as definições das variáveis expostas (ou acessíveis) de forma padronizada, bem como as definições de outras informações estruturais do modelo. As funções em linguagem C, que podem ser fornecidas de forma binária ou por código-fonte, definem não só as equações (sejam diferenciais, algébricas ou discretas no tempo) necessárias para o modelo da FMU, como também o acesso às ferramentas de gerenciamento da simulação. Utiliza-se a linguagem C devido à sua natureza portátil e simplificada em relação a outras linguagens de programação. Por último, a FMU pode conter arquivos extras de formatos específicos, como tabelas e mapas, dependendo do modelo correspondente.

Para as funções do modelo de uma FMU, adotam-se bibliotecas compartilhadas, ou Bibliotecas de Vínculo Dinâmico. Essa nomenclatura se deve ao fato de conter códigos que podem ser carregados dentro da memória e reutilizados por mais de um programa simultaneamente. Com isso, economizam-se recursos, códigos duplicados e espaço computacional. Dois exemplos de extensões de bibliotecas de vínculo dinâmico são: `.dll` (utilizada tipicamente pelo sistema *Windows*) e `.so` (utilizada tipicamente pelo sistema *Linux*).

Conforme descrito em (MODELICA ASSOCIATION, 2021), a Biblioteca de Objetos Compartilhados (extensão `.so`) é utilizada especialmente quando o fornecedor da FMU necessita ocultar o código-fonte com o objetivo de garantir o sigilo do conteúdo. Esse formato de arquivo compactado pode conter parâmetros físicos ou dimensões geométricas que não devem ser abertos. Por isso, é uma forma conveniente para que fabricantes de componentes forneçam a seus clientes um modelo dinâmico e padronizado de seus produtos sem comprometer sua propriedade intelectual. De forma resumida, o cliente conecta essa FMU a um modelo de um sistema maior, com o objetivo de verificar se aquele componente específico opera de forma satisfatória em conjunto com os outros componentes dentro do sistema em questão (PALENSKY et al., 2017).

Outras propriedades da FMU são (MODELICA ASSOCIATION, 2021):

- Reduzido tempo de execução: a comunicação entre uma FMU e um simulador através da FMI não apresenta um tempo de execução significativo. Isso se deve a alguns fatores, como por exemplo: primeiro, uma técnica de *caching* (armazenamento de dados) que evita processamentos exaustivos das mesmas variáveis. Segundo, o intercâmbio de vetores em vez de variáveis escalares.

- Robustez numérica: o padrão FMI permite que problemas numericamente críticos sejam tratados de forma robusta.

- Pouca quantidade de funções: a FMI consiste em poucas funções ortogonais, o que evita a presença de funções redundantes que possam ser definidas em função de

outras. Com isso, tem-se um API (*Application Programming Interface*, ou Interface de Programação de Aplicações) com uma documentação compacta e de fácil utilização.

2.2 ESTRUTURA DA FMU

Conforme dito anteriormente, este trabalho adota a versão da FMI 2.0.3, onde os modelos dinâmicos podem ser de dois tipos distintos: *Co-Simulation* (CS) e *Model Exchange* (ME). Enquanto o primeiro tipo possui um solucionador próprio, o segundo tipo precisa ser resolvido através de um solucionador do ambiente de simulação onde está sendo executado (ANDERSSON, 2016). Essa diferença está mostrada na Figura 2(a) e na Figura 2(b).

Na FMI do tipo *Model Exchange*, um ambiente de modelagem representa um modelo de sistema dinâmico em linguagem C. Além disso, a avaliação das equações correspondentes a um determinado modelo pode ser realizada em diferentes ambientes de simulação, seja com integradores explícitos ou implícitos, seja com passo de integração fixo ou variável. Se o código em C descrever um sistema contínuo, o sistema é resolvido através dos integradores presentes no ambiente de simulação onde o sistema está inserido. Os modelos tratados por essa interface possuem aplicação em sistemas de controle e microprocessadores.

Na FMI do tipo *Co-Simulation*, tem-se um padrão de interface para acoplamento de ferramentas de simulação, ou subsistemas, em um ambiente de cossimulação. Esses subsistemas realizam intercâmbio de dados através de pontos de comunicação discretos e são resolvidos de forma independente uns dos outros, isto é, são resolvidos por seus solucionadores individuais. Além disso, o intercâmbio de dados é controlado através de um algoritmo mestre, que também possui a função de sincronização de todos os solucionadores (ou escravos). No entanto, o algoritmo mestre, desde o mais simples até o mais elaborado, não é parte do padrão FMI.

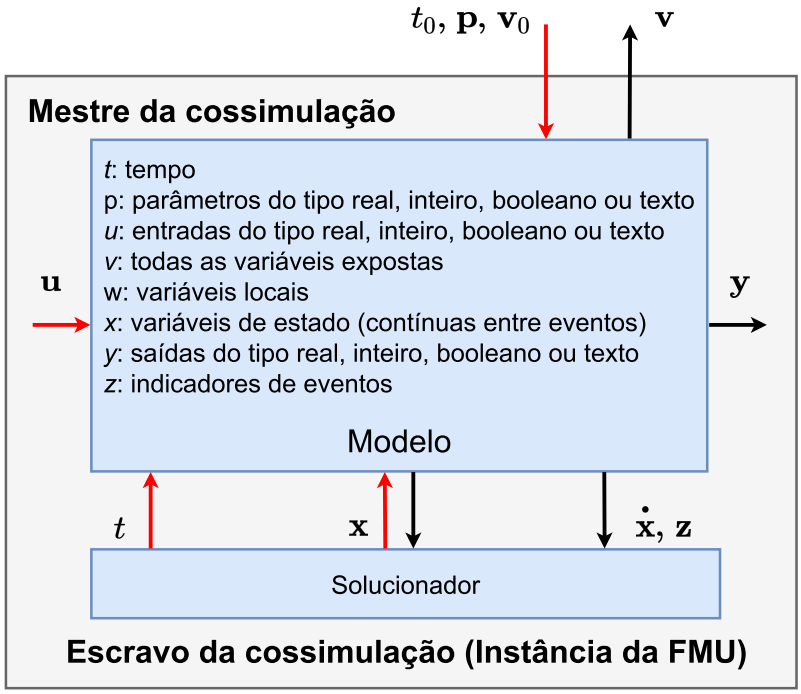
Os subsistemas podem ser representados por blocos, que possuem suas variáveis de estado $\mathbf{x}(t)$ e são conectados a outros blocos (ou subsistemas) através das entradas $\mathbf{u}(t)$ e das saídas $\mathbf{y}(t)$.

2.3 PADRÃO FMI PARA INTERCÂMBIO DE MODELOS

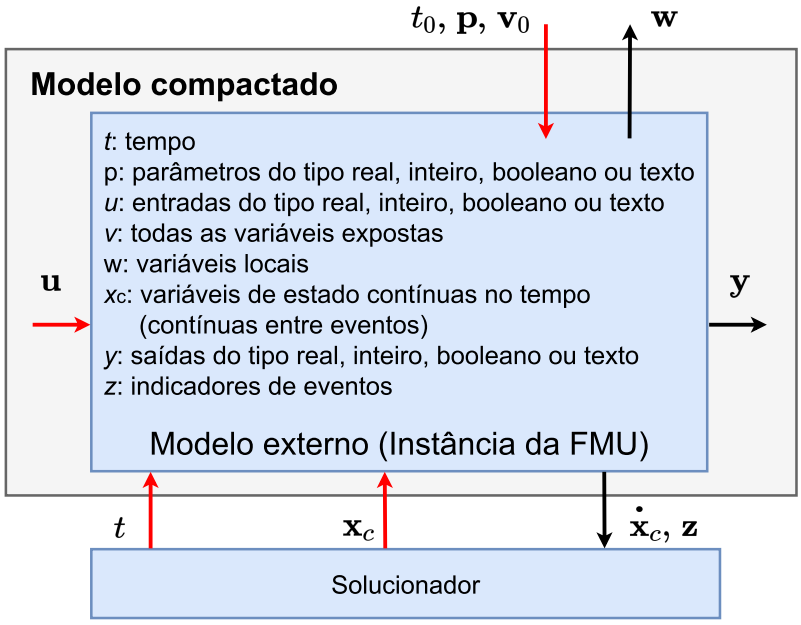
Conforme dito na seção anterior, os subsistemas da FMI do tipo *Co-Simulation* possuem soluções individuais e trocam informações restritamente por meio de pontos de comunicação discretos. No entanto, essa restrição acarreta um atraso dos dados trocados, ocasionando resultados indesejados de simulação, conforme mostrado no Apêndice A.

Portanto, para a modelagem das máquinas elétricas desta dissertação, escolhe-se adotar a FMU do tipo *Model Exchange*, onde o atraso das informações intercambiadas

Figura 2 – Representação dos dois tipos de FMUs da versão 2.0.3 da interface FMI.



(a) FMU do tipo *Co-Simulation* (CS).



(b) FMU do tipo *Model Exchange* (ME).

Fonte: (MODELICA ASSOCIATION, 2021) (Adaptada)

é evitado pela presença do modo de inicialização, modo contínuo no tempo e modo de evento, que são explicados em detalhes ao final desta seção.

Assim como o CS, o objetivo da interface do tipo ME é solucionar numericamente um sistema de equações diferenciais, algébricas e discretas no tempo. Isto é, solucionar um sistema de equações da forma mostrada em (2.1).

$$\begin{cases} \dot{\mathrm{x}}_\mathrm{c} = f(\mathrm{x}_\mathrm{c}, \mathrm{y}, \mathrm{u}, t) \\ \mathrm{y} = g(\mathrm{x}_\mathrm{c}, \mathrm{y}, \mathrm{u}, t) \end{cases} \tag{2.1}$$

No caso da versão 2.0.3 da interface, os sistemas de equações diferenciais ordinárias (ou EDOs) são formulados através de espaço de estados na presença de eventos.

Durante a solução das equações mostradas em (2.1) dentro dessa versão da FMI, tem-se a variável independente de tempo da forma $t = (t_R, t_I)$, onde se tem $t_R \in \mathbb{R}$ e $t_I \in \mathbb{N}$. A parte real t_R é a variável independente da FMU e é responsável pela descrição do comportamento de tempo contínuo do modelo entre eventos. A parte inteira t_I é um contador que enumera (e, portanto, distingue) os eventos que estão programados para ocorrerem em um mesmo instante de tempo contínuo t_R. Quando não há nenhum evento em um determinado instante t_R, tem-se $t_I = 0$.

Por exemplo, quando uma FMU possui um evento em $t_R = 2$ [s], uma variável sofre uma descontinuidade. Os cálculos correspondentes a um evento são executados sem realizar nenhum avanço no tempo. Sabendo disso, o instante de tempo no qual ocorre o evento é definido como $(2, 0)$, enquanto o instante no qual a integração é retomada é definido como $(2, 1)$.

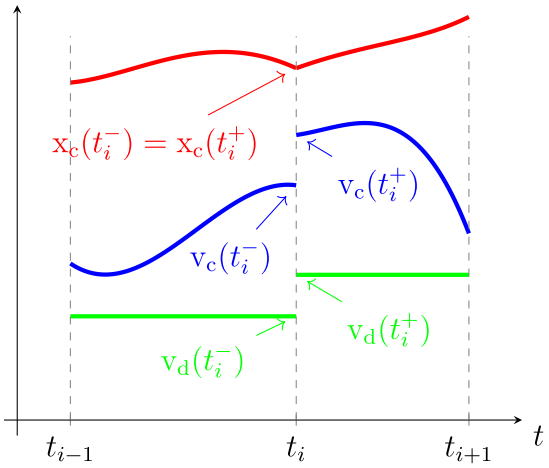
As EDOs híbridas suportadas pela FMI são descritas como sistemas contínuos no tempo por partes. Descontinuidades podem ocorrer em instantes de tempo $t_0, t_1, \ldots, t_n$, (onde $t_{i-1} < t_i$), denominados instantes de evento. Conhecendo esse conceito, pode-se citar os seguintes tipos de variáveis:

- Variável de estado contínua entre eventos (x_c): variável cuja derivada se comporta como uma função f contínua dentro de cada intervalo de tempo entre um evento e outro, conforme as expressões mostradas em (2.2)

$$\begin{cases} \dot{\mathrm{x}}_\mathrm{c} = f_i(\mathrm{x}_\mathrm{c}, t), \;\; t_{i-1}^+ \leq t \leq t_i^- \\ \mathrm{x}_\mathrm{c}(t_i^-) = \mathrm{x}_\mathrm{c}(t_i^+) \\ \dot{\mathrm{x}}_\mathrm{c} = f_{i+1}(\mathrm{x}_\mathrm{c}, t), \;\; t_i^+ \leq t \leq t_{i+1}^- \end{cases} \tag{2.2}$$

- Variável contínua no tempo (v_c): variável que se comporta como uma função g contínua dentro de cada intervalo de tempo entre um evento e outro, conforme as expressões mostradas em (2.3). Apenas variáveis reais podem ser contínuas no

Figura 3 – Variáveis do padrão FMI do tipo *Model Exchange*.



Fonte: (MODELICA ASSOCIATION, 2021) (Adaptada)

tempo.

$$\begin{cases} v_c = g_i(t), \;\; t_{i-1}^+ \leq t \leq t_i^- \\ v_c(t_i^-) \neq v_c(t_i^+) \\ v_c = g_{i+1}(t), \;\; t_i^+ \leq t \leq t_{i+1}^- \end{cases} \tag{2.3}$$

- Variável discreta no tempo (v_d): variável que se altera apenas nos instantes de evento, isto é, assume um valor constante k específico para cada intervalo de tempo entre um evento e outro, conforme as expressões mostradas em (2.4). Exemplo: controlador de um sistema de dados amostrados.

$$\begin{cases} v_d = k_i, \;\; t_{i-1}^+ \leq t \leq t_i^- \\ v_d(t_i^-) \neq v_d(t_i^+) \\ v_d = k_{i+1}, \;\; t_i^+ \leq t \leq t_{i+1}^- \end{cases} \tag{2.4}$$

As variáveis $x_c(t)$, $v_c(t)$ e $v_d(t)$ estão apresentadas na Figura 3.

A solução de um modelo na interface FMI é dividida em etapas, e em cada etapa são utilizados diferentes sistemas de equações e métodos de solução. Essas etapas são categorizadas de acordo com os seguintes modos:

1. Modo de inicialização: utilizado no instante inicial t_0 para computar os valores iniciais dos estados contínuos no tempo ($x_c(t_0)$) e dos estados discretos no tempo ($x_d(t_0)$).

2. Modo de tempo contínuo: utilizado para computar os valores de todas as variáveis contínuas no tempo (reais) entre um evento e outro, através da solução numérica de equações diferenciais ordinárias e algébricas. Todas as variáveis discretas no tempo são constantes durante esta etapa e, portanto, as equações discretas correspondentes não são avaliadas.

3. Modo de evento: utilizado para computar os novos valores das variáveis algébricas, bem como as variáveis discretas no tempo que são ativadas no instante de evento. Para isso, calcula-se a solução das equações algébricas e discretas correspondentes, o que pode resultar em descontinuidades nas derivadas das variáveis de estado.

   Na versão 2.0, a FMU não informa se a variável discreta no tempo está ativa ou inativa (não computada) no instante de evento. Portanto, o ambiente de simulação precisa assumir que, no instante de evento, as variáveis discretas no tempo sempre são computadas, embora internamente na FMU apenas um subconjunto de fato seja computado.

2.4 TESTE DE IMPLEMENTAÇÃO DA FMU DO TIPO INTERCÂMBIO DE MODELOS

A interface FMI fornece os recursos necessários para a exportação de FMUs a partir de modelos criados por diferentes programas de simulação, como por exemplo OpenModelica, Matlab, Dymola e EMTP. Nesta seção, o modelo matemático a ser testado é implementado através do programa OMEdit (*OpenModelica Connection Editor*), que consiste em uma interface para edição e execução do compilador OpenModelica (*OpenModelica Compiler*), também denominado OMC. O OMEdit possui a vantagem de uma rápida comunicação com o OMC conectado como uma DLL (OPEN SOURCE MODELICA CONSORTIUM, 2023).

Além disso, o OMEdit possui não só uma interface para criação, desenvolvimento e simulação, como também possui um conjunto de bibliotecas pré-definidas de modelos matemáticos, elétricos, magnéticos, mecânicos, etc. Esses modelos, por sua vez, podem ser visualizados e editados através de uma interface em *script* ou uma interface gráfica, onde podem ser representados por blocos e até mesmo conectados manualmente uns aos outros.

Quando o modelo está pronto, o usuário pode escolher os parâmetros gerais da simulação no tempo, como o tempo total, passo de integração, método de integração, etc. Caso a simulação seja bem sucedida, pode-se visualizar no próprio OMEdit os resultados obtidos através de gráficos em função do tempo. Outra forma de acessar os resultados é através de um arquivo do tipo `.mat`, que é gerado pelo programa quando a simulação no tempo é executada. Esse arquivo, por sua vez, pode ser lido pela linguagem *Python* através do pacote `DyMat`, conforme mostrado adiante nesta seção durante a verificação dos resultados da simulação.

Tanto o código escrito no OMEdit quanto os escritos em *Python* são apresentados nesta seção. Como são executados dentro do sistema operacional *Linux*, a biblioteca de vínculo dinâmico correspondente às funções da FMU criada possui a extensão `.so` (*Shared Object*, ou Objeto Compartilhado).

Algoritmo 1 – Implementação do sistema teste no programa OMEdit.

```
1  model SistemaTeste

2    // Entrada do modelo:
3    input Real u;

4    // Saída do modelo:
5    output Real y;

6    // Variável de estado do modelo:
7    Real x;

8    // Condições iniciais do modelo:
9    initial equation
10     der(x) = 0.0;

11   // Equações que descrevem o modelo:
12   equation
13     0.5*der(x) = - x + u;
14     y = 2*x;

15 end SistemaTeste;
```

Fonte: (CHAGAS, 2022)

Apresentadas as plataformas de simulação escolhidas, deve-se agora apresentar o modelo matemático a ser testado nesta seção. Conforme as expressões mostradas em (2.5), o modelo consiste em um sistema linear formado por uma equação diferencial, uma equação algébrica e uma condição inicial, onde constam a variável de entrada u, a variável de saída y e a variável de estado x (CHAGAS, 2022).

$$\begin{cases} 0{,}5\,\dfrac{dx}{dt} = -x + u \\ \qquad y = 2x \\ \left.\dfrac{dx}{dt}\right|_{t=0} = 0 \end{cases} \tag{2.5}$$

Definido o sistema linear, realiza-se a sua implementação no OMEdit dentro de um modelo denominado `SistemaTeste`, conforme mostrado no Algoritmo 1.

Primeiro, define-se a entrada `u` (`input` - linha 3), a saída `y` (`output` - linha 5) e a variável real `x` (linha 7). Em seguida, define-se a condição inicial (`initial equation` - linha 9) e as equações em função do tempo (`equation` - linha 12) através do operador `der`, que corresponde à derivada no tempo.

Após executar e salvar o sistema teste implementado no OMEdit, o modelo contido no arquivo `SistemaTeste.mo` deve ser compactado dentro de um arquivo do tipo `.fmu`. Para isso, realiza-se a exportação da FMU através de um programa implementado no ambiente *Python*, conforme mostrado no Algoritmo 2.

A exportação da FMU é realizada através do API `OMPython` (*OpenModelica Python*

Algoritmo 2 – Exportação da FMU do sistema teste através do ambiente *Python*.

```
1  from OMPython import OMCSessionZMQ, ModelicaSystem
2  import os
3  omc = OMCSessionZMQ()

4  # Definir o arquivo em Modelica e o modelo correspondente:
5  mo_file = "/diretorio/SistemaTeste.mo"
6  model = ["SistemaTeste.SistemaTeste"]
7  os.chdir("/temp/")

8  # Definir opções de exportação da FMU:
9  options = "-d=newInst"
10 options += "-homotopyApproach=equidistantGlobal"
11 options += "-generateSymbolicLinearization"

12 # Carregar o modelo a ser utilizado para a exportação da FMU:
13 mod = ModelicaSystem(mo_file, model, ["Modelica"], commandLineOptions=options)

14 # Definir o tipo da FMU como "me" (Model Exchange):
15 fmu_path = mod.convertMo2Fmu(version="2.0", fmuType="me")

16 # Exportar o arquivo .fmu para o diretório desejado:
17 fmu_base_name = os.path.basename(fmu_path)
18 fmu_name = ".".join(fmu_base_name.split(".")[-2:])
19 os.rename("{:s}".format(fmu_name), "/diretorio/{:s}".format(fmu_name))
```

Fonte: Elaborado pelo autor (2024)

Interface), que possibilita a fácil comunicação entre o OpenModelica e o ambiente *Python*. Além da sua natureza gratuita e de código aberto, possui a propriedade de fornecer duas classes: `OMCSession` e `OMCSessionZMQ`. Na linha 1 do algoritmo escrito, importa-se o `OMCSessionZMQ`, devido à alta performance da biblioteca `ZeroMQ`.

A outra classe a ser importada pelo `OMPython` é a `ModelicaSystem`, cuja instância é atribuída a um objeto denominado `mod`. Para isso, utilizam-se as seguintes entradas: o caminho para o arquivo `SistemaTeste.mo` (denominado `mo_file` - linha 5), o caminho do modelo em Modelica contido dentro do arquivo `.mo` e, por último, algumas opções (não obrigatórias) de exportação da FMU.

Obtido o objeto `mod` na linha 13, utiliza-se o comando `convertMo2Fmu` para converter o modelo de linguagem Modelica para FMU. Para a execução desse comando, define-se a versão da FMI como 2.0 e define-se o tipo da FMU como ME (linha 15). Finalmente, realizam-se comandos de ajuste do nome do arquivo `.fmu` a ser exportado para o diretório desejado.

Após a exportação da FMU do tipo ME, realiza-se a sua importação através de um outro programa implementado no ambiente *Python*, conforme mostrado no Algoritmo 3.

A importação da FMU é realizada através do módulo `PyFMI`, projetado para fornecer uma interface de alto nível e de fácil carregamento de FMUs em *Python* (ANDERSSON; ÅKESSON; FÜHRER, 2016). Essa biblioteca é disponível como um pacote independente ou como parte da plataforma de código aberto `JModelica.org`.

Algoritmo 3 – Importação da FMU do tipo *Model Exchange* através do módulo `PyFMI`.

```
1  from pyfmi.fmi import FMUModelME2
2  from trapezoidal import TrapezoidalIntegrator
3  import numpy as np

4  # Parâmetros da simulação (passo de integração e tempo final):
5  dt = 1e-3
6  tf = 10.0

7  # Importação da FMU:
8  fmu_me = FMUModelME2("/diretorio/SistemaTeste.fmu", log_level=2)
9  print(fmu_me.get_capability_flags())
10 fmu_me.instantiate()
11 fmu_me.setup_experiment(tolerance=1e-4, start_time=0.0, stop_time=tf)

12 # Modo de inicialização da FMU: entrada inicial igual a 1.
13 u_atual = 1
14 fmu_me.enter_initialization_mode()
15 fmu_me.set("u", u_atual)
16 fmu_me.exit_initialization_mode()

17 # Inicialização do integrador: método trapezoidal.
18 integrator = TrapezoidalIntegrator(fmu_me, dt, tf, atol=1e-4, rtol=1e-2)

19 # Modo de tempo contínuo da FMU:
20 fmu_me.enter_continuous_time_mode()
21 fmu_me.time = 0.0
```

Fonte: Elaborado pelo autor (2024)

A plataforma `JModelica.org`, por sua vez, visa a simulação e otimização de modelos em linguagem Modelica e modelos que seguem o padrão FMI. Além disso, é resultado das pesquisas do Departamento de Controle e Automação da Universidade de Lund, na Suécia, e atualmente é mantida e desenvolvida pela Modelon AB em colaboração com pesquisadores da área acadêmica. Dessa forma, problemas relevantes na indústria podem inspirar novas pesquisas e algoritmos de última geração podem ser propostos no meio acadêmico para depois serem difundidos para aplicações industriais (MODELON, 2018).

No Algoritmo 3, o `PyFMI` possibilita que o ambiente *Python* realize simulações no tempo, manipulação de parâmetros e representação gráfica dos resultados a partir de um modelo dinâmico compactado em uma FMU, seja do tipo *Co-Simulation* ou do tipo *Model Exchange*. Como este trabalho adota a FMU do tipo *Model Exchange* e a versão 2.0 da interface FMI, o `PyFMI` importa a classe `FMUModelME2`.

Para instanciar a classe `FMUModelME2`, a entrada correspondente é o caminho para o arquivo `SistemaTeste.fmu`. Com isso, tem-se o objeto denominado `fmu_me` (linha 8). Em seguida, utiliza-se o comando `setup_experiment` (linha 11) para atribuir ao objeto os parâmetros de simulação (tolerância de convergência igual a `1e-4`, instante inicial igual a `0.0` [s] e instante final igual a `tf = 10.0` [s]).

Agora, o modelo da FMU do tipo *Model Exchange* está pronto para entrar no

modo de inicialização, onde é necessário definir a entrada inicial `u_atual = 1` através do comando `set` (linha 15). A entrada é unitária até o instante de 1,0 [s], e depois desse instante se torna igual a -1. Com isso, tem-se a função de `u` descrita em (2.6).

$$u = \begin{cases} 1, & t < 1{,}0 \text{ [s]} \\ -1, & t \geq 1{,}0 \text{ [s]} \end{cases} \tag{2.6}$$

Após o modo de inicialização da FMU, deve-se agora instanciar o solucionador externo criado em *Python* para este trabalho. A classe a ser instanciada é denominada `TrapezoidalIntegrator`, devido ao fato de adotar o método trapezoidal, que por sua vez consiste em um método de integração numérica implícita consolidado na área da engenharia devido à sua estabilidade numérica (DOMMEL; SATO, 1972). A explicação matemática e gráfica do método trapezoidal está descrita no Apêndice B, enquanto os arquivos em `.py` correspondentes estão anexos no Apêndice C.

Os parâmetros de entrada dessa classe são: o objeto que contém o modelo da FMU (`fmu_me`), passo de integração (`dt = 1e-3` [s]), tempo final (`tf = 10.0` [s]), tolerância absoluta (`atol = 1e-4` [pu]) e tolerância relativa (`rtol = 1e-2` %). Com a instância da classe `TrapezoidalIntegrator`, tem-se o objeto `integrator` (linha 18), cujo termo histórico inicial é definido com base nas condições iniciais conhecidas do modelo, conforme mostrado na equação (2.7).

$$x_{h,0} = x(0) + \frac{\Delta t}{2} f(0) \tag{2.7}$$

Por fim, as últimas etapas da inicialização do modelo da FMU consistem na ativação do modo contínuo através do comando `enter_continuous_time_mode` (linha 20) e na definição do instante de tempo inicial dos modelos como igual a `0.0` [s] (linha 21).

Agora que o modelo da FMU do sistema teste está pronto para a simulação no tempo, a continuação do programa é mostrada conforme o Algoritmo 4.

A simulação no tempo é realizada através de uma estrutura de `while`, onde cada laço corresponde a um instante de tempo. Portanto, para o primeiro laço, o instante atual da simulação (`t`) é definido como igual a `0.0` [s], enquanto o último laço define o valor de `t` como igual ao instante final `tf`. Sabendo disso, a simulação no tempo escrita em *Python* se divide nas seguintes etapas:

1. Processo de integração (início na linha 5): realizado a cada instante de tempo da simulação. Nesta etapa, os mais recentes valores de instante `t` e entrada `u_atual` são atribuídos ao objeto `fmu_me` (linhas 6 e 7).

   Em seguida, o passo preditor é executado pelo comando `predictor` (linha 9), que consiste na estimativa inicial do estado através do método de Euler explícito, com o objetivo de acelerar a convergência do processo de integração. O estado resultante

Algoritmo 4 – Simulação no tempo da FMU do tipo *Model Exchange* através do `PyFMI`.

```
# Instante inicial da simulação:
n = 0
t = 0.0

while t <= tf:

  # ================= 1) Processo de integração:

  fmu_me.time = t
  fmu_me.set("u", u_atual)

  # ---------- 1.1) Passo preditor:
  integrator.predictor()
  fmu_me.continuous_states = integrator.x[:]

  # ---------- 1.2) Iteração funcional:
  step_iter = 0
  flag_model_converged = False
  while flag_model_converged == False:

    # a) Avanço de um passo de integração:
    integrator.step()

    # b) Teste de convergência do modelo:
    flag_model_converged = integrator.check_convergence()

    # c) Atribuição do estado mais recente ao modelo da FMU:
    fmu_me.continuous_states = integrator.x[:]
    step_iter += 1

  # ---------- 1.3) Atualização do termo histórico do integrador:
  integrator.update_history()

  # ---------- 1.4) Armazenamento dos resultados:
  x[n] = fmu_me.get("x")[0]
  u[n] = u_atual
  y[n] = fmu_me.get("y")[0]
  tempo[n] = t
  n += 1

  # ================= 2) Tratamento de eventos:
  if t == 1.0:

    # Novo valor de entrada: u = -1.
    u_atual = -1
    fmu_me.set("u", u_atual)

    # Derivada da variável de estado:
    integrator.der_x[:] = fmu_me.get_derivatives()

    # Variável de estado, que é contínua entre eventos:
    integrator.x[:] = fmu_me.continuous_states

    # Atualização do termo histórico do integrador:
    integrator.update_history()

    # Armazenamento dos resultados:
    u[n] = u_atual
    x[n] = fmu_me.get("x")[0]
    y[n] = fmu_me.get("y")[0]
    tempo[n] = t
    n += 1

  # ================= 3) Avanço de tempo:
  t += dt
```

Fonte: Elaborado pelo autor (2024)

da predição é atribuído ao modelo (linha 10), conforme mostrado em (2.8).

$$\begin{cases} x^{(0)} = x_{prev} + \Delta t \cdot f_{prev} \\ \texttt{fmu\_me.continuous\_states} \leftarrow x^{(0)} \end{cases} \tag{2.8}$$

Em seguida, são inicializados os seguintes parâmetros da iteração funcional do instante `t`: contador de iterações igual a `step_iter = 0` (linha 12) e variável binária de convergência igual a `flag_model_converged = False` (linha 13). Com isso, as etapas da iteração funcional são as seguintes:

a) Avanço de um passo de integração (linha 16): realizado pela função do integrador denominada `step`, que consiste em coletar da FMU a derivada atual (f) pelo comando `get_derivatives` e utilizá-la para a estimativa do estado por meio do método trapezoidal, conforme mostrado em (2.9).

$$\begin{cases} f \leftarrow \texttt{fmu\_me.get\_derivatives} \\ x = \dfrac{\Delta t}{2} f + x_h \end{cases} \tag{2.9}$$

b) Teste de convergência do modelo (linha 18): com as atualizações de f e x, realiza-se a função `check_convergence`, que consiste em verificar se os erros absolutos e relativos do estado atendem dadas tolerâncias de convergência. Em caso afirmativo, tem-se a variável binária `flag_model_converged` igual a `True` e encerra-se a iteração funcional do instante t. Caso contrário, tem-se `flag_model_converged` igual a `False` e continua-se a iteração funcional.

c) Atribuição dos estado mais recente ao modelo da FMU (linha 20): o estado mais recente do integrador é atribuído ao objeto `fmu_me` para ser utilizado na iteração seguinte, conforme mostrado em (2.10).

$$\texttt{fmu\_me.continuous\_states} \leftarrow x \tag{2.10}$$

Em seguida, incrementa-se o valor de `step_iter` (linha 21) e retorna-se à etapa 1a em caso de não convergência do modelo.

Após a convergência da iteração funcional do instante t, realiza-se uma função do integrador denominada `update_history` (linha 23), que consiste em atualizar o termo histórico (x_h) a ser utilizado pelo instante de tempo seguinte. Essa atualização está mostrada em (2.11).

$$\begin{cases} x_{prev} = x \\ f_{prev} = f \\ x_h = x_{prev} + \dfrac{\Delta t}{2} f_{prev} \end{cases} \tag{2.11}$$

Por último, a etapa do processo de integração é finalizada pelo armazenamento dos resultados nos vetores de estados (`x`), entradas (`u`), saídas (`y`) e instantes de tempo (`tempo`).

2. Tratamento de eventos (início na linha 30): realizado quando o instante de tempo atual da simulação for igual ao instante de evento, ou seja, quando a relação `t == 1.0` é atendida (linha 31). Nesse caso, o valor de `u_atual` passa a ser igual a `-1` e é atribuído ao objeto `fmu_me` pelo comando `set` (linha 34).

   Com essa nova entrada, é extraída do modelo da FMU a derivada correspondente (linha 36), ao mesmo tempo que se mantém a continuidade do estado entre eventos (linha 38). Esses dois procedimentos estão mostrados em (2.12).

$$\begin{cases} f \leftarrow \texttt{fmu\_me.get\_derivatives} \\ x \leftarrow \texttt{fmu\_me.continuous\_states} \end{cases} \tag{2.12}$$

   Em seguida, realiza-se a função do integrador denominada `update_history` (linha 40), que consiste em atualizar o termo histórico (x_h) a ser utilizado pelo instante de tempo seguinte. Essa atualização está mostrada em (2.13).

$$\begin{cases} x_{prev} = x \\ f_{prev} = f \\ x_h = x_{prev} + \dfrac{\Delta t}{2} f_{prev} \end{cases} \tag{2.13}$$

   Por último, a etapa do tratamento de eventos é finalizada pelo armazenamento dos resultados nos vetores `x`, `u`, `y` e `tempo`.

3. Avanço de tempo (linha 48): nessa última etapa da simulação no tempo, realiza-se o incremento do instante `t` através do passo de integração `dt` e encerra-se o laço de tempo do `while`.

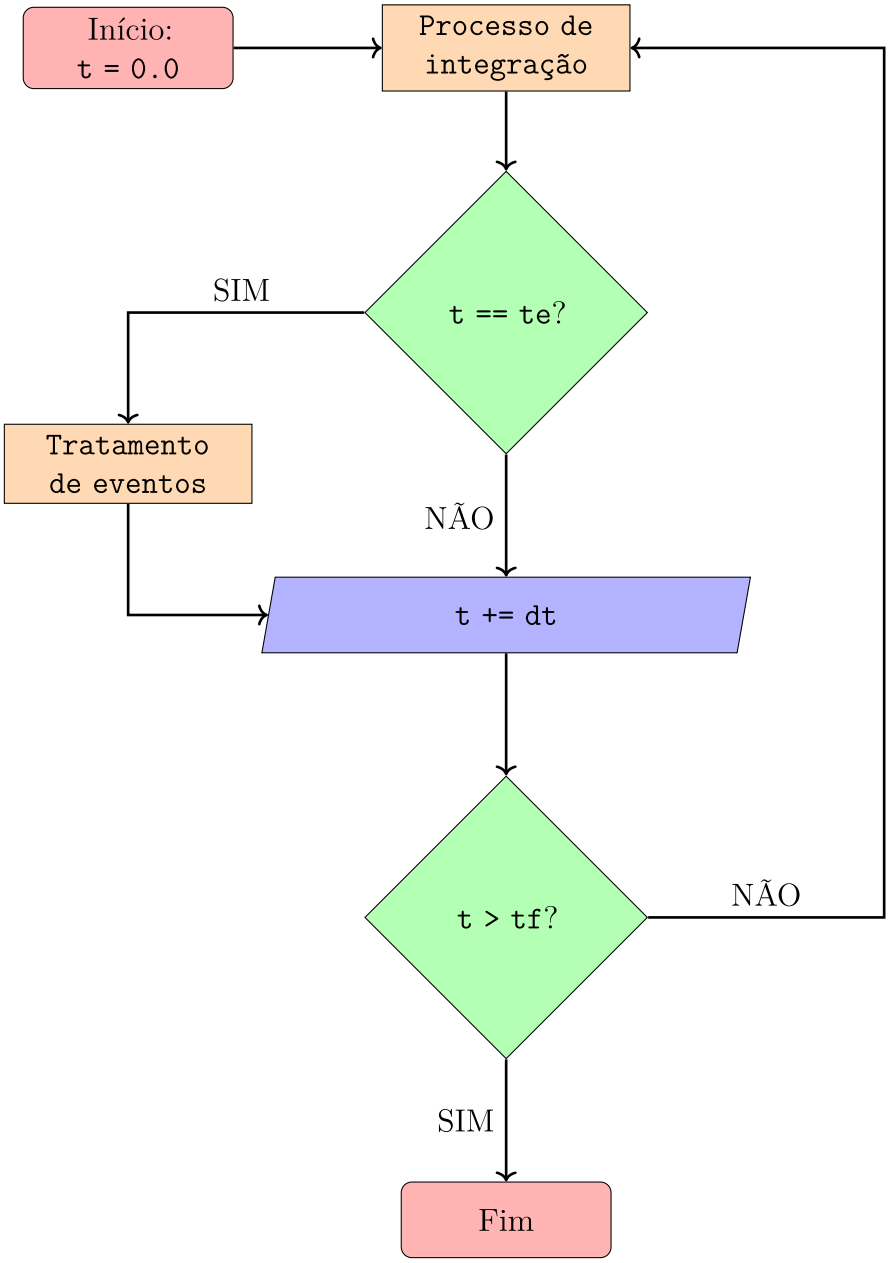
Essas três etapas da simulação no tempo da FMU do tipo *Model Exchange* estão resumidas no fluxograma da Figura 4.

Após executar o Algoritmo 3 e o Algoritmo 4 dentro de um único *script*, os vetores de armazenamento x, u e y estão prontos para serem apresentados graficamente. Com isso, tem-se na Figura 5(a) o gráfico das variáveis do modelo.

Para validar os resultados obtidos pelo ambiente *Python*, utiliza-se como referência o vetor x obtido pelo OMEdit, onde o sistema teste foi criado originalmente. Ao importar esse vetor para o *Python* através do pacote `DyMat`, tem-se na Figura 5(b) a comparação entre a variável de estado obtida pelo OMEdit e pela FMU do tipo ME.

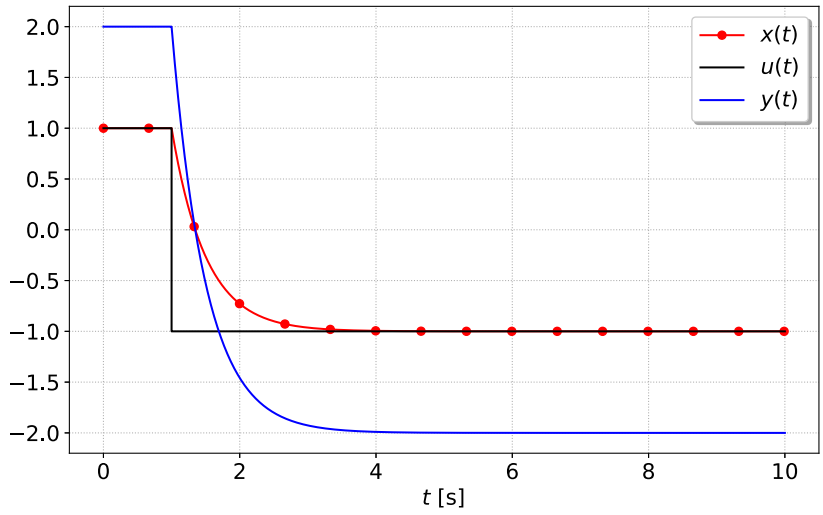
Com base na Figura 5(b), o erro absoluto máximo entre os resultados está na ordem de 10^{-6} e todos os instantes de tempo exigem apenas 1 iteração para a convergência.

Figura 4 – Fluxograma da simulação no tempo da FMU do tipo *Model Exchange*.

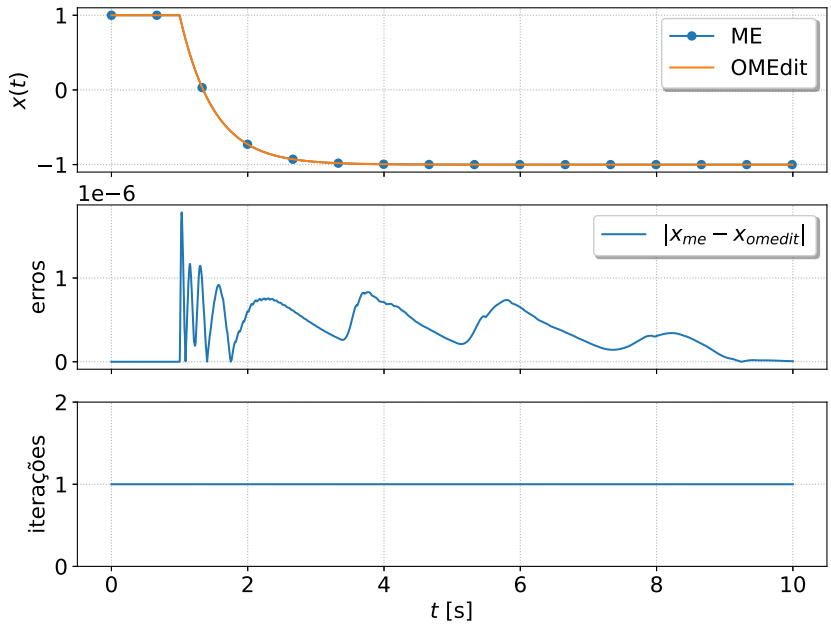


Fonte: Elaborada pelo autor (2023)

Figura 5 – Gráficos obtidos pela implementação da FMU do tipo *Model Exchange* no ambiente *Python*.



(a) Variáveis do modelo.



(b) Comparação entre a variável de estado $x(t)$ obtida pelo OMEdit e pela FMU do tipo ME.

Fonte: Elaborada pelo autor (2024)

Portanto, conforme esperado, os resultados da FMU acoplada ao *Python* são suficientemente próximos aos do OMEdit, o que valida a implementação da FMU do tipo *Model Exchange* em conjunto com o integrador trapezoidal externo descrito no Apêndice C.

2.5 IMPLEMENTAÇÃO DA FMU PARA MODELAGEM DE MÁQUINAS ELÉTRICAS

Com a validação da implementação da FMU do tipo ME dentro do ambiente *Python* no sistema operacional *Linux*, pode-se desenvolver FMUs que contenham os modelos das máquinas elétricas que devem ser acopladas à rede, assim como seus controladores, reguladores automáticos de tensão, reguladores de velocidade e estabilizadores de potência.

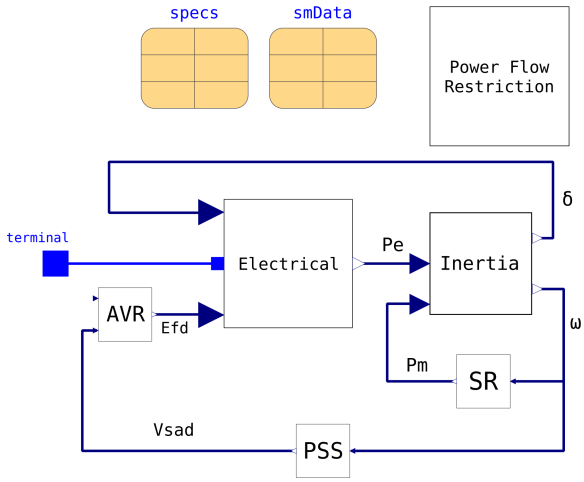
Neste trabalho, a modelagem de máquina síncrona escolhida para ser compactada em uma FMU é a utilizada para as unidades geradoras do sistema de transmissão de 11 barras, cujos detalhes são apresentados em (KUNDUR, 1994) e no Apêndice D. Além disso, a modelagem escolhida (denominada como `Generic_Machine_Kundur`) também pode ser encontrada como parte da biblioteca OmniPES, onde são definidas diferentes modelagens de equipamentos de sistemas de potência através da linguagem Modelica, conforme descrito em (TOMIM; HENRIQUES; PASSOS FILHO, 2024). Sabendo disso, tem-se na Figura 6 a interface gráfica do modelo de máquina síncrona.

A Figura 6(a) resume os blocos principais que compõem o modelo da máquina, a saber, parte elétrica (`Electrical`), parte mecânica (`Inertial`), regulador de tensão (`AVR`), regulador de velocidade (`SR`) e estabilizador de potência (`PSS`). O bloco `Power Flow Restriction` auxilia na definição da condição inicial de operação da máquina, enquanto os demais blocos contém parâmetros elétricos e definições do tipo de barra a ser utilizada para uma determinada máquina no fluxo de potência.

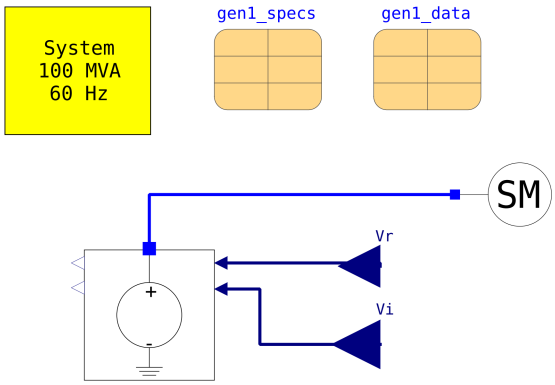
A Figura 6(b) ilustra a estrutura do modelo da máquina compilada na FMU. Nota-se a máquina conectada a uma fonte de tensão controlada, que permite a interface desse modelo com a rede elétrica. As entradas da FMU consistem na tensão terminal nas suas componentes real (`Vr`) e imaginária (`Vi`). No cálculo realizado pela FMU, a tensão sub-transitória da máquina é obtida a partir das condições internas do modelo e da tensão terminal definida pela rede elétrica. Esta tensão, junto com a reatância sub-transitória da máquina, compõe o equivalente de Norton conectado à rede, conforme mostrado na Figura 6(c).

Descrito o modelo de máquina síncronas adotado para esta dissertação, a sua compactação em forma de uma FMU pode ser realizada pelo Algoritmo 2, apresentado anteriormente neste capítulo. Outro recurso que também pode ser reutilizado é a biblioteca `PyFMI`, com o objetivo de possibilitar a comunicação entre o modelo de máquinas e o restante da simulação dentro do programa em *Python* desta dissertação. Essa comunicação é aprofundada no Capítulo 3.

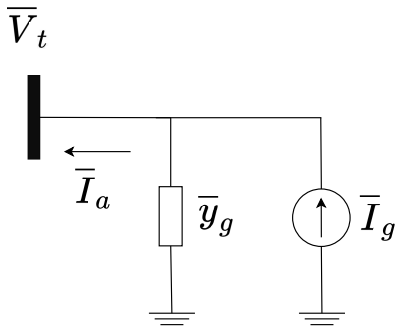
Figura 6 – Máquina síncrona modelada nas FMUs.



(a) Modelo genérico de máquina síncrona.



(b) Máquina síncrona com interface.



(c) Equivalente de Norton da máquina.

Fonte: Elaborada pelo autor (2023)

2.6 CONCLUSÕES PARCIAIS

Neste capítulo, foram apresentados os conceitos pertinentes da interface FMI 2.0 para a implementação da FMU. Foram discutidas algumas vantagens que incentivam a adoção dessa ferramenta para a modelagem de diferentes sistemas de equações. Dentre essas vantagens, tem-se a garantia de sigilo e a eficiência computacional. Em seguida, definiu-se que a FMU escolhida para esta dissertação foi o do tipo *Model Exchange*, ou Intercâmbio de Modelos, pois não apresenta atraso de informações intercambiadas da forma que ocorre com a FMU do tipo *Co-Simulation*. Dentre as propriedades particulares da FMU do tipo ME, uma das principais que foram citadas foi a ausência de um solucionador dentro do arquivo compactado. Devido a esse cenário, houve a necessidade de criação de um solucionador externo escrito em *Python* e baseado no método trapezoidal.

Em seguida, o solucionador criado (denominado `TrapezoidalIntegrator`) e uma FMU baseada em um modelo de EDO de primeira ordem foram inseridos em um mesmo teste de simulação em linguagem *Python*. A execução desse teste indicou os resultados desejados, mostrando que tanto a rotina de integração trapezoidal quanto a forma de importação e leitura da FMU foram implementados de forma correta.

Após mostrar o teste bem sucedido de um modelo simples, este capítulo se encerrou introduzindo um modelo de máquinas elétricas já testado na literatura, denominado `Generic_Machine_Kundur`, pertencente à biblioteca OmniPES. Em relação ao que foi testado neste capítulo, esse novo modelo possui maior complexidade, isto é, envolve múltiplas variáveis e diversas equações diferenciais, conforme equipamentos dinâmicos comumente encontrados em sistemas elétricos de potência. Sabendo disso, os próximos capítulos consistem em inserir a FMU do tipo ME em uma simulação computacional baseada no método alternado implícito, com o objetivo de verificar o potencial de representar modelos dinâmicos de maior complexidade.

3 FUNDAMENTOS DO MÉTODO ALTERNADO

Um sistema elétrico pode ser caracterizado através de um sistema de equações algébrico-diferenciais como denotado em (3.1).

$$\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}, \mathbf{v}\right) \tag{3.1a}$$

$$\mathbf{Y} \cdot \mathbf{v} = \mathbf{i}\left(\mathbf{x}, \mathbf{v}\right) \tag{3.1b}$$

A equação (3.1a) descreve o comportamento de equipamentos dinâmicos (geradores, cargas dinâmicas, elementos FACTS, etc.) e seus controladores associados. O conjunto (3.1b) consiste em um sistema de equações algébricas que descrevem as restrições impostas pela rede de transmissão passiva, assim como cargas funcionais dependentes da tensão. Comumente, esta rede é representada através da matriz de admitâncias nodais ($\mathbf{Y}$). Nessas equações, $\mathbf{x}$ é o vetor de variáveis dinâmicas (ou variáveis de estado), $\mathbf{v}$ é o vetor de tensões nodais e $\mathbf{i}$ é o vetor de correntes injetadas nodais (TOMIM, 2009).

Para a solução numérica do sistema sintetizado em (3.1a), pode-se empregar o método de integração trapezoidal, devido à sua consolidação na área da engenharia e sua estabilidade numérica (DOMMEL; SATO, 1972). Esse método consiste na aproximação de uma integral para um passo de tempo através da área de um trapézio, conforme descrito no Apêndice B. Vale ressaltar também que o método trapezoidal é adotado pelo programa ANATEM (ELETROBRAS CEPEL, 2020), que serve como referência para a validação dos resultados obtidos no Capítulo 5.

Com esses conceitos, a equação (3.1a) é discretizada conforme mostrado em (3.2).

$$\begin{aligned} \mathbf{x}(t) &= \mathbf{x}(t-\Delta t) + \int_{t-\Delta t}^{t} \mathbf{f}\Big(\mathbf{x}(\tau), \mathbf{v}(\tau)\Big) d\tau \approx \\ &\approx \mathbf{x}(t-\Delta t) + \frac{1}{2}\Big[\mathbf{f}(t) + \mathbf{f}(t-\Delta t)\Big]\Delta t \end{aligned} \tag{3.2}$$

Separando os termos presentes e os termos passados da equação (3.2), tem-se a nova equação de $\mathbf{x}(t)$ apresentada em (3.3a). Nessa equação, tem-se o vetor de termos históricos $\mathbf{x}_h(t)$, correspondentes às funções das variáveis de estado do instante imediatamente anterior $t - \Delta t$. Portanto, o vetor $\mathbf{x}_h(t)$ é mantido fixo durante a solução do instante t e precisa ser atualizado quando a solução convergente é encontrada.

$$\mathbf{x}(t) = \frac{\Delta t}{2}\,\mathbf{f}(t) + \mathbf{x}_h(t) \tag{3.3a}$$

$$\mathbf{x}_h(t) = \mathbf{x}(t-\Delta t) + \frac{\Delta t}{2}\,\mathbf{f}(t-\Delta t) \tag{3.3b}$$

Combinando as equações (3.1b), (3.3a) e (3.3b), obtém-se o modelo discretizado do sistema elétrico de potência que precisa ser solucionado a cada instante da simulação do regime transitório.

Para a solução desse sistema algébrico-diferencial, pode-se utilizar técnicas de método simultâneo ou alternado, por exemplo. Neste trabalho, emprega-se o método iterativo alternado, isto é, a solução do subsistema dos equipamentos dinâmicos é realizada separadamente do subsistema algébrico das equações da rede elétrica, até que a convergência na interface dos modelos seja verificada. Dessa forma, cada subsistema pode ser tratado por ferramentas computacionais distintas e adequadas, possibilitando um desempenho computacional satisfatório.

Como o método adotado para a simulação no tempo alterna entre as soluções do modelo de máquinas e as da rede elétrica de transmissão, pode ser denominado como método alternado do conjunto modelo-rede, cuja descrição é realizada na Seção 3.1.

3.1 MÉTODO ALTERNADO DO CONJUNTO MODELO-REDE

Antes da implementação do método alternado para a simulação no tempo de um sistema elétrico, é necessário calcular o seu ponto de operação em regime permanente. Para isso, este trabalho adota um método consolidado na literatura, que é o método iterativo de *Newton-Raphson*, explicado no Apêndice E e utilizado para a convergência das equações nodais da rede (mostradas em (3.1b)).

Como critério de convergência de cada iteração de *Newton-Raphson*, são escolhidas as potências ativas e reativas líquidas das barras da rede de transmissão, conforme a equação matricial em (3.4), onde se mostra a matriz jacobiana a ser recalculada a cada iteração de regime permanente.

$$\begin{bmatrix} \Delta\mathbf{P} \\ \\ \Delta\mathbf{Q} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial\mathbf{P}}{\partial\theta} & \dfrac{\partial\mathbf{P}}{\partial\mathbf{V}} \\ \\ \dfrac{\partial\mathbf{Q}}{\partial\theta} & \dfrac{\partial\mathbf{Q}}{\partial\mathbf{V}} \end{bmatrix} \cdot \begin{bmatrix} \Delta\theta \\ \\ \Delta\mathbf{V} \end{bmatrix} \tag{3.4}$$

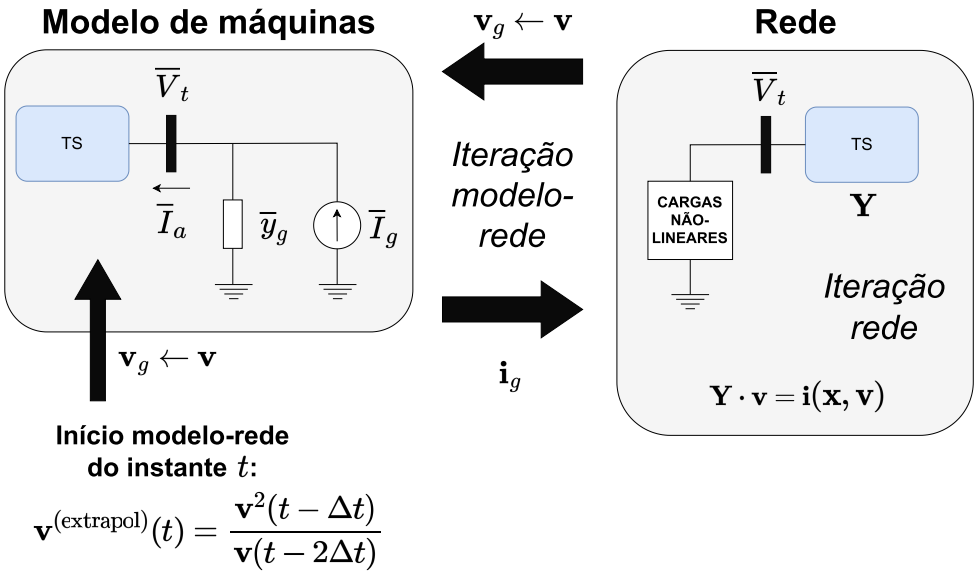
Encontrados os valores de regime permanente do sistema elétrico, tem-se o vetor inicial de variáveis de estado ($\mathbf{x}(0)$) do modelo de máquinas, bem como o vetor inicial de derivadas dos estados ($\mathbf{f}(0)$), que neste caso vez é um vetor nulo. Sabendo disso, tem-se o primeiro vetor de termos históricos das variáveis de estado ($\mathbf{x}_{h,0}$), mostrado em (3.5).

$$\mathbf{x}_{h,0} = \mathbf{x}(0) + \frac{\Delta t}{2}\,\mathbf{f}(0) \tag{3.5}$$

Agora, tem-se o método alternado da forma apresentada na Figura 7.

Neste trabalho, para a simulação do regime transitório, o cálculo do método alternado do instante atual (t) é iniciado pela extrapolação do vetor de tensões nodais ($\mathbf{v}^{(0)}$) da rede de transmissão. Conforme a equação (3.6), essa extrapolação é realizada com base nas tensões de um instante de tempo anterior ($t - \Delta t$) e de dois instantes anteriores

Figura 7 – Método alternado do regime transitório.



Fonte: Elaborada pelo autor (2023)

$(t - 2\Delta t)$. Possui a vantagem de aumento da velocidade de convergência (STOTT, 1979).

$$\begin{aligned} \mathbf{v}^{(0)}(t) &= \mathbf{v}^{(\text{extrapol})}(t) \\ &= \frac{\mathbf{v}^2(t-\Delta t)}{\mathbf{v}(t-2\Delta t)} \end{aligned} \tag{3.6}$$

Em seguida, executa-se um passo preditor para o vetor inicial de variáveis de estado ($\mathbf{x}^{(0)}$) do modelo de máquinas. Para isso, adota-se o método de Euler explícito, conforme explicado anteriormente na Seção 2.4. Com isso, tem-se a equação mostrada em (3.7).

$$\mathbf{x}^{(0)}(t) = \mathbf{x}(t-\Delta t) + \Delta t \cdot \mathbf{f}(t-\Delta t) \tag{3.7}$$

Com as tensões nodais e os estados estimados, inicia-se a primeira iteração "modelo-rede", ou iteração do conjunto modelo de máquinas mais rede de transmissão.

Descrição dos blocos da Figura 7:

(a) Inicialização do contador de iterações do conjunto modelo-rede: $i = 1$.

As etapas de (b) a (f) caracterizam a solução iterativa do conjunto modelo-rede.

(b) Armazenamento do mais recente vetor de tensões nodais da rede: mostrado na equação (3.8). Quando se tem $i = 1$, esse vetor corresponde às tensões extrapoladas.

$$\mathbf{v}^{(i-1)} = \begin{bmatrix} \overline{V}_{t,1}^{(i-1)} \\ \overline{V}_{t,2}^{(i-1)} \\ \vdots \\ \overline{V}_{t,k}^{(i-1)} \\ \vdots \\ \overline{V}_{t,n}^{(i-1)} \end{bmatrix} \tag{3.8}$$

(c) Extração das tensões terminais dos geradores: a partir do mais recente vetor $\mathbf{v}^{(i-1)}$, são extraídas as tensões terminais dos geradores ($\mathbf{v}_g^{(i-1)}$), conforme mostrado em (3.9).

$$\mathbf{v}_g^{(i-1)} \leftarrow \mathbf{v}^{(i-1)} \tag{3.9}$$

O vetor $\mathbf{v}_g^{(i-1)}$ serve como entrada para o bloco **Modelo de máquinas** da Figura 7.

(d) Bloco **Modelo de máquinas**: é o primeiro bloco do método alternado.

O mais recente vetor de tensões terminais dos geradores $\left(\mathbf{v}_g^{(i-1)}\right)$ serve como entrada para o modelo de máquinas que, neste trabalho, está compactado em uma FMU e é solucionado pelo método de integração trapezoidal, conforme explicado anteriormente na Seção 2.4. Com isso, a FMU entrega ao ambiente de simulação as tensões internas dos geradores $\left(\overline{E}_{int}^{(i)}\right)$, conforme mostrado em (3.10).

$$\overline{V}_g^{(i-1)} \xRightarrow[\text{máquinas da FMU}]{\text{Modelo de}} \overline{E}_{int}^{(i)} \tag{3.10}$$

Em seguida, o ambiente de simulação utiliza os valores de $\overline{E}_{int}^{(i)}$ da iteração atual para calcular as correntes de Norton correspondentes, conforme mostrado em (3.11).

$$\overline{I}_g^{(i)} = \overline{y}_g \cdot \overline{E}_{int}^{(i)} \tag{3.11}$$

Com o valor de $\overline{I}_g^{(i)}$ de cada gerador, tem-se o vetor de correntes de Norton da i-ésima iteração do conjunto modelo-rede ($\mathbf{i}_g^{(i)}$), conforme mostrado em (3.12). Caso

a k-ésima barra da rede não possua nenhuma unidade geradora, tem-se $\overline{I}_{g,k}^{(i)} = 0$.

$$\mathbf{i}_g^{(i)} = \begin{bmatrix} \overline{I}_{g,1}^{(i)} \\ \overline{I}_{g,2}^{(i)} \\ \vdots \\ \overline{I}_{g,k}^{(i)} \\ \vdots \\ \overline{I}_{g,n}^{(i)} \end{bmatrix} \tag{3.12}$$

Com isso, a saída do bloco **Modelo de máquinas** é o vetor $\mathbf{i}_g^{(i)}$, que serve como entrada para o bloco **Rede** da Figura 7.

(e) Bloco **Rede**: é o segundo bloco do método alternado.

O mais recente vetor de correntes de Norton dos geradores $\left(\mathbf{i}_g^{(i)}\right)$ serve como entrada para o cálculo iterativo da rede de transmissão. A cada iteração "rede" (ou iteração r), o vetor $\mathbf{i}_g^{(i)}$ se mantém constante.

Além disso, o mais recente vetor de tensões nodais da rede $\left(\mathbf{v}^{(r-1)}\right)$ é utilizado para duas finalidades: primeiro, atualizar a matriz de admitâncias nodais da rede $\left(\mathbf{Y}^{(r)}\right)$ em caso de baixas tensões e, segundo, calcular o vetor de correntes das cargas não lineares $\left(\mathbf{i}_l^{(r)}\right)$, conforme mostrado em (3.13).

$$\mathbf{v}^{(r-1)} \xRightarrow[\text{não lineares}]{\text{Cargas}} \mathbf{Y}^{(r)}, \mathbf{i}_l^{(r)} \tag{3.13}$$

Com os vetores de correntes dos geradores e das cargas, calcula-se o vetor de injeções de correntes nodais $\left(\mathbf{i}^{(r)}\right)$ através da equação (3.14).

$$\mathbf{i}^{(r)} = \mathbf{i}_g^{(i)} - \mathbf{i}_l^{(r)} \tag{3.14}$$

Em seguida, o vetor de tensões nodais $\left(\mathbf{v}^{(r)}\right)$ é calculado pelo método nodal, conforme a equação (3.15).

$$\mathbf{Y}^{(r)} \cdot \mathbf{v}^{(r)} = \mathbf{i}^{(r)} \tag{3.15}$$

A solução iterativa da rede é explicada em maiores detalhes na Subseção **3.1.1**.

Após a convergência da rede, a saída do bloco **Rede** é um novo vetor $\mathbf{v}^{(i)}$. Com isso, encerra-se a iteração "modelo-rede".

(f) Teste de convergência da iteração "modelo-rede": com o intuito de verificar se são necessárias mais iterações "modelo-rede", deve-se calcular a diferença entre o vetor

da iteração atual i e o vetor anterior das tensões nodais, conforme mostrado na equação (3.16).

$$\Delta \mathbf{v}^{(i)} = \mathbf{v}^{(i)} - \mathbf{v}^{(i-1)} \tag{3.16}$$

O processo iterativo do método alternado é encerrado quando os erros absolutos e relativos do vetor atendem às relações apresentadas em (3.17).

$$\begin{cases} \max \left| \Delta \mathbf{v}^{(i)} \right| \leq \varepsilon_{abs} \\ \max \left| \Delta \mathbf{v}^{(i)} \right| \leq \max \left| \mathbf{v}^{(i-1)} \right| \cdot \varepsilon_{rel} \end{cases} \tag{3.17}$$

Caso não sejam atendidas, realiza-se o incremento do contador de iterações do conjunto modelo-rede ($i = i + 1$) e o mais recente vetor $\mathbf{v}$ serve novamente como entrada para o bloco **Modelo de máquinas** da Figura 7, onde se inicia uma nova iteração "modelo-rede" no item (b).

Após a convergência do conjunto modelo-rede do instante t, tem-se o vetor de variáveis de estado ($\mathbf{x}$) do modelo de máquinas, bem como o vetor de derivadas dos estados ($\mathbf{f}$) correspondentes à solução convergente. Sabendo disso, atualiza-se o vetor de termos históricos das variáveis de estado ($\mathbf{x}_h$) para que seja utilizado pelo instante de tempo seguinte. Essa atualização está mostrada na equação (3.18).

$$\mathbf{x}_h(t) = \mathbf{x} + \frac{\Delta t}{2} \mathbf{f} \tag{3.18}$$

3.1.1 Solução iterativa da rede

Esta subseção tem como objetivo detalhar a solução iterativa da rede, correspondente ao bloco **Rede** da Figura 7.

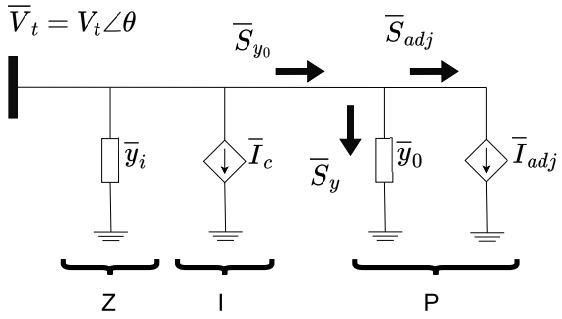
(i) Inicialização do contador de iterações da rede: $r = 1$.

As etapas de (ii) a (vi) caracterizam a solução iterativa da rede.

(ii) Armazenamento do mais recente vetor de tensões nodais da rede: mostrado na equação (3.19).

$$\mathbf{v}^{(r-1)} = \begin{bmatrix} \overline{V}_{t,1}^{(r-1)} \\ \overline{V}_{t,2}^{(r-1)} \\ \vdots \\ \overline{V}_{t,k}^{(r-1)} \\ \vdots \\ \overline{V}_{t,n}^{(r-1)} \end{bmatrix} \tag{3.19}$$

Figura 8 – Modelagem de carga ZIP completa.



Fonte: Elaborada pelo autor (2023)

(iii) Varredura das cargas não lineares da rede: neste trabalho, as cargas não lineares são representadas pela modelagem ZIP (explicada no Apêndice F).

Nesta etapa, a tensão de cada barra de carga de modelo ZIP é comparada a um valor V_{mn}. Neste trabalho, o valor adotado de V_{mn} é 70 %, que é o valor padrão adotado pelo programa ANATEM. Com isso, tem-se dois cenários possíveis de tensão de carga:

- Barra de carga com tensão $V_{t,k}^{(r-1)} \geq V_{mn}$: no primeiro cenário, tem-se a carga ZIP completa mostrada na Figura 8. Essa modelagem visa representar a carga para uma faixa de tensões próximas à nominal, isto é, maiores ou iguais a um determinado valor V_{mn}.

  Com o objetivo de economizar esforço computacional, as admitâncias $\overline{y}_i$ e $\overline{y}_0$ são previamente adicionadas à matriz de admitâncias nodais ($\mathbf{Y}$), isto é, logo depois do cálculo de regime permanente. Dessa forma, a matriz $\mathbf{Y}$ permanece inalterada quando as tensões nodais são próximas à nominal (isto é, acima de 0,7 [pu]) durante o regime transitório.

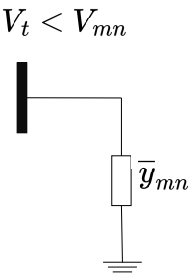
  Por outro lado, a cada iteração da rede, o vetor de correntes de carga $\left(\mathbf{i}_l^{(r)}\right)$ precisa ser inicializado como um vetor nulo e atualizado ao longo da varredura das cargas. As correntes $\overline{I}_c^{(r)}$ e $\overline{I}_{adj}^{(r)}$ da Figura 8 são recalculadas a cada iteração da rede, porque dependem de valores que variam a cada iteração, isto é, as tensões nodais da rede $\left(\overline{V}_t^{(r-1)}\right)$, conforme mostrado no Apêndice F.

  Realizados os cálculos, os valores de $\overline{I}_c^{(r)}$ e $\overline{I}_{adj}^{(r)}$ são adicionados ao k-ésimo termo do vetor de correntes de carga $\left(\mathbf{i}_l^{(r)}\right)$, conforme mostrado em (3.20).

$$\mathbf{i}_l^{(r)}(k) = \mathbf{i}_l^{(r)}(k) + \overline{I}_{c,k}^{(r)} + \mathbf{i}_{adj,k}^{(r)} \tag{3.20}$$

- Barra de carga com tensão $V_{t,k}^{(r-1)} < V_{mn}$: no segundo cenário, despreza-se a modelagem completa da Figura 8 e considera-se apenas uma impedância

Figura 9 – Modelagem de carga do tipo impedância constante para baixas tensões.



Fonte: Elaborada pelo autor (2023)

constante $\overline{y}_{mn}$, conforme mostrado na Figura 9.

A admitância $\overline{y}_{mn}$ é previamente calculada com base em valores constantes de regime permanente e, em caso de baixas tensões, precisa ser adicionada à matriz de admitâncias nodais da rede de transmissão ($\mathbf{Y}$). Além disso, deve-se também remover as admitâncias $\overline{y}_i$ e $\overline{y}_0$, pois ambas correspondem à modelagem completa. Com isso, o termo matricial $\mathbf{Y}^{(r)}(k,k)$ é incrementado da forma mostrada em (3.21).

$$\mathbf{Y}^{(r)}(k,k) = \mathbf{Y}^{(r)}(k,k) + \overline{y}_{mn,k} - (\overline{y}_{i,k} + \overline{y}_{0,k}) \tag{3.21}$$

Além disso, como a modelagem da Figura 9 não possui nenhuma fonte de corrente, o k-ésimo termo do vetor $\mathbf{i}_l^{(r)}$ se mantém igual a zero.

(iv) Cálculo do vetor de injeções de correntes nodais: conforme dito anteriormente nesta seção, a entrada do bloco **Rede** é o vetor de correntes dos geradores da i-ésima iteração do conjunto modelo-rede, isto é, o vetor $\mathbf{i}_g^{(i)}$.

Depois de calcular o vetor de correntes de carga $\left(\mathbf{i}_l^{(r)}\right)$ na etapa (iii), calcula-se o vetor de injeções de correntes nodais $\left(\mathbf{i}^{(r)}\right)$ através da equação (3.22).

$$\mathbf{i}^{(r)} = \mathbf{i}_g^{(i)} - \mathbf{i}_l^{(r)} \tag{3.22}$$

(v) Atualização do vetor de tensões nodais: atualizados os termos $\mathbf{Y}^{(r)}$ e $\mathbf{i}^{(r)}$ em função dos dois cenários de carga ZIP, resolve-se a equação matricial em (3.23).

$$\mathbf{Y}^{(r)} \cdot \mathbf{v}^{(r)} = \mathbf{i}^{(r)}\left(\mathbf{x}, \mathbf{v}^{(r-1)}\right) \tag{3.23}$$

Para fins de desempenho computacional, a equação matricial de (3.23) é resolvida por fatoração LU.

(vi) Teste de convergência da iteração "rede": com o intuito de verificar se são necessárias mais iterações "rede", deve-se calcular a diferença entre o vetor da iteração atual r e

o vetor anterior das tensões nodais, conforme mostrado na equação (3.24).

$$\Delta \mathbf{v}^{(r)} = \mathbf{v}^{(r)} - \mathbf{v}^{(r-1)} \tag{3.24}$$

O processo iterativo da rede é encerrado quando os erros absolutos e relativos do vetor atendem às relações apresentadas em (3.25).

$$\begin{cases} \max \left| \Delta \mathbf{v}^{(r)} \right| \leq \varepsilon_{abs} \\ \max \left| \Delta \mathbf{v}^{(r)} \right| \leq \max \left| \mathbf{v}^{(r-1)} \right| \varepsilon_{rel} \end{cases} \tag{3.25}$$

Caso não sejam atendidas, realiza-se o incremento do contador de iterações da rede ($r = r + 1$), os incrementos da matriz $\mathbf{Y}$ (equação (3.21)) são desfeitos e o mais recente vetor $\mathbf{v}$ inicia uma nova iteração "rede" no item (ii).

Após a convergência da rede, os incrementos da matriz $\mathbf{Y}$ (equação (3.21)) são desfeitos e a saída do bloco **Rede** é um novo vetor $\mathbf{v}^{(r)}$.

3.1.2 Tratamento de eventos

Com o objetivo de simular eventos (por exemplo, aplicação e remoção de curto-circuito, abertura de linha, etc.) que podem ocorrer na rede, a matriz $\mathbf{Y}$ precisa ser adequadamente alterada nos instantes de tempo correspondentes, chamados de t_e. Cada instante de evento exige o cálculo de duas soluções: uma em t_e^- (antes do evento) e outra em t_e^+ (após o evento) (DOMMEL; SATO, 1972; SILVA, J. P. A., 2016).

Com isso, pode ocorrer uma descontinuidade nas variáveis algébricas, como por exemplo as tensões nodais da rede ($\mathbf{v}$), conforme mostrado em (3.26).

$$\mathbf{v}(t_e^+) \neq \mathbf{v}(t_e^-) \tag{3.26}$$

Por outro lado, deve-se manter a continuidade das variáveis de estado ($\mathbf{x}$), como por exemplo as correntes de Norton dos geradores ($\mathbf{i}_g$), que são associadas aos ângulos internos das máquinas elétricas. Essa propriedade contínua de $\mathbf{x}$ e $\mathbf{i}_g$ diante de um evento no instante t_e está apresentada em (3.27).

$$\begin{cases} \mathbf{x}(t_e^+) = \mathbf{x}(t_e^-) \\ \mathbf{i}_g(t_e^+) = \mathbf{i}_g(t_e^-) \end{cases} \tag{3.27}$$

Descrição das etapas do tratamento de eventos:

(i) Armazenamento dos resultados correspondentes à iminência do evento: antes do tratamento de eventos propriamente dito, armazena-se o vetor $\mathbf{v}(t_e^-)$, que é o mais recente vetor de tensões nodais convergente do método alternado.

(ii) Modificação da rede de acordo com o evento: por exemplo, se ocorrer um curto-circuito na barra k da rede no instante t_e, realiza-se a modificação correspondente no termo matricial $\mathbf{Y}(k,k)$.

(iii) Cálculo da estimativa inicial das tensões nodais do instante t_e^+: com a modificação da matriz $\mathbf{Y}$, as tensões nodais podem mudar drasticamente. Portanto, para acelerar a convergência do tratamento de eventos, não se utiliza a extrapolação da equação (3.6). Em vez disso, utiliza-se o método nodal para o cálculo da estimativa inicial de $\mathbf{v}(t_e^+)$, conforme a equação matricial em (3.28).

$$\mathbf{Y}(t_e^+) \cdot \mathbf{v}^{(0)}(t_e^+) = \mathbf{i}(t_e^-) \tag{3.28}$$

onde:

$$\begin{cases} \mathbf{Y}(t_e^+): & \text{Matriz de admitâncias nodais modificada de acordo com o evento.} \\ \mathbf{v}^{(0)}(t_e^+): & \text{Estimativa inicial do vetor de tensões nodais no instante de ocorrência do evento.} \\ \mathbf{i}(t_e^-): & \text{Vetor de injeções de correntes nodais da mais recente solução convergente do método alternado.} \end{cases}$$

Para fins de desempenho computacional, a equação matricial de (3.28) é resolvida por fatoração LU.

(iv) Solução da rede: além de $\mathbf{v}^{(0)}(t_e^+)$, o mais recente vetor de correntes de Norton dos geradores ($\mathbf{i}_g(t_e^+)$) serve como entrada para o cálculo iterativo da rede de transmissão. A solução iterativa da rede é explicada em maiores detalhes na Subseção **3.1.1**.

Após a convergência da rede no instante t_e^+, a saída do bloco **Rede** é um novo vetor $\mathbf{v}(t_e^+)$.

(v) Extração das tensões terminais dos geradores: a partir de $\mathbf{v}(t_e^+)$, são extraídas as tensões terminais dos geradores ($\mathbf{v}_g(t_e^+)$), conforme mostrado em (3.29).

$$\mathbf{v}_g(t_e^+) \leftarrow \mathbf{v}(t_e^+) \tag{3.29}$$

(vi) Atualização das derivadas do modelo de máquinas: conforme mostrado em (3.26), um evento na rede acarreta uma descontinuidade das tensões nodais da rede, inclusive as tensões terminais dos geradores. Portanto, a partir do vetor $\mathbf{v}_g(t_e^+)$, atualiza-se o vetor de derivadas dos estados ($\mathbf{f}(t_e^+)$) do modelo de máquinas, conforme mostrado em (3.30).

$$\mathbf{v}_g(t_e^+) \xRightarrow[\text{máquinas da FMU}]{\text{Modelo de}} \mathbf{f}(t_e^+) \tag{3.30}$$

(vii) Atualização dos termos históricos: com o vetor de variáveis de estado ($\mathbf{x}(t_e^+)$) e o vetor de derivadas dos estados ($\mathbf{f}(t_e^+)$) do instante t_e^+, atualiza-se o vetor de termos

históricos das variáveis de estado ($\mathbf{x}_h(t_e^+)$) para que seja utilizado pelo instante de tempo seguinte. Essa atualização está mostrada na equação (3.31).

$$\mathbf{x}_h(t_e^+) = \mathbf{x}(t_e^+) + \frac{\Delta t}{2}\,\mathbf{f}(t_e^+) \tag{3.31}$$

3.2 IMPLEMENTAÇÃO DO MÉTODO ALTERNADO AO PROGRAMA DE ESTABILIDADE

Após detalhar na Seção 3.1 o método alternado entre a solução do modelo de máquinas e a solução da rede de transmissão com a presença de eventos, o protótipo do programa de estabilidade pode ser montado em linguagem *Python* e executado em conjunto com modelos baseados em FMUs. Nesta seção, são apresentados os pseudocódigos do programa principal desenvolvido, bem como as funções que o compõem.

3.2.1 Programa principal

Antes da implementação do método alternado propriamente dito, deve-se exportar e inicializar a FMU a ser utilizada para a representação dos geradores da rede de transmissão.

Neste trabalho, a modelagem de máquina síncrona escolhida para ser compactada em uma FMU é a utilizada para as unidades geradoras do sistema de transmissão de 11 barras, cujos detalhes são encontrados em (KUNDUR, 1994), no Apêndice D e na biblioteca OmniPES, conforme explicado anteriormente na Seção 2.5.

Com a modelagem escolhida, utiliza-se novamente o Algoritmo 2 (apresentado na Seção 2.4) para exportar a FMU correspondente, que por sua vez recebe o nome de `Generic_Machine_Kundur.fmu`. Em seguida, realiza-se a sua importação conforme mostrado no Algoritmo 5.

Primeiro, o módulo `PyFMI` importa a classe `FMUModelME2` da mesma forma mostrada na Seção 2.4. Em seguida, os termos `fmu_me` e `integrator` são inicializados como listas vazias (linhas 9 e 10). Através de cada laço de uma estrutura de `for`, essas listas armazenam as instâncias dos modelos e dos integradores de cada unidade geradora.

A primeira etapa do `for` é a importação da FMU através da classe `FMUModelME2` (linha 13), cuja entrada é o caminho para o arquivo `Generic_Machine_Kundur.fmu`. Com isso, o modelo da FMU é atribuído à g-ésima posição da lista `fmu_me`. Em seguida, utiliza-se o comando `setup_experiment` (linha 15) para atribuir à lista os parâmetros de simulação (tolerância de convergência igual a `1e-4`, instante inicial igual a `0.0` [s] e instante final igual a `10.0` [s]).

Agora, o g-ésimo modelo da FMU do tipo *Model Exchange* está pronto para entrar no modo de inicialização, onde é necessário definir os parâmetros de regime permanente através do comando `set` (linhas 18 a 20). Esses parâmetros são as potências ativas (`P0`) e

Algoritmo 5 – Importação das FMUs para os geradores da rede de transmissão.

```
1  from pyfmi.fmi import FMUModelME2
2  from trapezoidal import TrapezoidalIntegrator
3  import numpy as np
4  from scipy import linalg

5  # Parâmetros da simulação (passo de integração e tempo final):
6  dt = 1e-3
7  tf = 10.0

8  # Inicialização do objeto de armazenamento dos parâmetros da FMU e dos integradores para
   ↪  cada gerador da rede de transmissão:
9  fmu_me = []
10 integrator = []

11 for g in range(num_gen):

12   # Importação da FMU para o g-ésimo gerador:
13   fmu_me[g] = FMUModelME2("/diretorio/Generic_Machine_Kundur.fmu", log_level=2)
14   fmu_me[g].instantiate()
15   fmu_me[g].setup_experiment(tolerance=1e-4, start_time=0.0, stop_time=tf)

16   # Modo de inicialização da FMU: valores iniciais das potências ativas e tensões nodais do
     ↪  modelo de máquinas.
17   fmu_me[g].enter_initialization_mode()
18   fmu_me[g].set("P0", P0[g])
19   fmu_me[g].set("Vreal", Real(vg_0[g]))
20   fmu_me[g].set("Vimag", Imag(vg_0[g]))
21   fmu_me[g].exit_initialization_mode()

22   # Inicialização do integrador para o g-ésimo gerador: método trapezoidal.
23   integrator[g] = TrapezoidalIntegrator(fmu_me[g], dt, tf , atol=1e-4, rtol=1e-2)

24   # Modo de tempo contínuo da FMU:
25   fmu_me[g].enter_continuous_time_mode()
26   fmu_me[g].time = 0.0
```

Fonte: Elaborado pelo autor (2024)

as tensões terminais iniciais (`vg_0`) dos geradores, já que se adota a modelagem do tipo PV durante o regime permanente.

Após o modo de inicialização da FMU, deve-se agora instanciar o solucionador externo criado em *Python* para este trabalho. Seu nome é `TrapezoidalIntegrator` e sua explicação está apresentada na Seção 2.4. Seus parâmetros de entrada são: a lista dos objetos contendo os modelos da FMU (`fmu_me`), passo de integração (`dt = 1e-3` [s]), tempo final (`tf = 10.0` [s]), tolerância absoluta (`atol = 1e-4` [pu]) e tolerância relativa (`rtol = 1e-2` %).

Com a instância da classe `TrapezoidalIntegrator`, tem-se o g-ésimo integrador da lista `integrator` (linha 23), cujo termo histórico inicial é definido com base nas condições iniciais conhecidas do modelo.

Por fim, as últimas etapas da inicialização do modelo da FMU consistem na ativação do modo contínuo através do comando `enter_continuous_time_mode` (linha 25) e na definição do instante de tempo inicial dos modelos como igual a `0.0` [s] (linha 26).

Agora que os modelos das unidades geradoras estão prontos para a simulação no tempo em conjunto com a rede de transmissão, o método alternado do conjunto modelo-rede é realizado conforme o Algoritmo 6.

A simulação no tempo é realizada através de uma estrutura de `while`, onde cada laço corresponde a um instante de tempo. Portanto, para o primeiro laço, o instante atual da simulação (`t`) é definido como igual a `0.0` [s], enquanto o último laço define o valor de `t` como igual ao instante final `tf`.

Antes de iniciar o método alternado propriamente dito, inicializam-se três variáveis: primeiro, o instante de tempo da simulação (`t = 0.0` [s] - linha 2), que é incrementado a cada laço do `while`. Depois, a variável binária `flag_step_model = True` (linha 3), que define se será realizado o processo de integração do modelo de máquinas no instante correspondente. Por último, a variável binária `flag_extrapol = True` (linha 4), que define se será realizada a extrapolação das tensões nodais da rede.

Entrando no laço de simulação no tempo (início na linha 8), a primeira etapa é a extrapolação do vetor de tensões nodais da rede (`v` - linha 12) a ser utilizado como entrada inicial para as iterações do conjunto modelo-rede. Como requer vetores de dois instantes anteriores, há dois casos nos quais a extrapolação é suspensa: nos dois instantes iniciais da simulação no tempo e no instante imediatamente posterior a cada tratamento de eventos. A suspensão da extrapolação é definida pela variável binária `flag_extrapol`, que inicialmente é igual a `True`.

A próxima etapa da simulação no tempo é a varredura dos geradores, realizada através de uma estrutura de `for` (linha 15). Nesta etapa, o mais recente valor de instante `t` é atribuído ao g-ésimo modelo da lista `fmu_me` (linha 16). Em seguida, o passo preditor do g-ésimo modelo é executado pelo comando `predictor` (linha 18), com o objetivo de facilitar a convergência do processo de integração. Os estados resultantes da predição são atribuídos ao g-ésimo modelo de `fmu_me` (linha 19).

Em seguida, são inicializados os seguintes parâmetros do conjunto modelo-rede do instante `t`: contador de iterações igual a `iteration = 0` (linha 21) e variável binária de convergência do conjunto modelo-rede igual a `flag_model_converged = False` (linha 22). Com isso, as etapas de uma iteração "modelo-rede" são as seguintes:

(a) Armazenamento do mais recente vetor de tensões nodais da rede (linha 25): essencial para o teste de convergência do conjunto modelo-rede.

(b) Solução do modelo de máquinas compilado em uma FMU (linha 27): realizada através da função `model_solution` e explicada em maiores detalhes no Algoritmo 7. A solução do modelo retorna um vetor de correntes de Norton dos geradores (`ig`).

(c) Solução iterativa da rede de transmissão (linha 29): realizada através da função

Algoritmo 6 – Método alternado do conjunto modelo-rede.

```
# Inicializações:
t = 0.0                 # Instante de tempo da simulação.
flag_step_model = True  # Variável binária de avanço de um passo de integração do modelo.
flag_extrapol = True    # Variável binária de extrapolação das tensões nodais da rede.

# ============================================================
# ================= Simulação no tempo:
# ============================================================

while t <= tf:

  # ================= 1) Método alternado do conjunto modelo-rede:

  # ---------- 1.1) Extrapolação do vetor de tensões nodais da rede:
  if t >= 2*dt and flag_extrapol:
    v = v(t - dt)**2 / v(t - 2*dt)
  flag_extrapol = True

  # ---------- 1.2) Varredura dos geradores:
  for g in range(num_gen):

    fmu_me[g].time = t

    # Passo preditor do modelo de máquinas:
    integrator[g].predictor()
    fmu_me[g].continuous_states = integrator[g].x[:]

  # ---------- 1.3) Execução das iterações do conjunto modelo-rede:
  iteration = 0
  flag_model_net_converged = False
  while flag_model_net_converged == False:

    # (a) Armazenamento do mais recente vetor de tensões nodais da rede:
    v_prev = v

    # (b) Solução do modelo de máquinas compilado em uma FMU:
    ig = model_solution(fmu_me, integrator, v, flag_step_model)

    # (c) Solução iterativa da rede de transmissão:
    v = net_solution(Y, LU, piv, v, ig)

    # (d) Teste de convergência do conjunto modelo-rede:
    dv = np.abs(v - v_prev)
    flag_model_net_converged = np.max(dv) <= e_abs and np.max(dv) <= np.max(v_prev)*e_rel

    # (e) Incremento do contador de iterações:
    iteration += 1
    if iteration == 10:
      print("Erro do método iterativo do conjunto modelo-rede.")
      exit(1)

  # ---------- 1.4) Atualização dos termos históricos:
  for g in range(num_gen):
    integrator[g].update_history()

  # ================= 2) Tratamento de eventos:
  if t == te:
    Y, LU, piv, v = event_treatment(Y, v, i, ig)
    flag_extrapol = False

  # ================= 3) Avanço de tempo:
  t += dt
```

Fonte: Elaborado pelo autor (2024)

`net_solution` e explicada em maiores detalhes no Algoritmo 8. A solução da rede retorna um novo vetor de tensões nodais da rede de transmissão (`v`).

(d) Teste de convergência do conjunto modelo-rede (linha 32): realizado a partir da diferença entre o vetor da iteração atual (`v`) e o vetor anterior das tensões nodais (`v_prev`). O resultado do teste é armazenado dentro da variável binária de convergência denominada `flag_model_net_converged`.

(e) Incremento do contador de iterações (linha 34): incrementa-se o valor de `iteration` e verifica-se se foi atingido o limite de `10` iterações. Em caso afirmativo, o programa é interrompido e imprime uma mensagem de erro.

Após a convergência do método alternado do conjunto modelo-rede para o instante `t`, realiza-se uma função do integrador denominada `update_history` (linha 40), que consiste em atualizar os termos históricos a ser utilizados pelo instante de tempo seguinte.

Continuando o laço de simulação no tempo do instante atual `t`, a próxima etapa é o tratamento de eventos (linha 43), que é realizado quando o instante de tempo atual da simulação for igual a um instante de evento qualquer (`te`). Em caso afirmativo, essa etapa é realizada através da função `event_treatment`, que é explicada em maiores detalhes no Algoritmo 9. Como um evento da rede acarreta uma descontinuidade nas tensões nodais, deve-se atualizar a variável binária `flag_extrapol` como `False` (linha 44). Essa variável, por sua vez, volta a ser `True` depois de suspender a extrapolação no instante de tempo igual a `te + dt`.

A última etapa do laço é o avanço de tempo da simulação (linha 46), onde se realiza o incremento do instante `t` através do passo de integração `dt` e encerra-se o laço de tempo do `while`.

Agora que são conhecidas as etapas do método alternado do conjunto modelo-rede, as próximas subseções apresentam em detalhes as funções que compõem o programa principal do Algoritmo 6.

3.2.2 Função da solução do modelo

Conforme o programa principal da Subseção **3.2.1**, a solução do modelo de máquinas compilado em uma FMU é realizada através da função `model_solution`, mostrada no Algoritmo 7.

Como o vetor `v` armazena as tensões de todas as barras da rede de transmissão, a primeira etapa da função `model_solution` é a extração das mais recentes tensões terminais dos geradores através da função `gen_voltages` (linha 10), criada para este trabalho. Com isso, tem-se um novo vetor `vg`, que é utilizado durante a varredura dos geradores.

Algoritmo 7 – Função `model_solution`.

```
def model_solution(fmu_me, integrator, v, flag_step_model)

  # ----------- Entrada:
  # - fmu_me: lista de objetos do modelo da FMU do tipo Model Exchange.
  # - integrator: lista de objetos do integrador trapezoidal (criado em Python).
  # - v: vetor de tensões nodais da rede.
  # - flag_step_model: variável binária de avanço de um passo de integração do modelo.

  # ----------- Saída:
  # - ig: vetor de correntes de Norton dos geradores.

  # Extração das tensões terminais dos geradores:
  vg = gen_voltages(v)

  # ============================================================
  # ================= Solução do modelo de máquinas:
  # ============================================================

  # Varredura dos geradores:
  for g in range(num_gen):

    # Atribuição das tensões dos geradores ao g-ésimo modelo da lista fmu_me:
    fmu_me[g].set("Vreal", Real(vg[g]))
    fmu_me[g].set("Vimag", Imag(vg[g]))

    if flag_step_model:

      # ======== 1 - Método alternado (com avanço de um passo de integração):
      integrator[g].step()

      # Atribuição dos estados mais recentes ao g-ésimo modelo da lista fmu_me:
      fmu_me[g].continuous_states = integrator[g].x[:]

    else:
      # ======== 2 - Tratamento de eventos (sem avanço):

      # Derivadas das variáveis de estado:
      integrator[g].der_x = fmu_me[g].get_derivatives()

      # Variáveis de estado, que são contínuas entre eventos:
      integrator[g].x = fmu_me[g].continuous_states

    # Obtenção dos resultados do modelo:
    x[g] = fmu_me[g].get("x")
    ig[g] = Norton_model(x[g])

  return ig
```

Fonte: Elaborado pelo autor (2024)

A próxima etapa da solução do modelo de máquinas é a varredura dos geradores, realizada através de uma estrutura de `for` (linha 15). Nesta etapa, as componentes real e imaginária da mais recente tensão `vg[g]` são atribuídas ao g-ésimo modelo da lista `fmu_me` através do comando `set` (linhas 17 e 18). Em seguida, tem-se dois cenários possíveis:

1. Método alternado (início na linha 20): no primeiro cenário, a variável binária `flag_step_model` é igual a `True`, pois pretende-se realizar o avanço de um passo de integração. Para isso, executa-se a função do integrador denominada `step` (linha 21). Em seguida, os estados resultantes dessa função são atribuídos ao objeto `fmu_me[g]` (linha 23).

2. Tratamento de eventos (início na linha 25): no segundo cenário, a variável binária `flag_step_model` é igual a `False`, pois deseja-se atribuir ao `integrator` as derivadas acarretadas por um evento sem realizar um avanço de tempo. Utiliza-se o comando `get_derivatives` para extrair do modelo da FMU as derivadas que as variáveis de estado assumem após o evento (linha 27), ao mesmo tempo que se utiliza o comando `continuous_states` para manter a continuidade do estado `x` entre eventos (linha 29).

Depois da execução de um dos dois cenários descritos, as variáveis de estado do g-ésimo modelo de máquinas são extraídas do objeto `fmu_me[g]` através do comando `get` (linha 31). Depois, o termo `x[g]` serve como entrada para a função `Norton_model`, criada para este trabalho para extrair a corrente de Norton do g-ésimo gerador (`ig[g]`).

Portanto, ao final da varredura dos geradores, a saída da função `model_solution` é o vetor de correntes de Norton dos geradores (`ig`) acoplados à rede de transmissão.

3.2.3 Função da solução da rede

Conforme o programa principal da Subseção **3.2.1**, a solução da rede de transmissão é realizada através da função `net_solution`, mostrada no Algoritmo 8.

A primeira etapa da função `net_solution` é a inicialização do contador de iterações da rede como igual a `r = 0` (linha 10) e a inicialização da variável binária de convergência da rede como igual a `flag_net_converged = False` (linha 11). Com isso, inicia-se a execução das iterações da rede dentro de uma estrutura de `while` (linha 13). Sabendo disso, as etapas de uma iteração da rede são as seguintes:

1. Armazenamento do mais recente vetor de tensões nodais da rede (linha 15): essencial para o teste de convergência da rede.

2. Inicializações para a varredura das cargas ZIP: o vetor de correntes de carga (`il`) é inicializado como um vetor nulo, a matriz de admitâncias nodais (`Ymn`) é inicializada

Algoritmo 8 – Função `net_solution`.

```
def net_solution(Y, LU, piv, v, ig)

  # ----------- Entrada:
  # - Y: matriz de admitâncias nodais da rede.
  # - LU, piv, matrizes de fatoração LU da matriz Y.
  # - v: vetor de tensões nodais da rede.
  # - ig: vetor de correntes de Norton dos geradores.

  # ----------- Saída:
  # - v: vetor de tensões nodais da rede.

  # Inicializações: contador de iterações e variável binária de convergência.
  r = 0
  flag_net_converged = False

  # ================ Solução iterativa da rede:
  while flag_net_converged == False:

    # 1) Armazenamento do mais recente vetor de tensões nodais da rede:
    v_prev_net = v

    # 2) Inicializações para a varredura das cargas ZIP:
    il = np.zeros((num_bars, 1), dtype=complex)    # Vetor de correntes de carga.
    Ymn = Y                                        # Matriz de admitâncias nodais.
    flag_Y_alter = False                           # Variável binária de alteração de Ymn.

    # 3) Varredura das cargas ZIP:
    for l in range(num_loads):

      # 3.1) Índice da barra de carga:
      ind_bar = load_bars[l] - 1

      if np.abs( v[ind_bar] ) >= Vmn:
        # 3.2.1 - Modelagem de carga ZIP completa:
        Ic = Ic_aux[l] * np.exp( 1j * np.angle(v[ind_bar]) )
        Iadj = np.conjugate( S_y0[l]/v[ind_bar] ) * ( 1 - np.abs(v[ind_bar]/v0[ind_bar])**2 )
        il[ind_bar] += Iadj + Ic

      else:
        # 3.2.2 - Modelagem de carga do tipo impedância constante para baixas tensões:
        Ymn[ind_bar][ind_bar] += y_mn[l] - (yi[l] + y0[l])
        flag_Y_alter = True

    # 4) Cálculo do vetor de injeções de correntes nodais:
    i = ig - il

    # 5) Atualização do vetor de tensões nodais: dependente da variável binária flag_Y_alter.
    if flag_Y_alter == False:
      v = linalg.lu_solve((LU, piv), i)
    else:
      LU_mn, piv_mn = linalg.lu_factor(Ymn)
      v = linalg.lu_solve((LU_mn, piv_mn), i)

    # 6) Teste de convergência da rede:
    dv = np.abs(v - v_prev_net)
    flag_net_converged = (np.max(dv) <= e_abs and np.max(dv) <= np.max(v_prev_net)*e_rel)

    # 7) Incremento do contador de iterações:
    r += 1

  return v
```

Fonte: Elaborado pelo autor (2024)

como igual à matriz inicial `Y` e a variável binária de alteração de `Ymn` é inicializada como igual a `flag_Y_alter = False`.

3. Varredura das cargas ZIP (linha 21): realizada com base nas equações algébricas do Apêndice F. Nesta etapa, a tensão de cada barra de carga de modelo ZIP é comparada a um valor `Vmn`. Neste trabalho, conforme citado anteriormente, o valor adotado de `Vmn` é 70 %, que é o valor padrão adotado pelo programa ANATEM. Com isso, tem-se dois cenários possíveis de tensão de carga:

   - Barra de carga com tensão `v[ind_bar] >= Vmn` (início na linha 24): no primeiro cenário, adota-se a modelagem da carga ZIP completa mostrada na Figura 8. Essa modelagem visa representar a carga para uma faixa de tensões próximas à nominal, isto é, maiores ou iguais a um determinado valor `Vmn`.

     Com o objetivo de economizar esforço computacional, as admitâncias `yi` e `y0` são previamente adicionadas à matriz de admitâncias nodais (`Y`), isto é, logo depois do cálculo de regime permanente. Dessa forma, a matriz `Ymn` permanece inalterada quando as tensões nodais são próximas à nominal (isto é, acima de 0,7 [pu]) durante o regime transitório.

     Por outro lado, o vetor de correntes de carga (`il`) precisa ser atualizado ao longo da varredura das cargas. As correntes `Ic` e `Iadj` da Figura 8 são recalculadas a cada iteração da rede, porque dependem de valores que variam a cada iteração, isto é, as tensões nodais da rede (`v`). Realizados os cálculos, os valores de `Ic` e `Iadj` são adicionados ao vetor `il` (linha 28).

   - Barra de carga com tensão `v[ind_bar] < Vmn` (início na linha 29): no segundo cenário, despreza-se a modelagem completa da Figura 8 e considera-se apenas uma impedância constante `y_mn`, conforme mostrado na Figura 9.

     A admitância `y_mn` é previamente calculada com base em valores constantes de regime permanente e, em caso de baixas tensões, precisa ser adicionada à matriz de admitâncias nodais da rede de transmissão (`Ymn`). Além disso, deve-se também remover as admitâncias `yi` e `y0`, pois ambas correspondem à modelagem completa. Com isso, o termo matricial `Ymn[ind_bar][ind_bar]` é incrementado (linha 31) e define-se a variável binária `flag_Y_alter` como igual a `True` (linha 32).

4. Cálculo do vetor de injeções de correntes nodais das barras da rede de transmissão (linha 34): uma das entradas da função `net_solution` é o vetor de correntes de Norton dos geradores (`ig`), que se mantém fixo durante as iterações da rede, enquanto o vetor `il` é variável a cada iteração. Com esses dois vetores, calcula-se um novo vetor de injeções nodais `i = ig - il`.

5. Atualização do vetor de tensões nodais (início da linha 35): realizada através da função `linalg.lu_solve` (pacote `scipy`), cujas entradas são as matrizes de fatoração LU e o mais recente vetor de injeções (`i`).

   Caso a matriz `Ymn` não tenha sido alterada durante a varredura das cargas ZIP, tem-se `flag_Y_alter = False` (linha 36) e, portanto, o programa pode reutilizar as matrizes `LU` e `piv` (correspondentes à matriz inicial `Y`) para atualizar o vetor `v`. No entanto, no caso de `flag_Y_alter = True` (isto é, baixas tensões), deve-se calcular novas matrizes `LU_mn` e `piv_mn` e utilizá-las para atualizar o vetor de tensões (linha 40).

6. Teste de convergência da rede (linha 43): realizado a partir da diferença entre o vetor da iteração atual (`v`) e o vetor anterior das tensões nodais (`v_prev_net`). O resultado do teste é armazenado dentro da variável binária da convergência denominada `flag_net_converged`.

7. Incremento do contador de iterações (linha 45): incrementa-se o valor de `r` e verifica-se se foi atingido o limite de `10` iterações. Em caso afirmativo, o programa é interrompido e imprime uma mensagem de erro.

Após a convergência da solução iterativa da rede de transmissão, a saída da função `net_solution` é o vetor de tensões nodais da rede de transmissão (`v`).

3.2.4 Função do tratamento de eventos

Conforme o programa principal da Subseção **3.2.1**, o tratamento de eventos da rede de transmissão é realizado através da função `event_treatment`, mostrada no Algoritmo 9.

A primeira etapa da função `event_treatment` é a aplicação do evento através da função `apply_event` (linha 12), criada para este trabalho. Por exemplo, se ocorrer um curto-circuito na barra k da rede, realiza-se a alteração correspondente no termo matricial $\mathbf{Y}(k,k)$.

Com a modificação da matriz `Y` e o vetor de injeções de correntes nodais da mais recente solução convergente do método alternado (`i`), utiliza-se o método nodal para o cálculo da estimativa inicial das tensões nodais através da função `linalg.lu_solve` (pacote `scipy`). Com esse vetor inicial de tensões, acelera-se a convergência do tratamento de eventos.

Em seguida, na linha 16, utiliza-se a função `net_solution` (Subseção **3.2.3**) para calcular a solução iterativa da rede no instante t_e^+, isto é, levando em conta o evento.

Após a convergência da rede, a variável binária `flag_step_model` é definida como `False` (linha 18), para que a função `model_solution` (Subseção **3.2.2**) obtenha as

Algoritmo 9 – Função `event_treatment`.

```
1  def event_treatment(Y, v, i, ig)

2    # ----------- Entrada:
3    # - Y: matriz de admitâncias nodais da rede.
4    # - v: vetor de tensões nodais da rede.
5    # - i: vetor de injeções de correntes nodais da rede.
6    # - ig: vetor de correntes de geração.

7    # ----------- Saída:
8    # - Y: matriz de admitâncias nodais da rede.
9    # - LU, piv: matrizes da fatoração LU da matriz Y.
10   # - v: vetor de tensões nodais da rede.

11   # Aplicação do evento e estimativa inicial das tensões:
12   Y = apply_event(Y)
13   LU, piv = linalg.lu_factor(Y)
14   v = linalg.lu_solve((LU, piv), i)

15   # Solução da rede levando em conta o evento:
16   v = net_solution(Y, LU, piv, v, ig)

17   # Atualização das derivadas do modelo de máquinas:
18   flag_step_model = False
19   model_solution(fmu_me, integrator, v, flag_step_model)
20   flag_step_model = True

21   # Atualização dos termos históricos:
22   for g in range(num_gen):
23     integrator[g].update_history()

24   return Y, LU, piv, v
```

Fonte: Elaborado pelo autor (2024)

derivadas que as variáveis de estado do modelo de máquinas assumem após o evento sem realizar um avanço de tempo.

Com a obtenção das derivadas, define-se o termo `flag_step_model` como `True` (linha 20) e atualizam-se os termos históricos do `integrator` através da função do integrador denominada `update_history` (linha 23). Por último, a saída da função `event_treatment` é a matriz de admitâncias nodais (`Y`), as matrizes de fatoração LU correspondentes (`LU` e `piv`) e o vetor de tensões nodais da rede (`v`).

3.3 CONCLUSÕES PARCIAIS

Neste capítulo, foi apresentada a forma simplificada do sistema de equações algébrico-diferenciais que caracterizam um sistema elétrico de potência. Também foi apresentada a forma discretizada das equações diferenciais correspondentes aos geradores e equipamentos dinâmicos conectados à rede elétrica. Essa discretização, por sua vez, foi realizada pelo método de integração numérica trapezoidal, devido à sua consolidação na área de engenharia.

Em seguida, para a análise de regime transitório, foi adotado o método alternado

para a solução das equações da rede de transmissão (incluindo cargas lineares e não lineares) e das equações do modelo de máquinas compilado em uma FMU. Com a alternância das soluções de cada subsistema, é possível tratá-los de forma apropriada.

Além de descrever as etapas necessárias para a convergência da interface a cada instante de tempo de simulação, este capítulo também descreveu as etapas do tratamento de eventos, onde é necessário calcular duas soluções de rede: uma na iminência do evento e outra no instante infinitesimalmente posterior ao evento. Dessa forma, torna-se possível observar adequadamente a descontinuidade das derivadas dos estados dos modelos dinâmicos diante de perturbações.

Por fim, este capítulo utilizou o método iterativo alternado para a criação de um protótipo de um programa de estabilidade transitória em ambiente *Python*, onde foram criadas funções específicas para a solução do modelo, solução da rede e tratamento de eventos. Do ponto de vista da solução iterativa da rede de transmissão, cada gerador foi representado através de um equivalente de Norton, que por sua vez consiste em uma admitância constante e uma fonte de corrente que varia em função das tensões terminais que a rede entrega para os modelos das FMUs a cada iteração do método alternado. A representação de Norton foi adotada devido à possibilidade de aceleração da convergência e à ausência da necessidade de aumento das dimensões da matriz de admitâncias nodais da rede de transmissão.

4 INCLUSÃO DO SISTEMA DE DISTRIBUIÇÃO

Conforme visto no capítulo anterior, foi descrito o denominado método alternado do conjunto modelo-rede, que é utilizado como base para a simulação no tempo do regime transitório desta dissertação. Nesse método, o sistema elétrico é dividido em dois subsistemas menores, que são a rede de transmissão e as unidades geradoras, para que cada um seja resolvido separadamente e, consequentemente, a solução estimada de um subsistema seja utilizada para estimar a solução do outro subsistema. Sabendo disso, este capítulo adiciona à simulação um terceiro subsistema, correspondente ao sistema elétrico de distribuição.

Enquanto as redes de transmissão trabalham com tensões elevadas (maiores ou iguais a 230 kV, conforme a ANEEL) para conduzir a energia elétrica sem perdas acentuadas, as redes de distribuição trabalham com tensões reduzidas (menores do que 230 kV, conforme a ANEEL), com o objetivo de entregar a energia elétrica aos consumidores finais, sejam residenciais ou industriais, por exemplo. Portanto, a representação adequada do sistema de distribuição possui grande importância para simulações de sistemas elétricos de potência.

Para a representação desse novo subsistema, utiliza-se um módulo baseado no programa OpenDSS (*Open Distribution System Simulator*^{TM}). O OpenDSS consiste em um simulador de sistemas elétricos de distribuição e Recursos Energéticos Distribuídos (RED). Foi desenvolvido em 1997, na *Electrotek Concepts*, Inc. por Roger Dugan e Thomas McDemontt. Em 2004, foi adquirido pela EPRI Solutions (*Electric Power Research Institute*), que por sua vez liberou o programa para o público em 2008. Até hoje ganha novas atualizações de estudos de redes de distribuição e *Smart Grids* (DUGAN; MONTENEGRO, 2021).

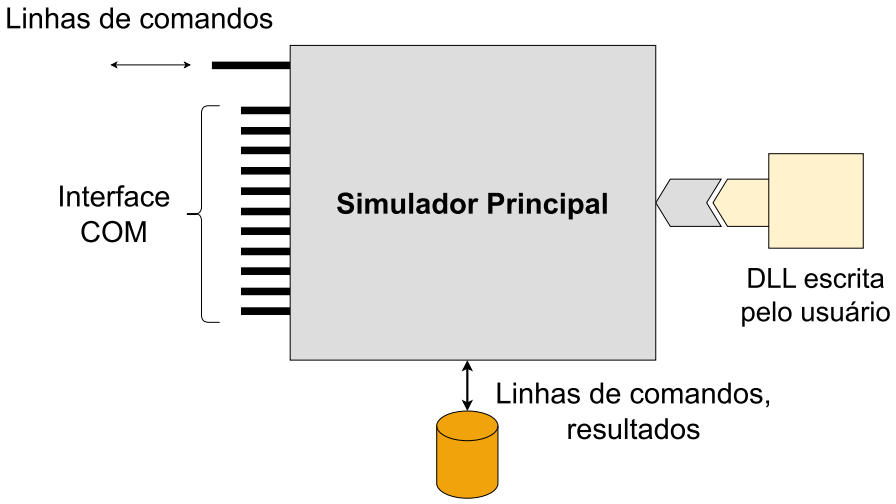
Uma das propriedades que incentiva a adoção do programa do EPRI é, por exemplo, o regime permanente de natureza senoidal, o que corresponde a simulações realizadas no domínio da frequência, através da solução de equações algébricas baseadas na matriz de admitâncias nodais da rede. Isso permite que as tensões nodais e correntes elétricas injetadas sejam representadas de forma fasorial (FREITAS, s.d.).

Outra propriedade importante é a linguagem de código aberto. Como o OpenDSS é a implementação *Open Source* do DSS, o programa é gratuito e sem custo de licença, o que incentiva sua utilização por parte de empresas, sendo inclusive homologado pela ANEEL (Agência Nacional de Energia Elétrica).

4.1 ESTRUTURA BÁSICA

A Figura 10 exibe um diagrama que representa a estrutura básica do OpenDSS, onde é mostrado que o programa pode ser implementado de três formas diferentes. A primeira forma é através de um programa executável autônomo, onde há uma interface de

Figura 10 – Estrutura do OpenDSS.



Fonte: (DUGAN; MONTENEGRO, 2021) (Adaptada)

usuário que permite a descrição de circuitos, a solução e a visualização de resultados. A segunda forma é através de um servidor COM (*Component Object Model*) implementado a partir de uma DLL (*Dynamic-link library*, ou Biblioteca de Vínculo Dinâmico), onde o OpenDSS pode ser controlado por uma variedade de plataformas de programas existentes (SEXAUER, 2016). Por último, a terceira forma é através de uma DLL padrão que fornece todas as funções do servidor COM, porém pode ser utilizada a partir de linguagens que não suportem o COM (DUGAN; MONTENEGRO, 2021).

Sabendo disso, a estrutura do OpenDSS é composta pelos seguintes elementos:

- Simulador principal (do inglês, *Simulation Engine*): recebe as linhas de comando, e também realiza a leitura da interface COM e da DLL escrita pelo usuário. Com essas entradas, realiza a simulação necessária e entrega os resultados correspondentes.
- Linhas de comando: servem de base para definir os circuitos do OpenDSS. Podem ser definidas diretamente pelo usuário, por um arquivo de texto fixo ou por programas externos (SEXAUER, 2016).
- Interface COM: permite o desenvolvimento de simulações, bem como o gerenciamento e exportação dos resultados da simulação (GRANADOS, 2018). Além disso, a interface COM também possibilita que o OpenDSS possa ser inicializado e controlado por programas externos e diferentes linguagens de programação, como por exemplo MATLAB, *Python*, C# e ferramentas do *Microsoft Office* (em destaque o *Visual Basic for Applications*, ou VBA). Dessa forma, o programa pode ser acionado de forma independente de qualquer banco de dados ou arquivo de texto fixo que defina um circuito (DUGAN; MONTENEGRO, 2021; SEXAUER, 2016).

- DLL: ou Biblioteca de Vínculo Dinâmico. Criada e customizada pelo usuário através de diversas linguagens de programação. Contém a implementação a ser acessada pelo simulador principal.

  Pode-se citar três vantagens da DLL. A primeira é a sua fácil substituição sem necessidade de alteração do restante do programa. A segunda vantagem é a separação da execução com a implementação, porque ao contrário das bibliotecas estáticas, as bibliotecas dinâmicas não precisam ser compiladas junto com o simulador principal. Vale ressaltar que a compilação do modelo na forma de DLL não requer o código-fonte do OpenDSS em si, o que torna o processo de inclusão de modelos mais robusto, encapsulando possíveis erros nas DLLs desenvolvidas. Por último, tem-se vantagem do foco na modelagem externa dos elementos de interesse, pois dessa forma o usuário pode se concentrar no aperfeiçoamento da DLL enquanto o simulador principal cuida de outros aspectos do modelo do sistema de distribuição.

4.2 FLUXO DE POTÊNCIA

Anteriormente nesta dissertação, citou-se o método iterativo de *Newton-Raphson* para o cálculo do fluxo de potência do sistema elétrico em regime permanente. Esse é um método comumente aplicado para o cálculo da rede de transmissão trifásica cuja representação é na forma de um monofásico equivalente. Com isso, adota-se apenas a componente de sequência positiva da transmissão, pois assume-se que a rede é satisfatoriamente equilibrada (FREITAS, s.d.).

No entanto, para o cálculo de sistemas elétricos de distribuição, deve-se adotar outros métodos iterativos que sejam mais adequados, porque a distribuição possui propriedades que a transmissão não possui, como por exemplo (SCHINCARIOL; BELIN, 2019):

- Escalabilidade: o número de barras das redes de distribuição de energia pode ser bem maior do que o número de barras dos sistemas de transmissão.

- Elevada relação R/X: na transmissão, há métodos iterativos que desprezam as resistências de linha, pois elas geralmente são muito menores do que as reatâncias ($R/X \approx 0$). No entanto, esses métodos não podem ser diretamente aplicados na distribuição, porque agora a grande maioria das resistências não são desprezíveis.

- Baixa capacitância das linhas: essa propriedade acarreta maiores quedas de tensão ao longo do alimentador. Portanto, se os métodos iterativos da transmissão forem aplicados a redes de distribuição, pode ocorrer uma dificuldade de convergência, ou até mesmo divergência.

- Geração Distribuída (GD): uma vez que a GD se dá na forma de fontes de energia renováveis (eólica e fotovoltaica, por exemplo), esse tipo de geração possui natureza

intermitente. Essa propriedade pode causar comportamentos indesejados para a rede de distribuição, como por exemplo uma inversão do fluxo de potência e o prejuízo do perfil de tensão da rede.

Dentre os métodos de solução encontrados na literatura, destacam-se os que levam em conta a característica trifásica da distribuição. Por exemplo, o *Forward-Backward Sweep*, que consiste em um método de varredura bastante utilizado para a solução de sistemas de distribuição radiais de pequeno e médio porte, devido à sua eficiência e simplicidade de implementação. No entanto, em sistemas de distribuição de grande porte (por exemplo, redes de 15000 barras trifásicas), métodos de solução que envolvem matrizes esparsas podem apresentar maior velocidade de convergência em relação aos métodos de varredura (KERSTING; DUGAN, 2006; FREITAS, 2015).

Dentre os métodos de solução de fluxo de potência adotados pelo OpenDSS, destaca-se a iteração de ponto fixo baseada na matriz de admitâncias nodais da rede de distribuição ($\mathbf{Y}_{dss}$). Esse método consiste em cálculos sucessivos do vetor de tensões nodais da rede através da equação matricial mostrada em (4.1), onde é conhecido o vetor de injeções de correntes nodais em função do vetor de tensões nodais da iteração anterior $(d-1)$ do OpenDSS (DUGAN; MONTENEGRO, 2021).

$$\mathbf{Y}_{dss} \cdot \mathbf{v}_{dss}^{(d)} = \mathbf{i}_{dss}\left(\mathbf{v}_{dss}^{(d-1)}\right) \tag{4.1}$$

Para calcular o vetor $\mathbf{v}_{dss}^{(d)}$, cada iteração do OpenDSS decompõe a matriz $\mathbf{Y}_{dss}$ através de uma função baseada na decomposição LU, denominada *KLUsolve*. Os elementos que constituem a matriz $\mathbf{Y}_{dss}$ (e também o vetor $\mathbf{i}_{dss}$) são apresentados na Subseção **4.2.1**, enquanto a função *KLUsolve* (em conjunto com a solução iterativa de ponto fixo) é explicada em maiores detalhes na Subseção **4.2.2**.

4.2.1 Modelos e elementos

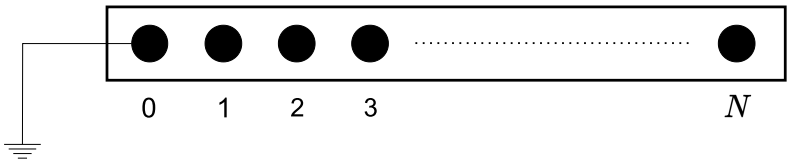
Conforme dito anteriormente, o cálculo do fluxo de potência do sistema de distribuição no OpenDSS é realizado através do método de iteração de ponto fixo baseado na matriz de admitâncias nodais. Para se obter essa matriz, o OpenDSS realiza a construção (apresentada de forma detalhada em (FREITAS, 2015)) das matrizes individuais de cada elemento presente na rede de distribuição. Esses elementos podem ser trifásicos, ao contrário dos da transmissão.

No OpenDSS, os elementos se dividem em basicamente dois tipos: elementos de transporte de energia e elementos de conversão de energia.

4.2.1.1 Modelo de barra

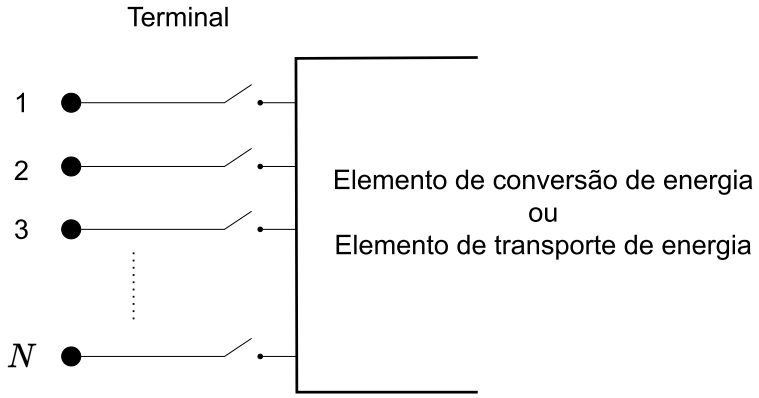
Em muitas análises de sistemas de potência, os conceitos de "barra" e "nó" são praticamente sinônimos, mas no circuito do OpenDSS, são conceitos distintos. Agora,

Figura 11 – Modelo de barra.



Fonte: (DUGAN; MONTENEGRO, 2021) (Adaptada)

Figura 12 – Modelo de terminal de um elemento.



Fonte: (DUGAN; MONTENEGRO, 2021) (Adaptada)

considera-se que uma barra consiste num elemento de N nós, que servem como pontos de conexão dos terminais de todos os outros elementos do circuito, conforme a Figura 11.

Ao contrário de alguns programas de fluxo de potência, não há tipos especiais de barras (como barra PQ, PV ou *slack*) no OpenDSS. Em vez disso, sua função depende dos elementos conectados a seus nós. Portanto, uma barra é definida somente depois da definição dos elementos conectados à mesma.

4.2.1.2 Modelo de terminal

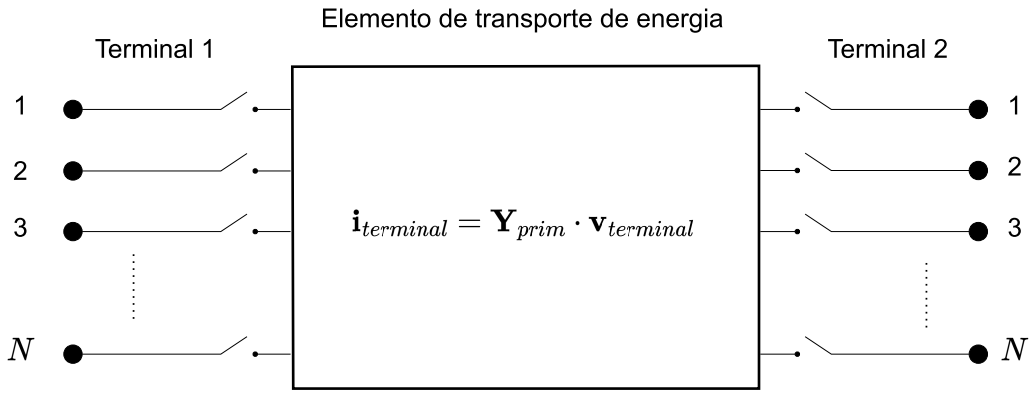
Cada elemento elétrico do sistema de potência possui um ou mais terminais. Cada terminal possui um ou N nós, conforme a Figura 12. Além disso, cada terminal deve ser conectado a apenas uma única barra.

4.2.1.3 Elementos de transporte de energia

No OpenDSS, a função básica desses elementos é transportar energia de um ponto para outro. Para isso, geralmente possuem dois ou mais terminais multifásicos, conforme a Figura 13. Além disso, no regime permanente, podem ser inteiramente representados por sua matriz de admitâncias nodais primitiva $\left(\overline{Y}_{prim}\right)$.

Alguns exemplos de elementos de transporte de energia são transformadores (configuração trifásica, bifásica ou monofásica; estrela-estrela ou delta-estrela; aterramento

Figura 13 – Elemento de transporte de energia.



Fonte: (DUGAN; MONTENEGRO, 2021) (Adaptada)

sólido ou por impedância) e linhas com presença de impedâncias mútuas entre as fases.

Vale ressaltar que capacitores e reatores também são elementos de transporte de energia de 2 terminais cada. Porém, também podem ser conectados em paralelo (ou *shunt*), ou seja, conectados à rede através de apenas 1 terminal, conforme citado na explicação a respeito de elementos de conversão de energia.

4.2.1.4 Elementos de conversão de energia

Elementos de conversão de energia possuem a função de converter a energia da forma elétrica para alguma outra forma, ou vice-versa. Alguns podem temporariamente armazenar a energia e depois liberá-la para a rede. A maioria possui apenas uma conexão ao sistema de potência e, por isso, apenas um terminal de uma ou mais fases.

A descrição dos elementos de conversão de energia pode ser desde uma simples impedância até um conjunto de equações diferenciais que resultam numa função **F** de corrente de injeção (no caso de geradores) ou de compensação (no caso de cargas). A representação genérica de **F** está mostrada na Figura 14.

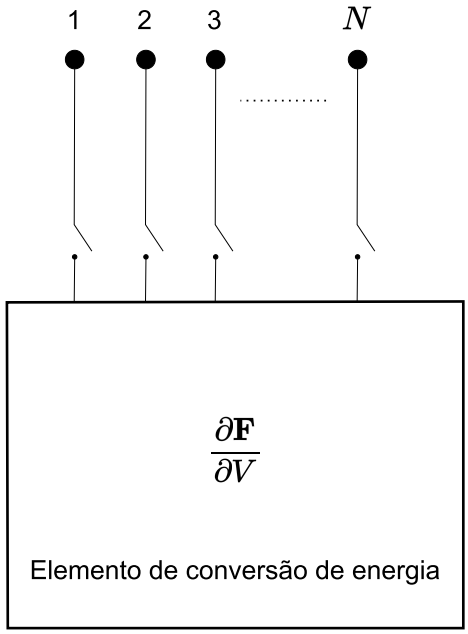
A função **F** de um elemento varia de acordo com o tipo de simulação a ser realizada. Em casos simples, essa variação corresponde apenas a uma matriz $\overline{Y}_{prim}$.

Exemplos de elementos de conversão de energia: geradores trifásicos com presença de impedâncias mútuas entre as fases; cargas com configuração monofásica, bifásica, trifásica estrela ou trifásica delta; elementos reativos (capacitores e reatores, conectados à rede através de apenas 1 terminal).

Considerando que podem ser elementos não lineares, cargas e geradores são tratados como equivalentes de Norton, conforme a Figura 15. Essa representação consiste em uma admitância primitiva constante $\overline{Y}_{prim}$ (adicionada à matriz de admitância da rede de distribuição $\mathbf{Y}_{dss}$) em paralelo com uma fonte de corrente de "compensação". A corrente

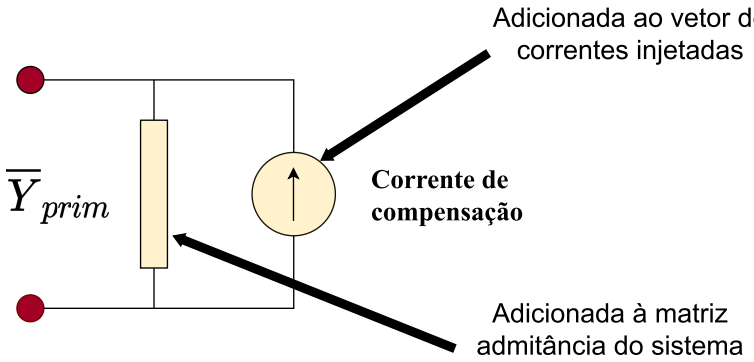
Figura 14 – Elemento de conversão de energia.

$$I_{terminal}(t) = \mathbf{F}\left(V_{terminal}, [\text{Estado}], t\right)$$



Fonte: (DUGAN; MONTENEGRO, 2021) (Adaptada)

Figura 15 – Equivalente de Norton do elemento de conversão de energia.



Fonte: (DUGAN; MONTENEGRO, 2021) (Adaptada)

recebe esse nome porque compensa a parcela não linear do elemento correspondente ao ser adicionada ao vetor de injeções de correntes nodais ($\mathbf{i}_{dss}$), conforme mostrado na Subseção **4.2.2**.

Algumas propriedades da representação de Norton são:

- Possui bom desempenho e permite uma variedade de modelagens de carga em função da tensão na distribuição.
- Atinge a convergência para a grande maioria das condições do sistema de distribuição, devido à admitância $\overline{Y}_{prim}$. Se os elementos de conversão de energia fossem tratados

apenas como uma corrente equivalente, não haveria a adição de $\overline{Y}_{prim}$ em $\mathbf{Y}_{dss}$, fazendo com que a convergência fosse mais custosa, ou até mesmo improvável.

- Apesar de não ser obrigatório, a matriz $\overline{Y}_{prim}$ geralmente é mantida constante, o que reduz a quantidade de vezes que a matriz $\mathbf{Y}_{dss}$ da distribuição é reconstruída. Com isso, tem-se um melhor desempenho computacional.

Independentemente da escolha do usuário, todos os modelos de carga presentes no OpenDSS são convertidos para o modelo impedância constante quando se encontram fora da faixa de tensão normal (isto é, fora da faixa entre $V_{min,pu}$ e $V_{max,pu}$). Essa conversão ocorre na tentativa de garantir a convergência mesmo diante de significativas quedas (ou significativos aumentos) de tensão.

4.2.2 Cálculo do fluxo de potência do sistema de distribuição

Conforme visto na subseção anterior, as matrizes de admitância primitiva $\left(\overline{Y}_{prim}\right)$ são criadas individualmente para cada elemento do circuito e são utilizadas para a construção da matriz de admitâncias nodais ($\mathbf{Y}_{dss}$) que representa toda a rede de distribuição. Com a matriz $\mathbf{Y}_{dss}$ montada pelo OpenDSS, utiliza-se o solucionador de equações de matrizes esparsas denominado *KLUsolve*, que consiste em uma biblioteca de vínculo dinâmico baseada na decomposição LU (DAVIS; PALAMADAI NATARAJAN, 2010).

As entradas da função *KLUsolve* (entregues pelo OpenDSS) são: matriz de admitâncias nodais do circuito; mapeamento das barras às quais cada elemento está conectado; correntes de injeção (no caso dos geradores) e de compensação (no caso das cargas). A construção das matrizes dos elementos, bem como o mapeamento dos mesmos, é realizada apenas uma vez (salvo alguma mudança na topologia da rede de distribuição), enquanto as correntes precisam ser atualizadas a cada iteração.

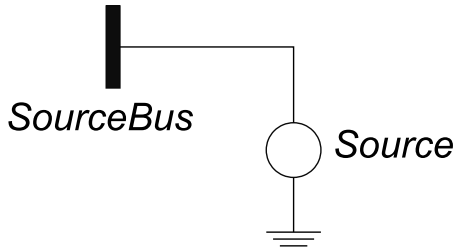
Depois de receber as entradas necessárias, o *KLUsolve* encontra as tensões nodais correspondentes, através da solução da equação $\mathbf{Y}_{dss} \cdot \mathbf{v}_{dss} = \mathbf{i}_{dss}$, e as entrega de volta para o OpenDSS. Essas tensões são utilizadas pelo OpenDSS para calcular novas correntes de injeção e de compensação, que por sua vez servem como entrada para o *KLUsolve* e iniciam uma nova iteração.

As etapas do cálculo do fluxo de potência do sistema de distribuição são os seguintes:

(i) Cálculo das estimativas iniciais de tensão $\left(\mathbf{v}_{dss}^{(0)}\right)$ pelo *KLUsolve*:

No OpenDSS, uma rede de distribuição é inicializada por um elemento denominado *Circuit*, que consiste em um equivalente trifásico de Thévenin cuja fonte de tensão é denominada *Source*. Esse elemento é conectado a uma barra denominada *SourceBus*, ou seja, a barra *swing* (referência) do cálculo de fluxo de potência, conforme a Figura 16.

Figura 16 – Equivalente de Thévenin do elemento *Circuit* do OpenDSS.



Fonte: (DUGAN; MONTENEGRO, 2021) (Adaptada)

Para acelerar a convergência do fluxo de potência, primeiro considera-se a rede em vazio, ou seja, modelam-se as cargas e geradores por seus equivalentes lineares sem correntes de injeção nem de compensação, restando apenas o elemento *Circuit*. Com isso, o equivalente de Thévenin correspondente é utilizado para calcular a estimativa inicial de corrente de injeção do elemento. Com isso, tem-se a corrente $\overline{I}_1^{(0)}$, calculada pela equação (4.2).

$$\overline{I}_1^{(0)} = \overline{Y}_{Circuit} \cdot \overline{V}_{nominal} \tag{4.2}$$

Com essa única corrente injetada à rede a vazio, o *KLUsolve* resolve a equação matricial de (4.3) e retorna para o OpenDSS as tensões nodais (vetor $\mathbf{v}_{dss}^{(0)}$) que dão início ao método iterativo.

$$\mathbf{Y}_{dss} \cdot \mathbf{v}_{dss}^{(0)} = \mathbf{i}_{dss}^{(0)}$$

$$\begin{bmatrix} \overline{Y}_{11} & \overline{Y}_{12} & \cdots & \overline{Y}_{1k} & \cdots & \overline{Y}_{1n} \\ \overline{Y}_{21} & \overline{Y}_{22} & \cdots & \overline{Y}_{2k} & \cdots & \overline{Y}_{2n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \overline{Y}_{k1} & \overline{Y}_{k2} & \cdots & \overline{Y}_{kk} & \cdots & \overline{Y}_{kn} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \overline{Y}_{n1} & \overline{Y}_{n2} & \cdots & \overline{Y}_{nk} & \cdots & \overline{Y}_{nn} \end{bmatrix} \cdot \begin{bmatrix} \overline{V}_1^{(0)} \\ \overline{V}_2^{(0)} \\ \vdots \\ \overline{V}_k^{(0)} \\ \vdots \\ \overline{V}_n^{(0)} \end{bmatrix} = \begin{bmatrix} \overline{I}_1^{(0)} \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.3}$$

Vale ressaltar que o vetor inicial $\mathbf{v}_{dss}^{(0)}$ costuma ser próximo à solução de convergência incluindo os elementos não lineares. Essa proximidade inicial é importante, porque o OpenDSS é projetado para resolver redes multifásicas arbitrárias que podem conter vários tipos de transformadores e conexões.

(ii) Início da d-ésima iteração e cálculo do vetor de correntes pelo OpenDSS $\left(\mathbf{i}_{dss}^{(d)}\right)$:

Através das tensões nodais, o método iterativo começa com a obtenção das correntes de todos os elementos de conversão de energia. Com isso, tem-se o seguinte:

- Quando se tem $d = 1$, $\mathbf{v}_{dss}^{(d-1)}$ diz respeito às tensões iniciais do passo (i).
- O vetor de tensões $\mathbf{v}_{dss}^{(d-1)}$ serve como entrada para o OpenDSS calcular as correntes de injeção (geradores) e de compensação (cargas). Essas correntes são armazenadas dentro do vetor $\mathbf{i}_{dss}^{(d)}$, conforme apresentado em (4.4).

$$\mathbf{v}_{dss}^{(d-1)} = \begin{bmatrix} \overline{V}_1^{(d-1)} \\ \overline{V}_2^{(d-1)} \\ \vdots \\ \overline{V}_k^{(d-1)} \\ \vdots \\ \overline{V}_n^{(d-1)} \end{bmatrix} \xrightarrow[\text{do OpenDSS}]{\text{Geradores e cargas}} \mathbf{i}_{dss}^{(d)} = \begin{bmatrix} \overline{I}_1^{(d)} \\ \overline{I}_2^{(d)} \\ \vdots \\ \overline{I}_k^{(d)} \\ \vdots \\ \overline{I}_n^{(d)} \end{bmatrix} \tag{4.4}$$

Esse processo possui a vantagem de permitir que o comportamento não linear dos elementos de conversão seja modelado de diversas formas.

(iii) Cálculo do vetor atualizado de tensões $\left(\mathbf{v}_{dss}^{(d)}\right)$ pelo *KLUsolve*:

O vetor de correntes $\mathbf{i}_{dss}^{(d)}$ serve como entrada para a função *KLUsolve* calcular as tensões das barras do sistema de distribuição. Essas tensões são armazenadas no vetor $\mathbf{v}_{dss}^{(d)}$, conforme apresentado em (4.5).

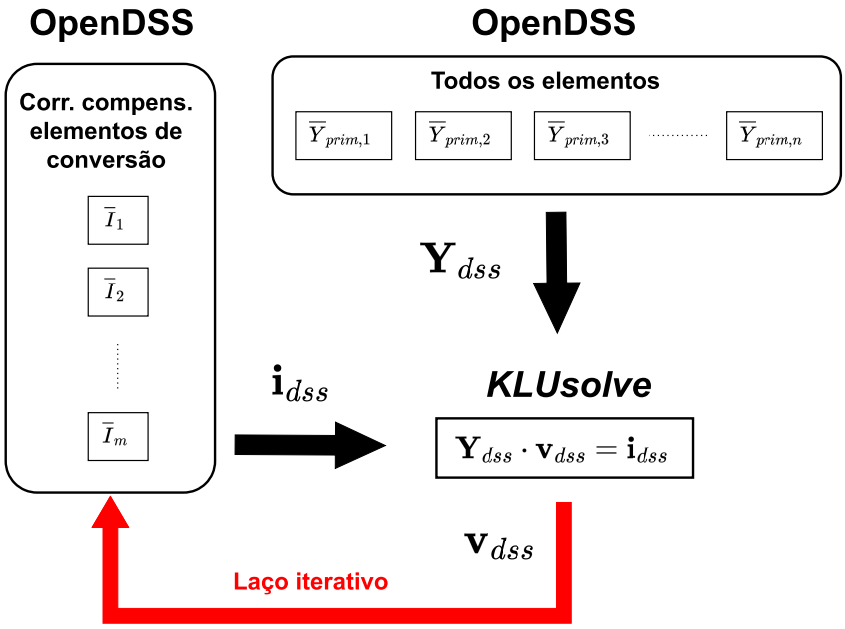
$$\mathbf{Y}_{dss} \cdot \mathbf{v}_{dss}^{(d)} = \mathbf{i}_{dss}^{(d)}$$

$$\begin{bmatrix} \overline{Y}_{11} & \overline{Y}_{12} & \cdots & \overline{Y}_{1k} & \cdots & \overline{Y}_{1n} \\ \overline{Y}_{21} & \overline{Y}_{22} & \cdots & \overline{Y}_{2k} & \cdots & \overline{Y}_{2n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \overline{Y}_{k1} & \overline{Y}_{k2} & \cdots & \overline{Y}_{kk} & \cdots & \overline{Y}_{kn} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \overline{Y}_{n1} & \overline{Y}_{n2} & \cdots & \overline{Y}_{nk} & \cdots & \overline{Y}_{nn} \end{bmatrix} \cdot \begin{bmatrix} \overline{V}_1^{(d)} \\ \overline{V}_2^{(d)} \\ \vdots \\ \overline{V}_k^{(d)} \\ \vdots \\ \overline{V}_n^{(d)} \end{bmatrix} = \begin{bmatrix} \overline{I}_1^{(d)} \\ \overline{I}_2^{(d)} \\ \vdots \\ \overline{I}_k^{(d)} \\ \vdots \\ \overline{I}_n^{(d)} \end{bmatrix} \tag{4.5}$$

(iv) Cálculo do erro absoluto de tensão $\left(\Delta\mathbf{v}_{dss}^{(d)}\right)$: equação (4.6).

$$\Delta\mathbf{v}_{dss}^{(d)} = \mathbf{v}_{dss}^{(d)} - \mathbf{v}_{dss}^{(d-1)} = \begin{bmatrix} \overline{V}_1^{(d)} \\ \overline{V}_2^{(d)} \\ \vdots \\ \overline{V}_k^{(d)} \\ \vdots \\ \overline{V}_n^{(d)} \end{bmatrix} - \begin{bmatrix} \overline{V}_1^{(d-1)} \\ \overline{V}_2^{(d-1)} \\ \vdots \\ \overline{V}_k^{(d-1)} \\ \vdots \\ \overline{V}_n^{(d-1)} \end{bmatrix} \tag{4.6}$$

Figura 17 – Resumo do método iterativo do fluxo de potência do sistema de distribuição.



Fonte: (DUGAN; MONTENEGRO, 2021) (Adaptada)

O método iterativo converge se for atendida a tolerância de convergência absoluta mostrada em (4.7). O valor padrão de ε_{dss} é 0,0001 [pu].

$$\max \left| \Delta \mathbf{v}_{dss}^{(d)} \right| \leq \varepsilon_{dss} \tag{4.7}$$

Caso a relação mostrada em (4.7) não seja atendida, realiza-se o incremento do contador de iterações ($d = d + 1$) e retorna-se ao passo (ii), onde o OpenDSS usa as tensões $\mathbf{v}_{dss}^{(d)}$ para calcular as correntes $\mathbf{i}_{dss}^{(d+1)}$, e retomam-se os cálculos até atingir a convergência.

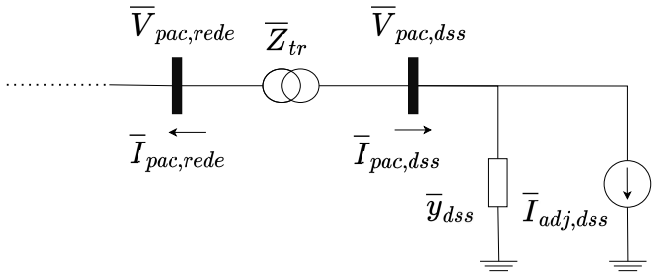
Esses passos do método iterativo do fluxo de potência do sistema de distribuição estão resumidos na Figura 17.

4.3 INCLUSÃO DO SISTEMA DE DISTRIBUIÇÃO AO MÉTODO ALTERNADO

No Capítulo 3, o protótipo do programa de estabilidade soluciona o regime permanente através de um método consolidado na literatura, que é o método iterativo de *Newton-Raphson* (explicado no Apêndice E), enquanto o regime transitório é simulado através da solução alternada entre a rede de transmissão e o modelo de máquinas elétricas.

No entanto, com a inclusão de um módulo baseado em OpenDSS à simulação computacional, o sistema de distribuição correspondente é conectado à rede de transmissão através de um transformador de conexão entre o PAC (Ponto de Acoplamento Comum) da transmissão e o PAC da distribuição. Portanto, para garantir o cálculo correto da tensão

Figura 18 – Conexão entre o PAC da rede de transmissão e o PAC da rede de distribuição.



Fonte: Elaborada pelo autor (2023)

nodal de ambos os PACs, não só a solução de regime transitório deve ser adaptada, como também o de regime permanente. Essas adaptações estão apresentadas nesta seção.

4.3.1 Considerações iniciais

Neste trabalho, o sistema de distribuição modelado é o de 38 barras, descrito no Apêndice D. Nesse sistema, a barra 1 é o PAC, isto é, a barra 1 é conectada ao PAC do sistema de transmissão de 11 barras através de um transformador de impedância $\overline{Z}_{tr}$.

Sabendo disso, a matriz de admitâncias nodais da rede de transmissão ($\mathbf{Y}$) é aumentada em uma linha e uma coluna. Portanto, no caso do sistema de 11 barras acoplado ao de 38 barras, a matriz $\mathbf{Y}$ correspondente passa a ter 12 linhas e 12 colunas. Dessa forma, o valor de $\overline{Z}_{tr}$ é adicionado à matriz conforme mostrado em (4.8).

$$\mathbf{Y} \leftarrow \overline{Z}_{tr} \tag{4.8}$$

Além disso, durante a solução da transmissão, a rede de distribuição é representada por um equivalente de Norton, que consiste em uma admitância $\overline{y}_{dss}$ em paralelo com uma fonte de corrente de ajuste $\overline{I}_{adj,dss}$, conforme a Figura 18. No regime permanente, ambos os parâmetros são calculados de forma iterativa. No entanto, no regime transitório, a admitância é mantida fixa (com base nos resultados de regime permanente), enquanto a fonte de corrente continua sendo calculada de forma iterativa.

Com o objetivo de facilitar a convergência do sistema de transmissão com a presença do módulo de OpenDSS, o valor de $\overline{y}_{dss}$ é calculado através do método de sensibilidade central, descrito em (CHAGAS, 2022). Para descrevê-lo, escreve-se a corrente $\overline{I}_{pac,dss}$ como uma função f_{dss} dependente da tensão do PAC da rede de distribuição, conforme mostrado na equação (4.9).

$$\overline{I}_{pac,dss} = f_{dss}\left(\overline{V}_{pac,dss}\right) \tag{4.9}$$

Aplicando a aproximação de primeira ordem da série de *Taylor* em torno de uma tensão nodal $\overline{V}_0$ qualquer, tem-se a equação mostrada em (4.10).

$$\overline{I}_{pac,dss} \approx \overline{I}_0 + \left.\frac{\Delta \overline{I}_{pac,dss}}{\Delta \overline{V}_{pac,dss}}\right|_{V_0} \cdot \left(\overline{V}_{pac,dss} - \overline{V}_0\right)$$

$$\overline{I}_{pac,dss} - \overline{I}_0 \approx \left.\frac{\Delta \overline{I}_{pac,dss}}{\Delta \overline{V}_{pac,dss}}\right|_{V_0} \cdot \left(\overline{V}_{pac,dss} - \overline{V}_0\right)$$

$$\Delta I_{pac,dss} \approx \left.\frac{\Delta \overline{I}_{pac,dss}}{\Delta \overline{V}_{pac,dss}}\right|_{V_0} \cdot \Delta V_{pac,dss} \tag{4.10}$$

De acordo com a equação (4.10), a admitância $\overline{y}_{dss}$ pode ser definida com base na variação da corrente $\overline{I}_{pac,dss}$ em função da variação da tensão $\overline{V}_{pac,dss}$ em torno de $\overline{V}_0$, conforme mostrado na equação (4.11).

$$\overline{y}_{dss} = \left.\frac{\Delta \overline{I}_{pac,dss}}{\Delta \overline{V}_{pac,dss}}\right|_{V_0} \tag{4.11}$$

Neste trabalho, a função f_{dss} da equação (4.9) é o fluxo de potência que o módulo `OpenDSSDirect` executa no ambiente em *Python*. Em conjunto com as execuções desse módulo, são realizadas variações algébricas de tensão em torno do ponto $\left(\overline{I}_0, \overline{V}_0\right)$. Com isso, calcula-se o valor de $\overline{y}_{dss}$ através da equação mostrada em (4.12).

$$\Delta I_{pac,dss} = \overline{y}_{dss}\, \Delta V_{pac,dss} \tag{4.12}$$

Com base na equação (4.12), a estimativa proposta para $\overline{y}_{dss}$ ocorre da seguinte forma:

- Em uma iteração qualquer de regime permanente, tem-se o PAC da distribuição com tensão igual a $\overline{V}_0 = V_0^{(Re)} + j\, V_0^{(Im)}$ e corrente igual a $\overline{I}_0$.
- Aplicando, por exemplo, um incremento positivo $\Delta V_{pac,dss}^{(Re)}$ à parte real de $\overline{V}_0$, tem-se uma nova tensão $\overline{V}_{0,pos} = \overline{V}_0 + \Delta V_{pac,dss}^{(Re)}$ e uma nova corrente correspondente $\overline{I}_{0,pos}$. Com isso, tem-se a equação mostrada em (4.13).

$$\Delta I_{pac,dss} = \overline{y}_{dss}\, \Delta V_{pac,dss}$$

$$\overline{I}_{0,pos} - \overline{I}_0 = \overline{y}_{dss}\, \Delta V_{pac,dss}^{(Re)} \tag{4.13}$$

- Analogamente, aplicando o mesmo incremento $\Delta V_{pac,dss}^{(Re)}$ de forma negativa à parte real de $\overline{V}_0$, tem-se uma nova tensão $\overline{V}_{0,neg} = \overline{V}_0 - \Delta V_{pac,dss}^{(Re)}$ e uma nova corrente correspondente $\overline{I}_{0,neg}$. Com isso, tem-se a equação mostrada em (4.14).

$$\Delta I_{pac,dss} = \overline{y}_{dss}\, \Delta V_{pac,dss}$$

$$\overline{I}_{0,neg} - \overline{I}_0 = \overline{y}_{dss}\, \left(-\Delta V_{pac,dss}^{(Re)}\right) \tag{4.14}$$

- Realizando uma subtração entre as equações (4.13) e (4.14), tem-se a equação de $\overline{y}_{dss}$ mostrada em (4.15).

$$\overline{I}_{0,pos} - \overline{I}_0 - \left(\overline{I}_{0,neg} - \overline{I}_0\right) = \overline{y}_{dss}\,\Delta V^{(Re)}_{pac,dss} - \left(\overline{y}_{dss}\,\left(-\Delta V^{(Re)}_{pac,dss}\right)\right)$$

$$\overline{I}_{0,pos} - \cancel{\overline{I}_0} - \overline{I}_{0,neg} + \cancel{\overline{I}_0} = \overline{y}_{dss}\,\Delta V^{(Re)}_{pac,dss} + y_{dss}\,\Delta V^{(Re)}_{pac,dss}$$

$$\overline{I}_{0,pos} - \overline{I}_{0,neg} = 2\;\overline{y}_{dss}\,\Delta V^{(Re)}_{pac,dss}$$

$$\overline{y}_{dss} = \frac{\overline{I}_{0,pos} - \overline{I}_{0,neg}}{2\,\Delta V^{(Re)}_{pac,dss}} \tag{4.15}$$

Considerações sobre a estimativa de $\overline{y}_{dss}$ pelo método de sensibilidade:

- Deve ser realizada de forma iterativa, porque cada iteração possui um valor diferente de $\overline{V}_0$ e, consequentemente, possui uma relação diferente entre a corrente e a tensão do PAC. Essa característica está associada à natureza não linear do sistema de distribuição, que deve ser levada em conta durante a solução da rede de transmissão.
- O valor de $\overline{y}_{dss}$ poderia ser calculado diretamente pela equação (4.13) (sensibilidade positiva da tensão do PAC) ou pela equação (4.14) (sensibilidade negativa). No entanto, aplicar uma sensibilidade central à tensão do PAC (equação (4.15)) pode contribuir para o cálculo de valores mais precisos de $\overline{y}_{dss}$, principalmente quando o sistema de distribuição possui equipamentos não lineares que operam sob um valor qualquer de tensão $\overline{V}_0$ que possa acarretar um acentuado comportamento não linear.
- A explicação apresentada envolve apenas o incremento da parte real da tensão, mas também é possível aplicar o mesmo raciocínio ao incremento da parte imaginária.

4.3.2 Regime permanente com OpenDSS

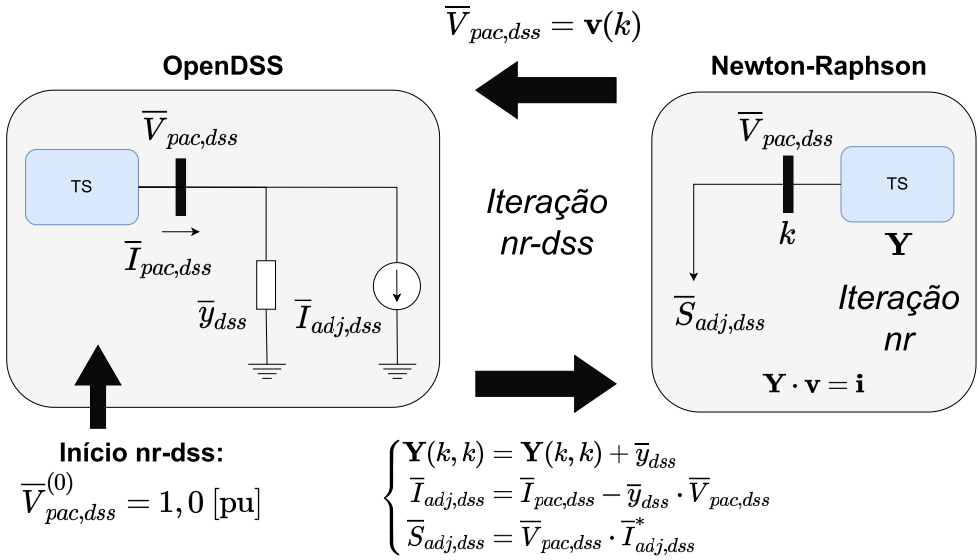
Neste trabalho, o cálculo do regime permanente é iniciado por um valor de tensão unitário do elemento *Circuit*, ou seja, $\overline{V}^{(0)}_{pac,dss} = 1{,}0$ [pu], que serve como entrada para o bloco **OpenDSS** da Figura 19 e inicia a primeira iteração "nr-dss", ou iteração do conjunto *Newton-Raphson* mais OpenDSS.

Descrição dos blocos da Figura 19:

(a) Inicialização do contador de iterações do conjunto nr-dss: $i = 1$.

As etapas de (b) a (e) caracterizam a solução iterativa do conjunto nr-dss.

Figura 19 – Regime permanente com OpenDSS.



Fonte: Elaborada pelo autor (2023)

(b) Armazenamento da mais recente tensão do PAC da distribuição $\left(\overline{V}^{(i-1)}_{pac,dss}\right)$: quando se tem $i = 1$, essa tensão corresponde à estimativa inicial igual a 1,0 [pu].

(c) Bloco **OpenDSS**: é o primeiro bloco do regime permanente.

A mais recente tensão do PAC da distribuição $\left(\overline{V}^{(i-1)}_{pac,dss}\right)$ serve como entrada para o módulo do OpenDSS e é utilizada para calcular a admitância $\overline{y}^{(i)}_{dss}$ e a corrente $\overline{I}^{(i)}_{pac,dss}$.

Primeiro, o valor de $\overline{I}^{(i)}_{pac,dss}$ é obtido através de dois passos: fornecer a tensão de entrada $\overline{V}^{(i-1)}_{pac,dss}$ ao módulo do OpenDSS e habilitar a execução do fluxo de potência da distribuição. Com isso, a corrente pode ser descrita como uma função f_{dss} dependente da tensão do PAC, conforme mostrado na equação (4.16a). Em seguida, o valor de $\overline{y}^{(i)}_{dss}$ é calculado com base no método de sensibilidade central, detalhado na Subseção **4.3.1** e mostrado na equação (4.16b).

$$\overline{I}^{(i)}_{pac,dss} = f_{dss}\left(\overline{V}^{(i-1)}_{pac,dss}\right) \tag{4.16a}$$

$$\overline{y}^{(i)}_{dss} = \frac{\overline{I}^{(i)}_{0,pos} - \overline{I}^{(i)}_{0,neg}}{2\,\Delta V^{(Re)}_{pac,dss}} \tag{4.16b}$$

Com os valores estimados de $\overline{y}_{dss}^{(i)}$ e $\overline{I}_{pac,dss}^{(i)}$, calcula-se a corrente de ajuste de Norton $\left(\overline{I}_{adj,dss}^{(i)}\right)$ através da equação (4.17).

$$\overline{I}_{adj,dss}^{(i)} = \overline{I}_{pac,dss}^{(i)} - \overline{y}_{dss}^{(i)} \cdot \overline{V}_{pac,dss}^{(i-1)} \tag{4.17}$$

Estimados os parâmetros do equivalente de Norton da distribuição, devem ser realizadas duas atualizações. Na primeira atualização, a admitância $\overline{y}_{dss}^{(i)}$ é adicionada à matriz de admitâncias nodais da rede de transmissão ($\mathbf{Y}$), conforme mostrado em (4.18a). Na segunda atualização, a corrente de ajuste $\overline{I}_{adj,dss}^{(i)}$ é utilizada para calcular a potência de ajuste $\overline{S}_{adj,dss}^{(i)}$, conforme a equação (4.18b).

$$\mathbf{Y}^{(i)}(k,k) = \mathbf{Y}^{(i)}(k,k) + \overline{y}_{dss}^{(i)} \tag{4.18a}$$

$$\overline{S}_{adj,dss}^{(i)} = \overline{V}_{pac,dss}^{(i-1)} \cdot \left(\overline{I}_{adj,dss}^{(i)}\right)^{*} \tag{4.18b}$$

Com isso, as saídas do bloco **OpenDSS** são a matriz $\mathbf{Y}^{(i)}$ incrementada e a potência de ajuste $\overline{S}_{adj,dss}^{(i)}$, que servem como entrada para o bloco **Newton-Raphson** da Figura 19.

(d) Bloco **Newton-Raphson**: é o segundo bloco do regime permanente.

A mais recente potência de ajuste $\left(\overline{S}_{adj,dss}^{(i)}\right)$ é acoplada ao sistema de transmissão como uma nova carga a ser considerada durante as iterações "nr", ou iterações de *Newton-Raphson*. Além disso, considera-se também a mais recente matriz $\mathbf{Y}^{(i)}$.

Como critério de convergência de cada iteração de *Newton-Raphson*, são escolhidas as potências ativas e reativas líquidas das barras da rede de transmissão, conforme a equação matricial em (4.19), onde se mostra a matriz jacobiana a ser recalculada a cada iteração "nr".

$$\begin{bmatrix} \Delta\mathbf{P} \\ \Delta\mathbf{Q} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial\mathbf{P}}{\partial\theta} & \dfrac{\partial\mathbf{P}}{\partial\mathbf{V}} \\ \dfrac{\partial\mathbf{Q}}{\partial\theta} & \dfrac{\partial\mathbf{Q}}{\partial\mathbf{V}} \end{bmatrix} \cdot \begin{bmatrix} \Delta\theta \\ \Delta\mathbf{V} \end{bmatrix} \tag{4.19}$$

Após a convergência de *Newton-Raphson* da rede, tem-se um novo vetor de tensões nodais da rede $\left(\mathbf{v}^{(i)}\right)$. Considerando que a barra k corresponde ao PAC da distribuição, a saída do bloco **Newton-Raphson** é um novo valor de $\overline{V}_{pac,dss}^{(i)}$, que agora passa a ser obtido da forma apresentada na equação (4.20).

$$\overline{V}_{pac,dss}^{(i)} = \mathbf{v}^{(i)}(k) \tag{4.20}$$

Com isso, encerra-se a iteração "nr-dss".

(e) Teste de convergência da iteração "nr-dss": com o intuito de verificar se são necessárias mais iterações "nr-dss", deve-se calcular a diferença entre a tensão do PAC da iteração atual i e a tensão anterior, conforme mostrado na equação (4.21).

$$\Delta V_{pac,dss}^{(i)} = \overline{V}_{pac,dss}^{(i)} - \overline{V}_{pac,dss}^{(i-1)} \tag{4.21}$$

O processo iterativo do regime permanente é encerrado quando os erros absolutos e relativos de tensão do PAC da distribuição atendem às relações apresentadas em (4.22).

$$\begin{cases} |\Delta V_{pac,dss}^{(i)}| \leq \varepsilon_{abs} \\ |\Delta V_{pac,dss}^{(i)}| \leq V_{pac,dss}^{(i-1)} \cdot \varepsilon_{rel} \end{cases} \tag{4.22}$$

Caso não sejam atendidas, realiza-se o incremento do contador de iterações do conjunto nr-dss ($i = i+1$), o incremento da matriz $\mathbf{Y}$ (equação (4.18a)) é desfeito e a mais recente tensão $\overline{V}_{pac,dss}$ serve novamente como entrada para o bloco **OpenDSS** da Figura 19, onde se inicia uma nova iteração "nr-dss" no item (b).

Após a convergência do regime permanente, estão definidas a tensão e a corrente do PAC da distribuição ($\overline{V}_{0,pac,dss}$ e $\overline{I}_{0,pac,dss}$, respectivamente). Portanto, antes do cálculo do regime transitório, utiliza-se a equação (4.23a) para calcular a admitância $\overline{y}_{0,dss}$, que é diferente da admitância do método de sensibilidade. O valor de $\overline{y}_{0,dss}$ é fixo durante os cálculos do regime transitório e, portanto, pode ser adicionado à matriz de admitâncias nodais da rede de transmissão ($\mathbf{Y}$), conforme mostrado em (4.23b).

$$\overline{y}_{0,dss} = \frac{\overline{I}_{0,pac,dss}}{\overline{V}_{0,pac,dss}} \tag{4.23a}$$

$$\mathbf{Y}(k,k) = \mathbf{Y}(k,k) + \overline{y}_{0,dss} \tag{4.23b}$$

4.3.3 Regime transitório com OpenDSS

Neste trabalho, para a simulação do regime transitório, o cálculo do método alternado do instante atual (t) é iniciado pela extrapolação do vetor de tensões nodais $\left(\mathbf{v}^{(0)}\right)$ da rede de transmissão. Conforme a equação (4.24), essa extrapolação é realizada com base nas tensões de um instante de tempo anterior ($t - \Delta t$) e de dois instantes anteriores ($t - 2\Delta t$). Possui a vantagem de aumento da velocidade de convergência (STOTT, 1979).

$$\begin{aligned} \mathbf{v}^{(0)}(t) &= \mathbf{v}^{(\text{extrapol})}(t) \\ &= \frac{\mathbf{v}^2(t-\Delta t)}{\mathbf{v}(t-2\Delta t)} \end{aligned} \tag{4.24}$$

Em seguida, executa-se um passo preditor para o vetor inicial de variáveis de estado ($\mathbf{x}^{(0)}$) do modelo de máquinas. Para isso, adota-se o método de Euler explícito, conforme explicado anteriormente na Seção 2.4. Com isso, tem-se a equação mostrada em (4.25).

$$\mathbf{x}^{(0)}(t) = \mathbf{x}(t - \Delta t) + \Delta t \cdot \mathbf{f}(t - \Delta t) \tag{4.25}$$

Com as tensões nodais e os estados estimados, inicia-se a primeira iteração "modelo-rede-dss", ou iteração do conjunto modelo de máquinas mais rede de transmissão mais OpenDSS, conforme mostrado na Figura 20.

4.3.3.1 Bloco **Modelo de máquinas**

Para iniciar a i-ésima iteração do conjunto modelo-rede-dss, armazena-se o mais recente vetor de tensões nodais $\left(\mathbf{v}^{(i-1)}\right)$. Quando se tem $i = 1$, esse vetor corresponde às tensões extrapoladas.

Em seguida, a partir de $\mathbf{v}^{(i-1)}$, são extraídas as tensões terminais dos geradores ($\mathbf{v}_g$), conforme mostrado em (4.26).

$$\mathbf{v}_g^{(i-1)} \leftarrow \mathbf{v}^{(i-1)} \tag{4.26}$$

O vetor $\mathbf{v}_g^{(i-1)}$ serve como entrada para o modelo de máquinas que, neste trabalho, está compactado em uma FMU e é solucionado pelo método de integração trapezoidal, conforme explicado anteriormente na Seção 2.4. Com isso, a FMU entrega ao ambiente de simulação as tensões internas dos geradores $\left(\overline{E}_{int}\right)$, conforme mostrado em (4.27).

$$\overline{V}_g^{(i-1)} \xrightarrow[\text{máquinas da FMU}]{\text{Modelo de}} \overline{E}_{int}^{(i)} \tag{4.27}$$

Em seguida, o ambiente de simulação utiliza os valores de $\overline{E}_{int}^{(i)}$ da iteração atual para calcular as correntes de Norton correspondentes, conforme mostrado em (4.28).

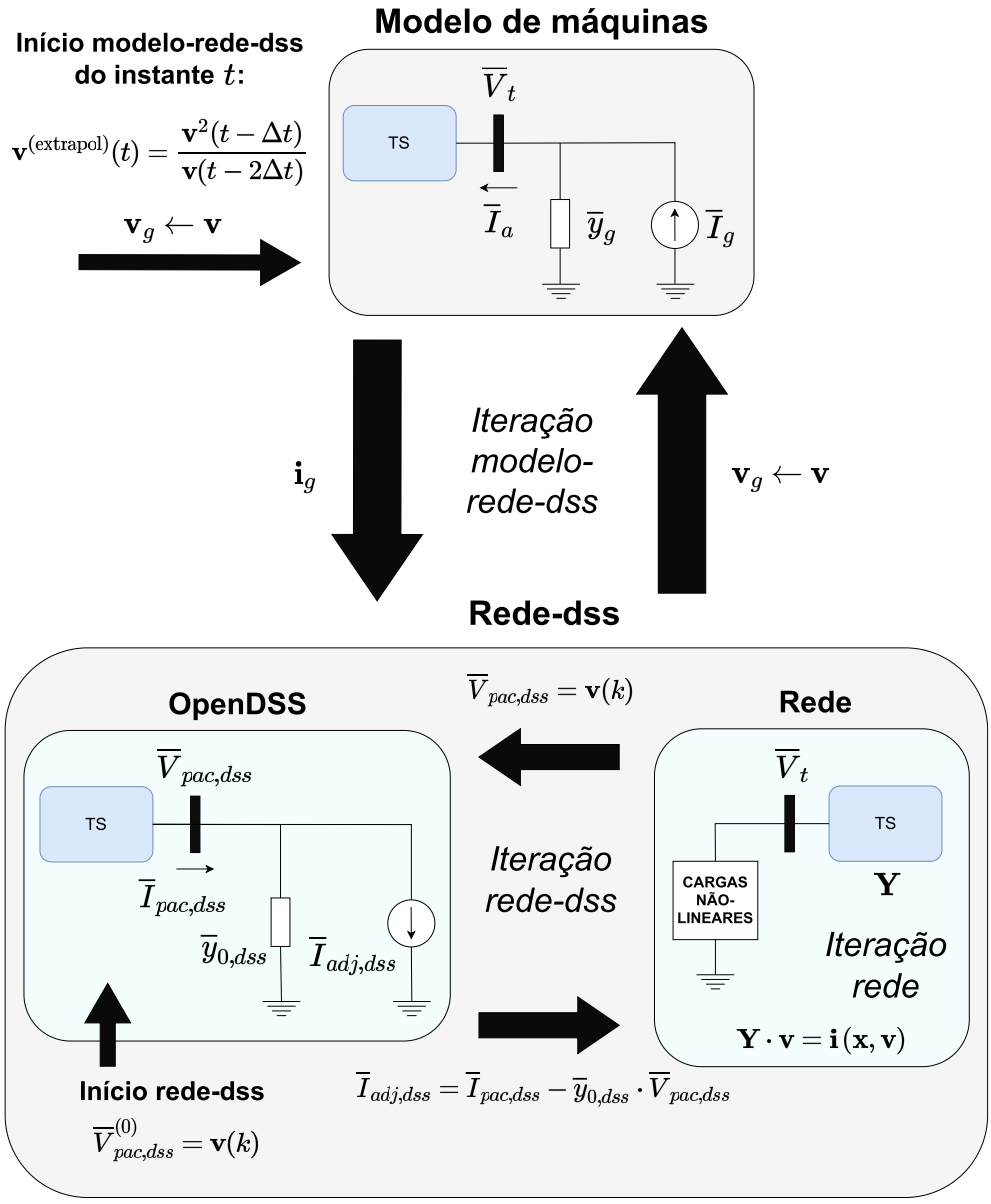
$$\overline{I}_g^{(i)} = \overline{y}_g \cdot \overline{E}_{int}^{(i)} \tag{4.28}$$

Com o valor de $\overline{I}_g^{(i)}$ de cada gerador, a saída do bloco **Modelo de máquinas** é o vetor de correntes de Norton da i-ésima iteração do conjunto modelo-rede-dss $\left(\mathbf{i}_g^{(i)}\right)$, que serve como entrada para o bloco **Rede-dss** da Figura 20.

4.3.3.2 Bloco **Rede-dss**

O mais recente vetor de correntes de Norton dos geradores $\left(\mathbf{i}_g^{(i)}\right)$ serve como entrada para o cálculo iterativo do conjunto rede-dss, que por sua vez possui 2 sub-blocos: **OpenDSS** e **Rede**. A cada iteração "rede-dss" (ou iteração j), o vetor $\mathbf{i}_g^{(i)}$ se mantém constante.

Figura 20 – Regime transitório com OpenDSS.



Fonte: Elaborada pelo autor (2023)

A execução do bloco **Rede-dss** é iniciado por uma estimativa inicial de $\overline{V}_{pac,dss}$, conforme mostrado na equação (4.29), onde $\mathbf{v}^{(i-1)}(k)$ é a tensão do PAC da distribuição do mais recente vetor de tensões nodais.

$$\overline{V}^{(0)}_{pac,dss} = \mathbf{v}^{(i-1)}(k) \tag{4.29}$$

A tensão $\overline{V}^{(0)}_{pac,dss}$ serve como entrada para o sub-bloco **OpenDSS** da Figura 20 e inicia a primeira iteração "rede-dss", ou iteração do conjunto rede de transmissão mais OpenDSS.

Descrição do bloco **Rede-dss**:

(a) Bloco **OpenDSS**: é o primeiro sub-bloco do bloco **Rede-dss**.

Primeiro, a corrente do PAC da distribuição da j-ésima iteração $\left(\overline{I}^{(j)}_{pac,dss}\right)$ é obtida através de dois passos: fornecer a mais recente tensão do PAC da distribuição $\left(\overline{V}^{(j-1)}_{pac,dss}\right)$ ao módulo do OpenDSS e habilitar a execução do fluxo de potência da distribuição. Em seguida, a corrente de ajuste de Norton da distribuição $\left(\overline{I}^{(j)}_{adj,dss}\right)$ é calculada através da equação (4.30).

$$\overline{I}^{(j)}_{adj,dss} = \overline{I}^{(j)}_{pac,dss} - \overline{y}_{0,dss} \cdot \overline{V}^{(j-1)}_{pac,dss} \tag{4.30}$$

Com isso, a saída do sub-bloco **OpenDSS** é a corrente $\overline{I}^{(j)}_{adj,dss}$, que serve como entrada para o sub-bloco **Rede** da Figura 20.

(b) Bloco **Rede**: é o segundo sub-bloco do bloco **Rede-dss**.

O mais recente vetor de correntes de Norton dos geradores $\left(\mathbf{i}^{(i)}_{g}\right)$ e a mais recente corrente de ajuste de Norton da distribuição $\left(\overline{I}^{(j)}_{adj,dss}\right)$ servem como entradas para o cálculo iterativo da rede de transmissão. A cada iteração "rede" (ou iteração r), o vetor $\mathbf{i}^{(i)}_{g}$ e a corrente $\overline{I}^{(j)}_{adj,dss}$ se mantêm constantes.

Além disso, o mais recente vetor de tensões nodais da rede $\left(\mathbf{v}^{(r-1)}\right)$ é utilizado para duas finalidades: primeiro, atualizar a matriz de admitâncias nodais da rede $\left(\mathbf{Y}^{(r)}\right)$ em caso de baixas tensões e, segundo, calcular o vetor de correntes das cargas não lineares $\left(\mathbf{i}^{(r)}_{l}\right)$, conforme mostrado em (4.31).

$$\mathbf{v}^{(r-1)} \xRightarrow[\text{não lineares}]{\text{Cargas}} \mathbf{Y}^{(r)}, \mathbf{i}^{(r)}_{l} \tag{4.31}$$

Com os vetores de correntes dos geradores e das cargas, calcula-se o vetor de injeções de correntes nodais $\left(\mathbf{i}^{(r)}\right)$ através da equação (4.32a). Logo depois, a corrente $\overline{I}^{(j)}_{adj,dss}$ é adicionada ao k-ésimo termo desse vetor, conforme mostrado em (4.32b).

$$\mathbf{i}^{(r)} = \mathbf{i}^{(i)}_{g} - \mathbf{i}^{(r)}_{l} \tag{4.32a}$$

$$\mathbf{i}^{(r)}(k) = \mathbf{i}^{(r)}(k) - \overline{I}^{(j)}_{adj,dss} \tag{4.32b}$$

Em seguida, o vetor de tensões nodais $\left(\mathbf{v}^{(r)}\right)$ é calculado pelo método nodal, conforme a equação (4.33).

$$\mathbf{Y}^{(r)} \cdot \mathbf{v}^{(r)} = \mathbf{i}^{(r)} \tag{4.33}$$

A solução da rede é explicada em maiores detalhes na Subseção **3.1.1**. Porém, com a presença do módulo do OpenDSS, o vetor $\mathbf{i}$ é obtido não só por $\mathbf{i}_g$ e $\mathbf{i}_l$, como também pelo valor de $\overline{I}_{adj,dss}$.

Após a convergência da rede, tem-se um novo vetor de tensões nodais da rede $\left(\mathbf{v}^{(j)}\right)$. Considerando que a barra k corresponde ao PAC da distribuição, a saída do sub-bloco **Rede** é um novo valor de $\overline{V}^{(j)}_{pac,dss}$, conforme mostrado em (4.34).

$$\overline{V}^{(j)}_{pac,dss} = \mathbf{v}^{(j)}(k) \tag{4.34}$$

Com isso, encerra-se a iteração "rede-dss".

(c) Teste de convergência da iteração "rede-dss": com o intuito de verificar se são necessárias mais iterações "rede-dss", deve-se calcular a diferença entre a tensão do PAC da iteração atual j e a tensão anterior, conforme mostrado na equação (4.35).

$$\Delta V^{(j)}_{pac,dss} = \overline{V}^{(j)}_{pac,dss} - \overline{V}^{(j-1)}_{pac,dss} \tag{4.35}$$

O processo iterativo do conjunto rede-dss é encerrado quando os erros absolutos e relativos de tensão do PAC da distribuição atendem às relações apresentadas em (4.36).

$$\begin{cases} |\Delta V^{(j)}_{pac,dss}| \leq \varepsilon_{abs} \\ |\Delta V^{(j)}_{pac,dss}| \leq V^{(j-1)}_{pac,dss} \cdot \varepsilon_{rel} \end{cases} \tag{4.36}$$

Caso não sejam atendidas, realiza-se o incremento do contador de iterações do conjunto rede-dss ($j = j + 1$) e a mais recente tensão $\overline{V}_{pac,dss}$ serve novamente como entrada para o sub-bloco **OpenDSS** da Figura 19, onde se inicia uma nova iteração "rede-dss" no item (a).

Após a convergência do conjunto rede-dss, a saída do bloco **Rede-dss** é um novo vetor $\mathbf{v}^{(i)}$. Com isso, encerra-se a iteração "modelo-rede-dss".

4.3.3.3 Teste de convergência da iteração "modelo-rede-dss"

Com o intuito de verificar se são necessárias mais iterações "modelo-rede-dss", deve-se calcular a diferença entre o vetor da iteração atual i e o vetor anterior das tensões nodais, conforme mostrado na equação (4.37).

$$\Delta \mathbf{v}^{(i)} = \mathbf{v}^{(i)} - \mathbf{v}^{(i-1)} \tag{4.37}$$

Algoritmo 10 – Inicialização do sistema de distribuição de 38 barras em *Python*.

```
1 import opendssdirect as dss

2 # Importação do sistema de distribuição de 38 barras:
3 dss.run_command("Redirect '/diretório/Run_bus38.dss'")

4 # Número máximo de iterações e critério de convergência:
5 dss.Solution.MaxIterations(500)
6 dss.Solution.Convergence(1e-6)
```

Fonte: Elaborado pelo autor (2024)

O processo iterativo do método alternado é encerrado quando os erros absolutos e relativos do vetor atendem às relações apresentadas em (4.38).

$$\begin{cases} \max |\Delta \mathbf{v}^{(i)}| \leq \varepsilon_{abs} \\ \max |\Delta \mathbf{v}^{(i)}| \leq \max |\mathbf{v}^{(i-1)}| \cdot \varepsilon_{rel} \end{cases} \tag{4.38}$$

Caso não sejam atendidas, realiza-se o incremento do contador de iterações do conjunto modelo-rede-dss ($i = i + 1$) e o mais recente vetor $\mathbf{v}$ serve novamente como entrada para o bloco **Modelo de máquinas** da Figura 20, onde se inicia uma nova iteração "modelo-rede-dss" na Subsubseção *4.3.3.1*.

4.4 INCLUSÃO DO SISTEMA DE DISTRIBUIÇÃO AO PROGRAMA DE ESTABILIDADE

Dentro do programa de estabilidade implementado em *Python*, o sistema de distribuição é inicializado e comandado através do módulo `OpenDSSDirect`, conforme mostrado no Algoritmo 10.

Com o `OpenDSSDirect.py`, torna-se possível que o ambiente de simulação em *Python* realize a leitura do arquivo `.dss` de dados da rede de distribuição, bem como a execução das funções do DSS, sem a necessidade de instalação do programa do EPRI. Sabendo disso, pode-se realizar a importação como um módulo `dss` (linha 1), que por sua vez possui submódulos que equivalem a interfaces para vários componentes do OpenDSS (KRISHNAMURTHY, 2017).

Com o módulo importado, deve-se realizar a importação dos dados da rede de distribuição (por exemplo, uma rede de 38 barras contida num arquivo `Run_bus38.dss`) para o ambiente de simulação através do comando `run_command` (linha 3). Em seguida, tem-se a opção de definir dois parâmetros de execução da solução do sistema: número máximo de iterações (igual a `500`, por exemplo) e critério de convergência (igual a `1e-6`, por exemplo).

Algoritmo 11 – Inicialização dos tipos das cargas e dos geradores do objeto `dss` para o regime permanente.

```
1  # -------- Varredura das cargas do sistema de distribuição:
2  num_loads_dss = dss.Loads.Count()
3  names_loads_dss = dss.Loads.AllNames()
4  for l in range(num_loads_dss):

5    # Ativação da carga através do seu nome:
6    dss.Loads.Name(names_loads_dss[l])

7    # Definição dos modelos de carga como Pcte no regime permanente:
8    dss.Loads.Model(1)

9  # -------- Varredura dos geradores do sistema de distribuição:
10 num_gen_dss = dss.Generators.Count()
11 names_gen_dss = dss.Generators.AllNames()
12 for g in range(num_gen_dss):

13   # Ativação do gerador através seu nome:
14   dss.Generators.Name(names_gen_dss[g])

15   # Definição dos modelos de geração como PV para o regime permanente:
16   dss.Generators.Model(3)
```

Fonte: Elaborado pelo autor (2024)

Após a importação do arquivo `.dss` correspondente, alguns de seus dados podem ser alterados dentro do ambiente *Python* através de comandos específicos, cuja execução está apresentada nas subseções a seguir.

4.4.1 Regime permanente

Para a execução de regime permanente do sistema de distribuição, há a possibilidade de adicionar novos dados ao objeto `dss`, como por exemplo os tipos desejados das cargas e dos geradores da distribuição, conforme o Algoritmo 11.

Para a definição do tipo de cada carga contida na biblioteca `Loads`, realizam-se os seguintes procedimentos: primeiro, utiliza-se o comando `dss.Loads.AllNames` (linha 3) para criar um vetor de armazenamento dos nomes de todas as cargas presentes no objeto `dss`. Em seguida, realiza-se uma varredura dessas cargas (linhas de 4 a 8), para que seja ativada uma carga de cada vez através dos seus respectivos nomes. Uma vez ativada, pode-se definir o modelo de cada uma através do comando `dss.Loads.Model` (linha 8). Conforme o manual do OpenDSS em (DUGAN; MONTENEGRO, 2021), a entrada dessa função pode ser um número de `1` a `8`, de modo que cada número corresponde a um modelo diferente de carga. No caso de modelagem de carga do tipo potência constante, a entrada correspondente é o número `1`.

Para a definição do tipo de gerador para cada elemento contido na biblioteca `Generators` (linhas de 10 a 16), realizam-se os procedimentos análogos aos tipos de carga. No entanto, a entrada da função `dss.Generators.Model` pode ser um número de `1` a `7`.

Algoritmo 12 – Função da solução do DSS.

```
1  def dss_solution(dss, Vpac_dss)

2    # ----------- Entrada:
3    # - dss: objeto contendo os parâmetros do opendssdirect.
4    # - Vpac_dss: tensão nodal do PAC do dss.

5    # ----------- Saída:
6    # - Ipac_dss: corrente do PAC do dss.

7    # Entrada de módulo (em pu) e ângulo (em graus) do Vpac_dss:
8    dss.Vsources.PU( np.abs(Vpac_dss) )
9    dss.Vsources.AngleDeg( np.angle(Vpac_dss) * 180/np.pi )

10   # Execução da solução e teste de convergência do dss:
11   dss.Solution.Solve()
12   if dss.Solution.Converged() == False:
13     print("O cálculo do dss não convergiu.")
14     exit(1)

15   # Ativação do elemento de onde se deseja extrair os resultados da solução: barra de
     ↪  referência.
16   dss.Circuit.SetActiveElement("Vsource.source")

17   # Corrente de sequência positiva (em [pu-A]):
18   Ipac_dss = dss.CktElement.CplxSeqCurrents()[2] + 1j * dss.CktElement.CplxSeqCurrents()[3]
19   Ipac_dss = Ipac_dss / Ibase

20   return Ipac_dss
```

Fonte: Elaborado pelo autor (2024)

No caso de modelagem de gerador PV, a entrada correspondente é o número `3`.

Com a definição dos elementos de conversão de energia do `dss` (cargas do tipo potência constante e geradores do tipo PV), realiza-se o cálculo do regime permanente descrito na Subseção **4.3.2**. Para cada iteração "nr-dss", o cálculo da corrente do PAC da distribuição (mostrado na função f da equação (4.9)) equivale à função `dss_solution` mostrada no Algoritmo 12.

A função `dss_solution` possui duas entradas: o módulo `dss` e o valor de `Vpac_dss`, que é a tensão nodal do PAC (ou da barra de referência) da rede de distribuição. Primeiro, nas linhas 8 e 9, atribui-se à biblioteca `dss.Vsources` o módulo (em [pu]) e o ângulo (em graus) da tensão do PAC. Em seguida, o comando `dss.Solution.Solve` (linha 11) executa o fluxo de potência da distribuição e, logo depois, realiza-se um teste de convergência através da verificação da variável binária `dss.Solution.Converged` (linha 12). Caso a solução não seja convergente, o programa é interrompido e imprime uma mensagem de erro.

Caso o fluxo da rede de distribuição seja convergente, a função `dss_solution` ativa o elemento da rede de onde se deseja extrair os resultados. Como esse elemento é a barra de referência da distribuição, a ativação é realizada pelo nome escolhido `Vsource.source` (linha 16). Com isso, o comando `dss.CktElement.CplxSeqCurrents` (linha 18) é utilizado

Algoritmo 13 – Função do cálculo do equivalente de Norton do OpenDSS.

```
1  def norton_dss(dss, Vpac_dss)

2    # ----------- Entrada:
3    # - dss: objeto contendo os parâmetros do opendssdirect.
4    # - Vpac_dss: tensão nodal do PAC do dss.

5    # ----------- Saída:
6    # - y_dss: admitância de Norton do dss.
7    # - Sadj_dss: potência de ajuste de Norton do dss.

8    # Incremento da tensão nodal do PAC da distribuição:
9    dVpac_dss = 0.001

10   # Corrente correspondente ao incremento positivo de Vpac_dss:
11   V0_pos = Vpac_dss + dVpac_dss
12   I0_pos = dss_solution(dss, V0_pos)

13   # Corrente correspondente ao incremento negativo de Vpac_dss:
14   V0_neg = Vpac_dss - dVpac_dss
15   I0_neg = dss_solution(dss, V0_neg)

16   # Admitância de Norton do dss:
17   y_dss = (I0_pos - I0_neg) / (2 * dVpac_dss)

18   # Corrente e potência de ajuste de Norton do dss:
19   Iadj_dss = Ipac_dss - y_dss * Vpac_dss
20   Sadj_dss = Vpac_dss * np.conjugate(Iadj_dss)

21   return y_dss, Sadj_dss
```

Fonte: Elaborado pelo autor (2024)

para obter a corrente do PAC do `dss` em [A]. Por último, divide-se o valor obtido pela corrente base (`Ibase`), para que a saída da função seja a corrente `Ipac_dss` em [pu].

Definida a função de solução da distribuição, a mesma é utilizada para implementar uma função destinada ao cálculo dos parâmetros do equivalente de Norton do OpenDSS, conforme mostrado no Algoritmo 13. Nesse algoritmo, a admitância de Norton (`y_dss`) é calculada com base na equação (4.15), isto é, com base no método de sensibilidade central, que considera incrementos positivos e negativos da tensão nodal do PAC da distribuição.

A primeira etapa da função `norton_dss` é a inicialização do incremento da tensão `Vpac_dss`. Neste trabalho, o valor escolhido para o incremento é `dVpac_dss = 0.001` [pu] (linha 9). Em seguida, utiliza-se a função `dss_solution` para calcular as correntes correspondentes ao incremento positivo (linha 12) e negativo (linha 15) de `Vpac_dss`. Com as duas correntes `I0_pos` e `I0_neg`, calcula-se a admitância de Norton da rede de distribuição (`y_dss` - linha 17). Essa admitância, por sua vez, é utilizada para calcular a corrente de ajuste de Norton (linha 19), tornando possível o cálculo da potência de ajuste correspondente (linha 20).

Com isso, as saídas da função `norton_dss` são os parâmetros atualizados de Norton (`y_dss` e `Sadj_dss`), que são considerados durante a solução de *Newton-Raphson* da rede

Algoritmo 14 – Atualização de parâmetros do objeto dss para o regime transitório.

```
1  # Varredura das cargas normais e das cargas negativas:
2  num_loads_and_gen = dss.Loads.Count()
3  names_loads_and_gen = dss.Loads.AllNames()
4  for l in range(num_loads_and_gen):

5    # Ativação da carga através do seu nome:
6    dss.Loads.Name(names_loads_and_gen[l])

7    # Definição dos modelos de carga como Zcte para o regime transitório:
8    dss.Loads.Model(2)

9  # Modo de controle: tempo.
10 dss.Solution.ControlMode(2)

11 # Modo de solução: dinâmico.
12 dss.Solution.Mode(14)

13 # Passo de tempo:
14 dss.Solution.StepSize(dt)

15 # Algoritmo: normal (modo de injeção de corrente).
16 dss.Solution.Algorithm(0)
```

Fonte: Elaborado pelo autor (2024)

de transmissão.

4.4.2 Regime transitório

Após a convergência de regime permanente da rede de distribuição em conjunto com a transmissão, é necessário atualizar determinados parâmetros para a simulação do regime transitório (descrito na Subseção **4.3.3**). Essas definições estão mostradas no Algoritmo 14.

Para o regime transitório, a modelagem escolhida para as cargas e os geradores é do tipo impedância constante. No caso dos geradores, os elementos do objeto `Generator` devem ser substituídos por cargas negativas, cujas potências devem ser definidas como iguais aos valores negativos das gerações (ativas e reativas) calculadas no regime permanente. Essa substituição se deve à ausência de modelos de geradores no OpenDSS que possam ser empregados para a mesma estratégia que o programa ANATEM adota para barras de geração sem modelos dinâmicos associados (ELETROBRAS CEPEL, 2020). Vale ressaltar também que, no modo dinâmico do OpenDSS, o objeto `Load` é convertido para uma admitância que é incluída à matriz $\mathbf{Y}_{dss}$ da distribuição, enquanto o objeto `Generator` é convertido para uma fonte de tensão atrás de uma impedância.

Após a substituição, tem-se as denominadas cargas normais e cargas negativas (ambas dentro da biblioteca `dss.Loads`). Portanto, para ambos os elementos, a entrada do comando `dss.Loads.Model` é definida como igual a `2` (linha 8).

Em seguida, deve-se definir outros parâmetros associados ao regime transitório do

`dss.Solution`, como por exemplo (DUGAN; MONTENEGRO, 2021):

- Modo de controle (`ControlMode` - linha 10): tempo (entrada igual a `2`). As ações de controle são executadas quando o instante de tempo é alcançado ou ultrapassado. Utilizado para o modo dinâmico.
- Modo de solução (`Mode` - linha 12): dinâmico (entrada igual a `14`). Precisa ser antecedido por uma solução de fluxo de potência convergente.
- Tamanho do passo de integração (`StepSize` - linha 14): a unidade padrão de tempo é em segundos. Neste trabalho, tem-se `dt = 0.001` [s].
- Algoritmo (`Algorithm` - linha 16): normal (entrada igual a `0`). Consiste em iterações de injeção de corrente de ponto fixo. É adequado para a maioria dos sistemas de distribuição. Sua execução é cerca de duas vezes mais rápida em relação à iteração de *Newton*, conforme dito em (DUGAN; MONTENEGRO, 2021).

Com os parâmetros definidos, realiza-se o cálculo do regime transitório descrito na Subseção **4.3.3**. Para cada iteração "rede-dss", o cálculo de `Ipac_dss` equivale à função `dss_solution` mostrada anteriormente no Algoritmo 12.

4.5 CONCLUSÕES PARCIAIS

Neste capítulo, buscou-se continuar o desenvolvimento do protótipo de estabilidade transitória apresentado no capítulo anterior. Para isso, foi proposta a inclusão de um terceiro subsistema, correspondente ao sistema elétrico de distribuição, através de um módulo baseado no programa OpenDSS. Sabendo disso, foram discutidas algumas características e vantagens que incentivaram a adoção do programa do EPRI, como por exemplo a linguagem de código aberto e a solução de fluxo de potência através do método da iteração de ponto fixo baseada na matriz de admitâncias nodais.

Com a inclusão de um subsistema de distribuição ao restante da simulação computacional, não só as etapas de solução do regime transitório precisaram ser adaptadas, como também as do regime permanente. Sabendo disso, este capítulo apresentou métodos de sensibilidade envolvendo variações em torno de valores estimados de tensão. Esses métodos, por sua vez, são úteis para acelerar a convergência do regime permanente quando o sistema de distribuição possui equipamentos de natureza altamente não linear.

Por fim, foram apresentados os principais comandos em *Python* necessários para importar o sistema de distribuição e alterar os parâmetros de simulação conforme a necessidade do usuário, como por exemplo o modelo das cargas, modelo dos geradores, modo de solução e algoritmo. Além disso, também foram criadas funções específicas para o cálculo do fluxo de potência e do equivalente de Norton da rede de distribuição,

onde foi adotado o método de sensibilidade central. Com isso, foi possível observar que o módulo `OpenDSSDirect` permite o usuário adaptar a rede de distribuição de diversas formas diretamente pelo ambiente *Python*, ou seja, sem a necessidade de instalação do programa do EPRI.

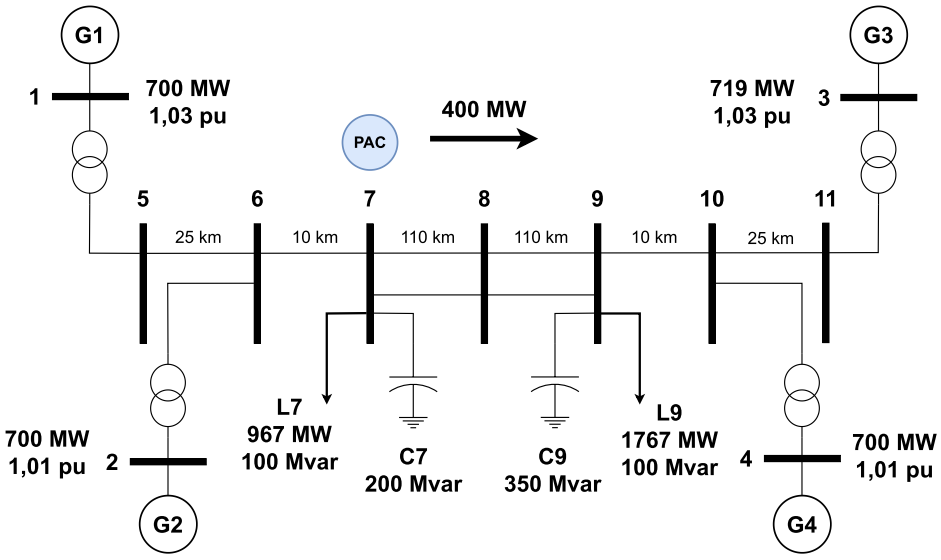
5 IMPLEMENTAÇÃO DO MÉTODO ALTERNADO E RESULTADOS OBTIDOS

Considerando os conceitos e as modelagens apresentadas nos capítulos anteriores, cria-se um protótipo em *Python* com o objetivo de implementar o método iterativo alternado em conjunto com a interface FMI. Para testar esse protótipo, executa-se uma simulação no tempo do sistema de transmissão de 11 barras, representado por seu equivalente de sequência positiva. Esse sistema pode ser visto na Figura 21 (KUNDUR, 1994), e os seus dados completos (assim como os dados dos equipamentos dinâmicos correspondentes) estão apresentados no Apêndice D.

A Seção 5.1 apresenta os parâmetros adotados para a simulação no tempo, bem como os resultados obtidos do sistema elétrico. Além disso, os programas ANAREDE (Análise de Redes Elétricas) e ANATEM (Análise de Transitórios Eletromecânicos), ambos desenvolvidos pelo CEPEL (Centro de Pesquisa em Energia Elétrica), são utilizados como referência para a validação dos resultados de regime permanente e transitório, respectivamente.

Em seguida, na Seção 5.2, utiliza-se um módulo do programa OpenDSS para que um sistema de distribuição seja acoplado ao sistema elétrico inicial. Além dos mesmos parâmetros da Seção 5.1, a nova simulação no tempo adota um método específico para a solução da rede, porém a validação dos resultados obtidos é realizada da mesma forma.

Figura 21 – Sistema de transmissão de 11 barras.



Fonte: Elaborada pelo autor (2023)

Tabela 3 – Resultados de regime permanente do sistema de transmissão de 11 barras.

Barra	**V [pu]**	**Ângulo [°]**	**P_g [MW]**	**Q_g [Mvar]**
1	1,03	27,07	700,00	185,00
2	1,01	17,31	700,00	234,59
3	1,03	0,00	719,09	176,00
4	1,01	-10,19	700,00	202,05
5	1,0065	20,61	-	-
6	0,9781	10,52	-	-
7	0,9610	2,11	-	-
8	0,9486	-11,76	-	-
9	0,9714	-25,35	-	-
10	0,9835	-16,94	-	-
11	1,0083	-6,63	-	-

Fonte: Elaborada pelo autor (2024)

5.1 SIMULAÇÃO NO TEMPO

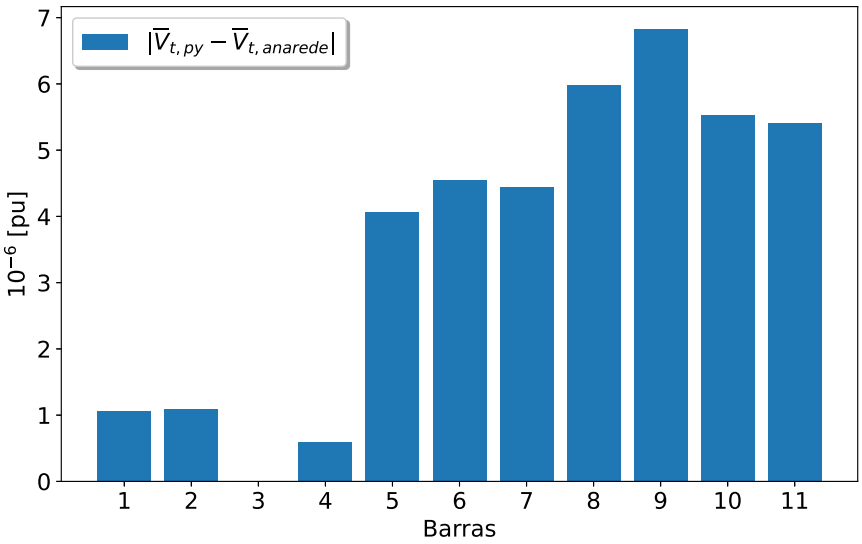
Antes da simulação no tempo do sistema de transmissão de 11 barras, é necessário calcular o ponto de operação correspondente ao regime permanente. Com isso, os parâmetros e os resultados desse cálculo são mostrados na Subseção **5.1.1**. Em seguida, apresentam-se na Subseção **5.1.2** os parâmetros e resultados do regime transitório.

5.1.1 Regime permanente

Para o início da execução do programa montado em ambiente *Python*, o regime permanente é calculado pelo método iterativo de *Newton-Raphson*, onde são adotadas uma tolerância absoluta de 10^{-6} [pu] e relativa de 10^{-4} % para as potências ativas e reativas líquidas das barras da rede de transmissão. Além disso, os outros parâmetros adotados para o cálculo iterativo de regime permanente são: modelagem do tipo potência constante para as cargas L_7 e L_9; modelagem do tipo PV para os geradores G_1, G_2 e G_4 e do tipo *slack* para o gerador G_3.

Com a execução do programa, a solução de regime permanente do sistema é encontrada em 4 iterações do *Newton-Raphson*. Nesse ponto de operação, as unidades geradoras G_1, G_2 e G_4 despacham 700 [MW] cada, enquanto G_3, que é o gerador de referência do sistema, despacha 719,09 [MW]. Com isso, a área 1 exporta cerca de 400 [MW] para a área 2. Esses e outros resultados de regime permanente do sistema de transmissão de 11 barras estão apresentados na Tabela 3. Além disso, a tabela destaca em vermelho os resultados do PAC da transmissão, onde será acoplado o sistema de distribuição (Seção 5.2).

Figura 22 – Comparações das tensões das barras do sistema de transmissão de 11 barras.



Fonte: Elaborada pelo autor (2024)

A validação das tensões nodais obtidas pelo *Python* é realizada através da comparação com o ANAREDE, conforme a Figura 22.

Com base na Figura 22, os erros das tensões apresentam um valor máximo de $6{,}8 \cdot 10^{-6}$ [pu] e uma média de $3{,}2 \cdot 10^{-6}$ [pu]. Portanto, pode-se considerar adequada a solução do regime permanente do sistema de transmissão de 11 barras através do protótipo em *Python*.

Vale ressaltar que a modelagem do tipo *slack* do gerador da barra 3 torna o seu erro de tensão igual a zero. Além disso, a modelagem PV das barras 1, 2 e 4 no regime permanente torna os seus erros de tensão menos expressivos em relação aos erros de outras barras da rede.

5.1.2 Regime transitório

Agora que se conhece o ponto de operação de regime permanente do sistema elétrico, realiza-se uma simulação no tempo de 10 [s], na qual um curto-circuito de impedância $\overline{Z}_{flt} = j0{,}001$ [pu] na base 100 [MVA] é aplicado à barra 8 da rede de transmissão no instante $t_{flt} = 0{,}2$ [s] e removido 100 [ms] depois, junto com a linha defeituosa entre as barras 8 e 9.

Os outros parâmetros adotados para a simulação no tempo são: passo de integração igual a 1 [ms]; modelagem do tipo impedância constante para as cargas L_7 e L_9; tolerância absoluta de 10^{-4} [pu] e relativa de 10^{-2} % para as tensões nodais da rede de transmissão (tanto na solução do conjunto modelo-rede quanto na solução individual da rede).

Além desses parâmetros, outra propriedade adotada para a simulação é a modelagem dos geradores da rede de transmissão através da FMU do tipo *Model Exchange*, apresentada no Capítulo 3. Sabendo disso, a FMU desenvolvida para este trabalho é utilizada para representar os geradores G_1, G_2, G_3 e G_4 da Figura 21.

Após a execução da simulação no tempo, são gerados os seguintes gráficos: número de iterações, ângulos internos relativos e tensões das barras dos geradores. Em seguida, realiza-se uma comparação dos resultados obtidos pelo *Python* em relação ao do programa ANATEM.

Durante a simulação no tempo, são contabilizados dois tipos de iterações: as iterações do conjunto "modelo-rede" e da rede, sendo que o segundo tipo está contido dentro do primeiro. Os gráficos correspondentes estão apresentados na Figura 23.

No instante de evento 0,2 [s], é necessário calcular apenas uma solução para o conjunto "modelo-rede", devido à continuidade do vetor das variáveis de estado ($\mathbf{x}$) do modelo de máquinas. No entanto, é necessário calcular duas soluções (uma antes e outra após o evento) para a rede, devido à descontinuidade do vetor de tensões nodais ($\mathbf{v}$) diante de perturbações. No caso da Figura 23(b), ambos os instantes $0{,}2^-$ e $0{,}2^+$ [s] exigem apenas 1 iteração "rede" cada. Caso não fosse aplicada a estimativa inicial da equação (3.28), o instante $0{,}2^+$ [s] exigiria mais de 1 iteração. Essas observações também se aplicam ao instante de evento 0,3 [s].

Além disso, também é possível observar na Figura 23(a) que, após a aplicação do curto-circuito, a simulação no tempo exige apenas 1 iteração "modelo-rede" por instante de tempo, exceto em instantes logo após os eventos da rede (0,201, 0,301 e 0,302 [s]), que exigem 2 iterações cada. Essa observação também se aplica às iterações "rede" da Figura 23(b).

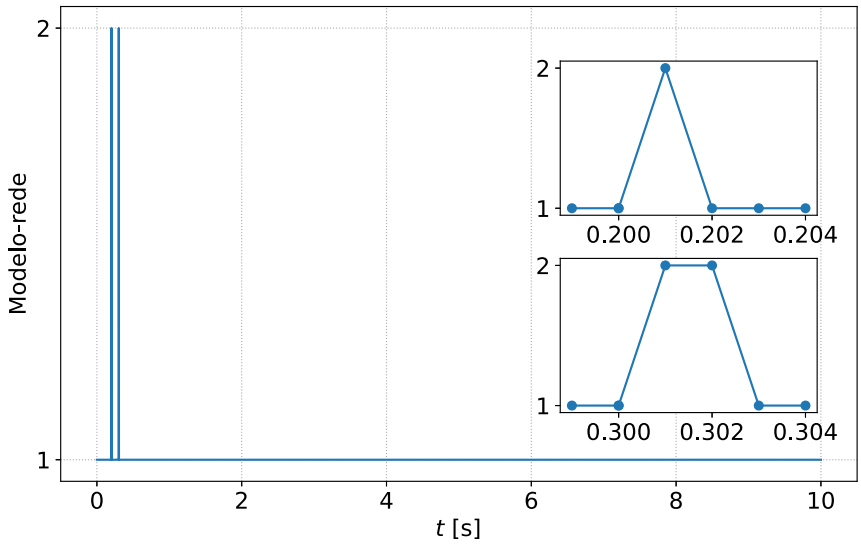
Essas observações a respeito das iterações da simulação no tempo podem ser justificadas pelos seguintes motivos:

- Os eventos da rede causam alterações na matriz de admitâncias nodais e, portanto, descontinuidades no vetor $\mathbf{v}$. Com isso, nos instantes imediatamente posteriores aos eventos, não é possível utilizar a extrapolação das tensões nodais, o que acarreta a necessidade de mais iterações para a convergência dos instantes 0,201, 0,301 e 0,302 [s].

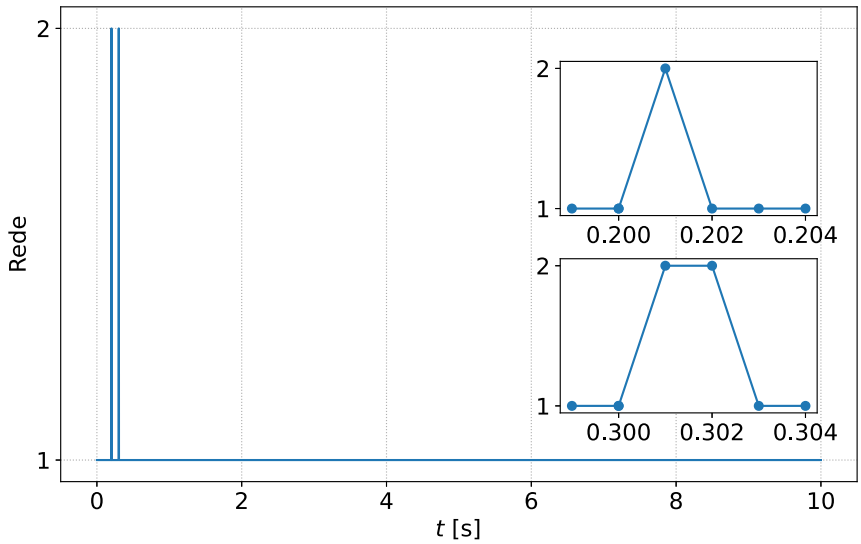
- Devido à modelagem de carga do tipo impedância constante, a solução da rede exige apenas 1 iteração para cada execução do conjunto "modelo-rede". Isto é, para cada iteração "modelo-rede", executa-se também uma iteração "rede".

Agora, apresentam-se graficamente os resultados dos geradores do sistema de transmissão de 11 barras. Os ângulos internos relativos estão mostrados na Figura 24 e as

Figura 23 – Número de iterações da simulação no tempo.



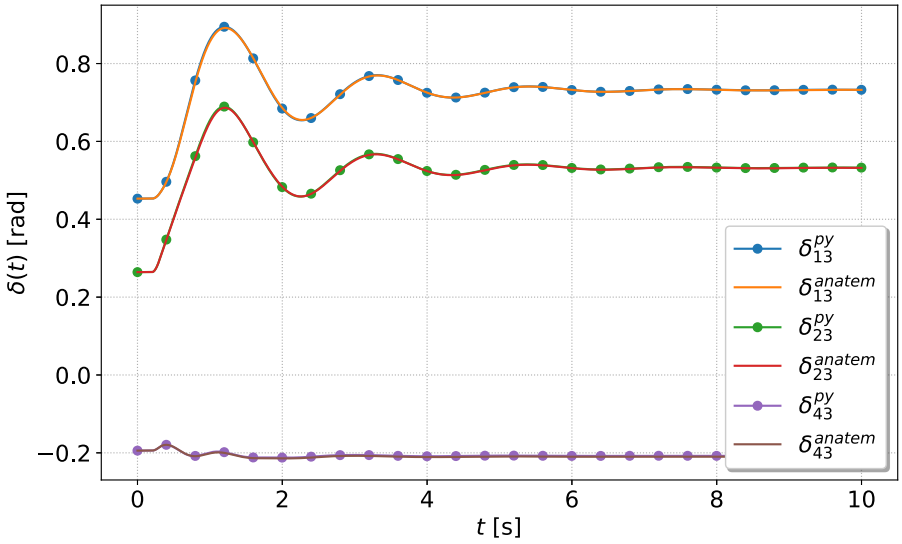
(a) Número de iterações “modelo-rede”.



(b) Número de iterações da rede.

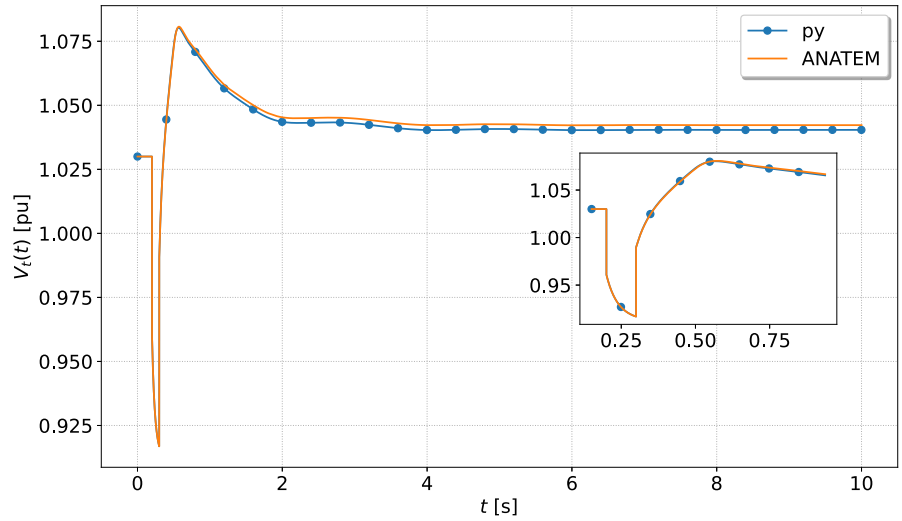
Fonte: Elaborada pelo autor (2024)

Figura 24 – Ângulos internos relativos dos geradores do sistema de transmissão de 11 barras.



Fonte: Elaborada pelo autor (2024)

Figura 25 – Tensão terminal do gerador G_1 do sistema de transmissão de 11 barras.



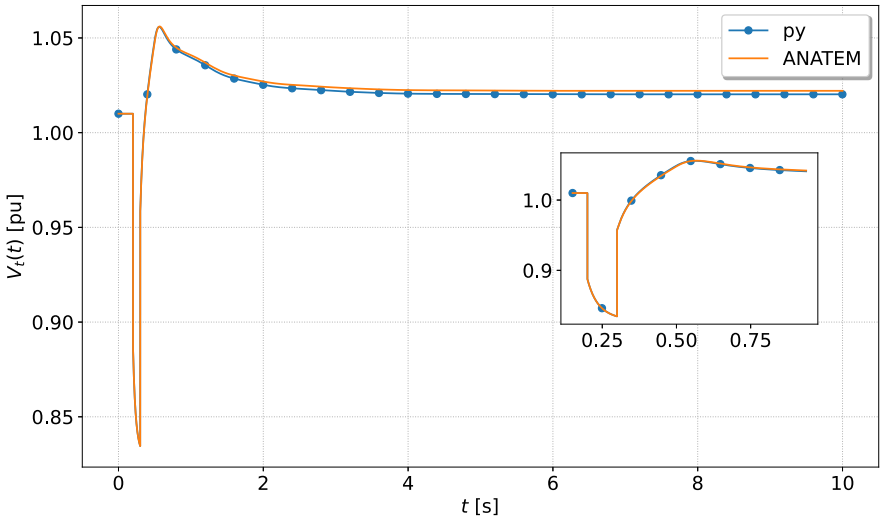
Fonte: Elaborada pelo autor (2024)

tensões terminais estão mostradas da Figura 25 até a Figura 28. É possível observar a proximidade gráfica entre os resultados do *Python* e os do ANATEM.

5.1.3 Comparação quantitativa dos resultados

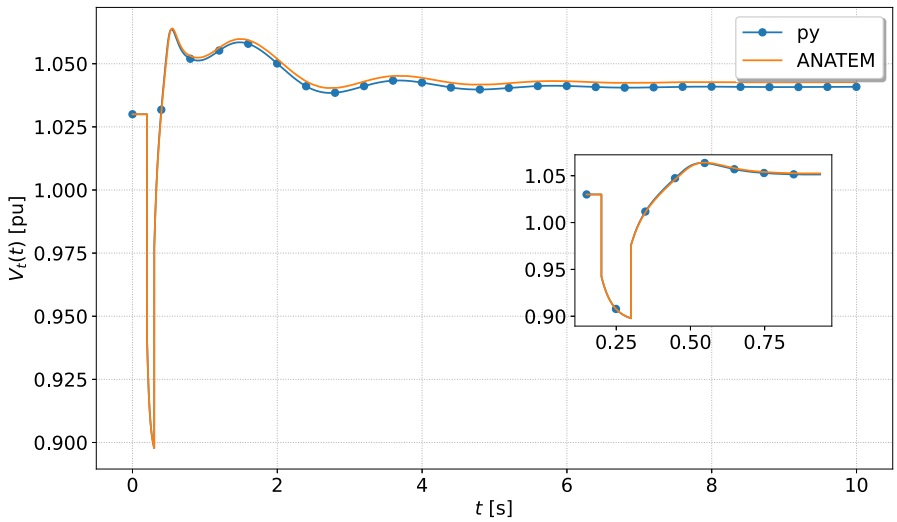
No entanto, apesar da proximidade gráfica, é necessário realizar uma verificação quantitativa da precisão dos resultados do protótipo em *Python* em relação aos resultados

Figura 26 – Tensão terminal do gerador G_2 do sistema de transmissão de 11 barras.



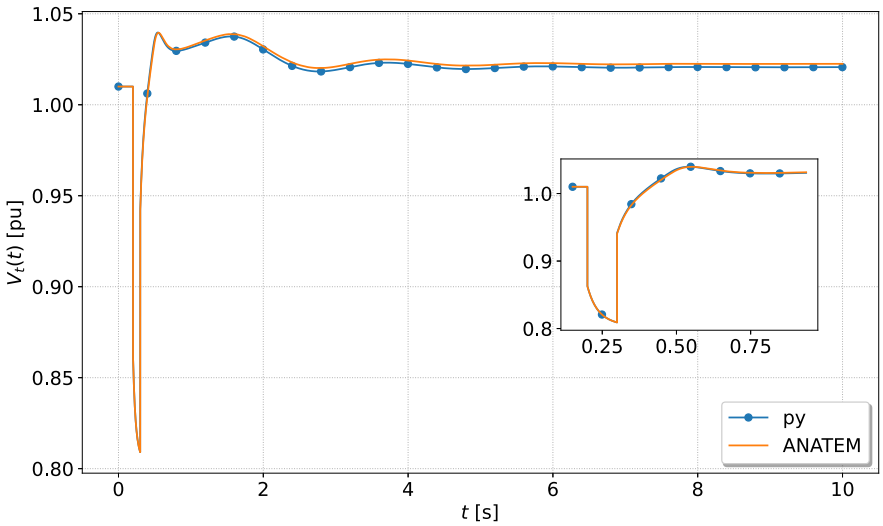
Fonte: Elaborada pelo autor (2024)

Figura 27 – Tensão terminal do gerador G_3 do sistema de transmissão de 11 barras.



Fonte: Elaborada pelo autor (2024)

Figura 28 – Tensão terminal do gerador G_4 do sistema de transmissão de 11 barras.



Fonte: Elaborada pelo autor (2024)

Tabela 4 – Valores de NIAE dos geradores do sistema de transmissão de 11 barras.

Gerador	δ_{g3}	ω	P_e	I_a
G_1	0,9989	0,9998	0,9996	0,9984
G_2	0,9979	0,9998	0,9996	0,9996
G_3	-	0,9998	0,9996	0,9991
G_4	0,9907	0,9998	0,9996	0,9984

Fonte: Elaborada pelo autor (2024)

do ANAREDE e do ANATEM. Para isso, adota-se um coeficiente chamado *Normalized Integral Absolute Error* (NIAE), ou Integral do Erro Absoluto Normalizado, calculado através da equação (5.1), onde $\mathbf{x}(\tau)$ corresponde aos resultados que se deseja comparar e $\mathbf{x}_{ref}(\tau)$ corresponde aos resultados utilizados como referência para a comparação.

$$NIAE = 1 - \frac{\int_0^t |\mathbf{x}(\tau) - \mathbf{x}_{ref}(\tau)|\, d\tau}{\int_0^t |\mathbf{x}_{ref}(\tau)|\, d\tau} \tag{5.1}$$

Neste trabalho, as integrais da equação (5.1) são calculadas através da regra dos trapézios. Quando a relação $NIAE \geq 0{,}95$ é atendida, significa que os resultados obtidos de $\mathbf{x}(\tau)$ podem ser considerados adequados.

Depois de realizar os cálculos correspondentes, tem-se na Tabela 4 os valores de NIAE dos geradores, enquanto os valores de NIAE das tensões das barras da rede estão apresentados na Tabela 5.

- Na Tabela 4, todos os coeficientes dos geradores estão acima de 0,9980, exceto o

Tabela 5 – Valores de NIAE das tensões nodais do sistema de transmissão de 11 barras.

Barra	V_t
1	0,9984
2	0,9983
3	0,9984
4	0,9984
5	0,9985
6	0,9988
7	0,9992
8	0,9970
9	0,9994
10	0,9994
11	0,9987

Fonte: Elaborada pelo autor (2024)

coeficiente de δ_{43}, que é igual a 0,9907.

- Na Tabela 5, todos os coeficientes das tensões estão acima de 0,9980, exceto o coeficiente da barra 8, que é igual a 0,9970. Essa barra, por sua vez, é o elemento da rede no qual se aplica o curto-circuito e, portanto, é o elemento submetido à maior variação de tensão em relação às outras barras da rede.

Uma vez que todos os valores da Tabela 4 e da Tabela 5 são maiores do que 0,95, tem-se que a estratégia de simulação eletromecânica com a interface FMI apresentada neste trabalho pode ser considerada adequada.

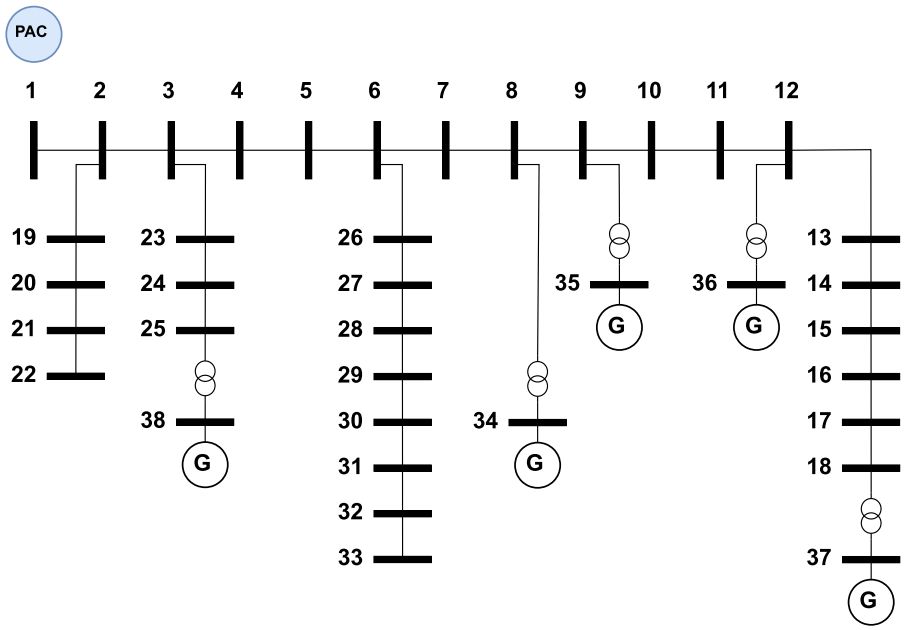
5.2 SIMULAÇÃO NO TEMPO COM UM SISTEMA DE DISTRIBUIÇÃO

Agora que está validada a aplicação do método alternado acoplado à FMU do tipo *Model Exchange*, esta seção apresenta a mesma simulação no tempo do sistema de transmissão de 11 barras, porém agora considerando também o acoplamento de um sistema de distribuição através do módulo `OpenDSSDirect`, baseado no programa OpenDSS.

Neste trabalho, o sistema de distribuição escolhido é o de 38 barras, que também é representado por seu equivalente de sequência positiva. Esse sistema pode ser visto na Figura 29 e no Apêndice D (SINGH; MISRA; SINGH, 2007).

Considerando a barra 1 da Figura 29 como o PAC da distribuição, a sua conexão com a rede de transmissão de 11 barras é realizada através de um transformador, cuja impedância adotada neste trabalho é igual a $\overline{Z}_{tr} = j0{,}001$ %, nas bases 100 [MVA] e 69/230 [kV].

Figura 29 – Sistema de distribuição de 38 barras.



Fonte: Elaborada pelo autor (2023)

5.2.1 Regime permanente com OpenDSS

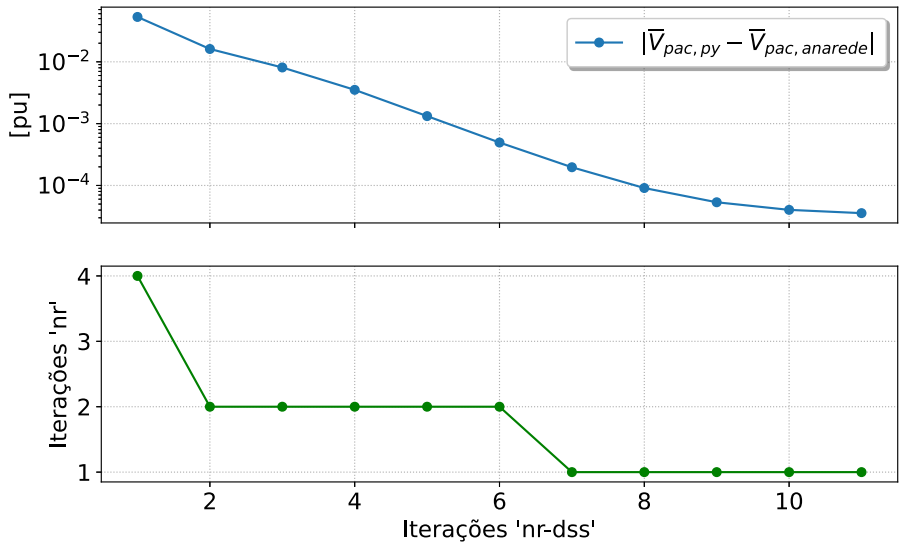
Sem o subsistema de distribuição, o regime permanente é calculado apenas pelo método iterativo de *Newton-Raphson*. No entanto, com a distribuição acoplada ao restante da simulação, essa solução é encontrada através do conjunto nr-dss, isto é, através da alternância entre as soluções de *Newton-Raphson* e do `OpenDSSDirect`, conforme descrito na Subseção **4.3.2**.

Com esse novo método de solução de regime permanente, adotam-se uma tolerância absoluta de 10^{-6} [pu] relativa de 10^{-4} % para as potências ativas e reativas líquidas das barras da transmissão (no caso da solução individual de *Newton-Raphson*) e para a tensão do PAC da distribuição (no caso da solução do conjunto nr-dss). Além disso, os parâmetros adotados para as cargas e geradores da transmissão e da distribuição são: modelagem do tipo potência constante para as cargas e modelagem do tipo PV para os geradores, exceto o gerador G_3 do sistema de 11 barras, cuja modelagem adotada é do tipo *slack*.

Considerando o sistema de distribuição de 38 barras como parte da carga $L_7 = 967 + j100$ [MVA] do sistema de transmissão de 11 barras, a distribuição consome 199 [MW] e entrega 95,7 [Mvar] para a transmissão durante o regime permanente. O processo de convergência dessa solução está mostrado nos gráficos da Figura 30.

O primeiro gráfico da Figura 30 (em azul) apresenta a diferença entre $\overline{V}_{pac,py}$ e $\overline{V}_{pac,anarede}$, onde $\overline{V}_{pac,py}$ é a tensão do PAC da distribuição obtida pelo *Python* a cada iteração "nr-dss", enquanto $\overline{V}_{pac,anarede}$ é a solução convergente do ANAREDE. O segundo gráfico da Figura 30 (em verde) apresenta as iterações de *Newton-Raphson* que são

Figura 30 – Processo de convergência de regime permanente do sistema de transmissão de 11 barras acoplado ao sistema de distribuição de 38 barras.



Fonte: Elaborada pelo autor (2024)

necessárias a cada iteração “nr-dss”.

Sabendo disso, o processo de convergência ocorre da seguinte forma:

- Na 1ª iteração “nr-dss”, considera-se o valor de tensão do PAC da distribuição igual a $\overline{V}_{pac,dss} = 1{,}0\angle 0°$ [pu] e são necessárias 4 iterações “nr”;
- Na 2ª iteração “nr-dss”, considera-se $\overline{V}_{pac,dss} = 0{,}9750\angle 1{,}64°$ [pu] e são necessárias 2 iterações “nr”, e assim por diante. Os valores das próximas iterações estão apresentados na Tabela 6.
- Por fim, na 11ª iteração “nr-dss”, a transmissão e a distribuição convergem para uma tensão igual a $\overline{V}_{pac,dss} = 0{,}9610\angle 2{,}11°$ [pu], sendo necessária apenas 1 iteração “nr”. Conforme esperado, há uma redução gradual do número de iterações “nr” à medida que as soluções do *Python* se aproximam da solução convergente do ANAREDE.

Além dos valores obtidos de $\overline{V}_{pac,dss}$, outros resultados de regime permanente do sistema de distribuição de 38 barras estão apresentados na Tabela 7, onde os resultados do PAC da distribuição estão destacados em vermelho.

Conforme esperado, o PAC da transmissão (barra 7 da Tabela 3) possui tensão suficientemente igual à tensão do PAC da distribuição (barra 1 da Tabela 7), haja vista que a impedância adotada para o transformador de conexão ($\overline{Z}_{tr} = j0{,}001$ %) é consideravelmente menor em relação às outras impedâncias das linhas do sistema.

Tabela 6 – Processo de convergência de $\overline{V}_{pac,dss}$ ao longo das iterações do conjunto "nr-dss".

Iteração "nr-dss"	$V_{pac,dss}$ [pu]	$\angle\overline{V}_{pac,dss}$ [°]	**Iterações "nr"**
1	1,0000	0,00	4
2	0,9750	1,64	2
3	0,9668	1,78	2
4	0,9630	1,94	2
5	0,9617	2,05	2
6	0,9613	2,09	2
7	0,9611	2,10	1
8	0,9611	2,11	1
9	0,9610	2,11	1
10	0,9610	2,11	1
11	0,9610	2,11	1

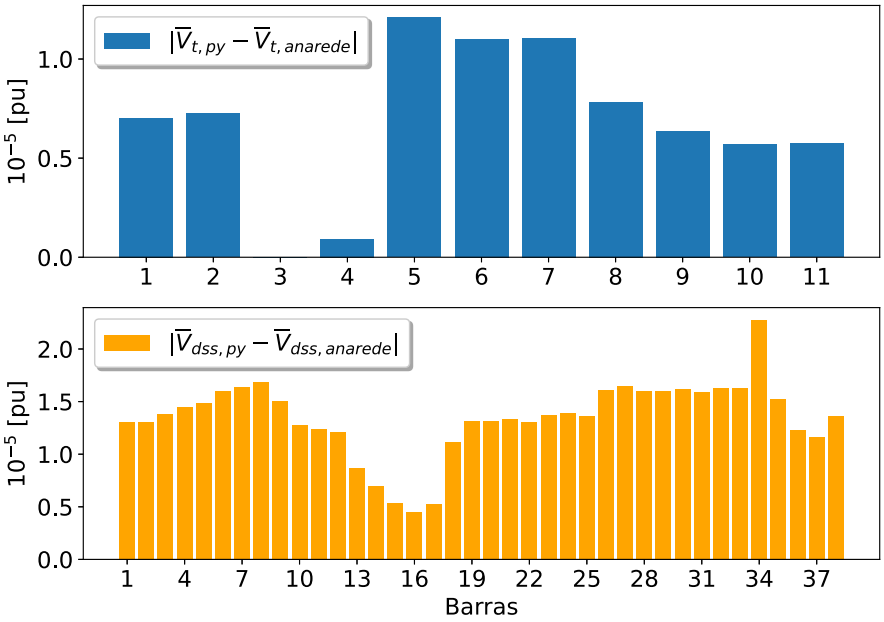
Fonte: Elaborada pelo autor (2024)

Tabela 7 – Resultados de regime permanente do sistema de distribuição de 38 barras.

Barra	**V [pu]**	**Ângulo [°]**	**Barra**	**V [pu]**	**Ângulo [°]**
1	0,9610	2,11	20	0,9559	1,96
2	0,9601	2,04	21	0,9551	1,94
3	0,9572	1,67	22	0,9545	1,92
4	0,9574	1,43	23	0,9546	1,60
5	0,9580	1,17	24	0,9500	1,44
6	0,9626	0,58	25	0,9488	1,33
7	0,9733	0,54	26	0,9607	0,61
8	0,9809	-0,09	27	0,9582	0,67
9	0,9882	-0,54	28	0,9469	0,75
10	0,9891	-0,76	29	0,9388	0,82
11	0,9892	-0,80	30	0,9353	0,93
12	0,9894	-0,87	31	0,9312	0,84
13	0,9896	-1,06	32	0,9303	0,82
14	0,9904	-1,14	33	0,9301	0,82
15	0,9917	-1,21	34	1,0100	-0,52
16	0,9932	-1,29	35	1,0049	-0,87
17	0,9985	-1,38	36	0,9950	-0,98
18	1,0008	-1,44	37	1,0100	-1,55
19	0,9596	2,03	38	0,9546	1,21

Fonte: Elaborada pelo autor (2024)

Figura 31 – Comparações das tensões das barras dos sistemas de transmissão e distribuição com os resultados do ANAREDE.



Fonte: Elaborada pelo autor (2024)

Além da validação da tensão nodal do PAC, a validação das outras tensões das redes de transmissão e distribuição obtidas pelo *Python* é realizada através da comparação com o ANAREDE, conforme a Figura 31. O gráfico em azul apresenta os erros das barras da transmissão, enquanto o gráfico em laranja apresenta os erros das barras da distribuição.

Com base na Figura 31, os erros das tensões da transmissão apresentam um valor máximo de $1{,}2 \cdot 10^{-5}$ [pu] e uma média de $6{,}3 \cdot 10^{-6}$ [pu], enquanto os erros das tensões da distribuição apresentam um valor máximo de $2{,}3 \cdot 10^{-5}$ [pu] e uma média de $1{,}2 \cdot 10^{-5}$ [pu]. Portanto, pode-se considerar adequada a solução do regime permanente do sistema de transmissão de 11 barras através do protótipo em *Python*. Vale ressaltar que a modelagem do tipo *slack* do gerador da barra 3 torna o seu erro de tensão igual a zero.

Em comparação com os resultados do regime permanente sem o OpenDSS (Subseção **5.1.1**), a inserção da rede de distribuição acarreta erros pouco maiores para as barras da transmissão. Isto é, enquanto a simulação sem a distribuição acarreta erros de tensão na ordem de 10^{-6} [pu], o conjunto "nr-dss" acarreta erros na ordem de 10^{-5} [pu].

Além disso, mesmo com a modelagem do tipo PV dos geradores durante o regime permanente, a presença do OpenDSS faz com que os erros das tensões das barras 1 e 2 da transmissão se tornem maiores em relação aos erros de outras barras da transmissão, ao contrário da simulação sem OpenDSS. Isto é, mesmo as magnitudes das tensões terminais dos geradores sendo as mesmas da Subseção **5.1.1**, as fases encontradas pelo conjunto

Tabela 8 – Valores de $\overline{y}_{dss}$ obtidos por diferentes métodos de sensibilidade.

Método	**Iterações**	$\overline{y}_{dss}$ [pu]	$\overline{I}_{0,pac,dss}/\overline{V}_{0,pac,dss}$ [pu]
Positivo	12	-5,73-j44,04	2,15+j1,04
Negativo	11	-5,84-j43,44	2,15+j1,04
Central	11	-5,79-j43,74	2,15+j1,04

Fonte: Elaborada pelo autor (2024)

"nr-dss" aumentam os erros absolutos.

Por fim, vale ressaltar que os resultados mostrados nesta seção correspondem a uma simulação na qual a admitância de Norton do OpenDSS ($\overline{y}_{dss}$) é obtida através do método de sensibilidade central, explicado na Subseção **4.3.1**. No entanto, para fins de comparação, tem-se na Tabela 8 os valores de $\overline{y}_{dss}$ obtidos pelos métodos de sensibilidade positiva, negativa e central. Além desses valores, apresentam-se também a quantidade necessária de iterações "nr-dss" e a relação $\overline{I}_{0,pac,dss}/\overline{V}_{0,pac,dss}$ de cada método.

Comparando os métodos da Tabela 8, é possível observar que as iterações estão em torno de 11 a 12, convergindo para valores próximos de $\overline{y}_{dss}$ e valores idênticos de $\overline{I}_{0,pac,dss}/\overline{V}_{0,pac,dss}$. Vale mencionar também que, para cada iteração "nr-dss", o método da sensibilidade central demanda duas soluções do OpenDSS, enquanto o positivo e o negativo demandam apenas uma.

Portanto, sendo $\overline{y}_{0,dss}$ a admitância de Norton do sistema de distribuição que deve ser mantida constante durante o regime transitório, tem-se na equação (5.2) o valor a ser adicionado à matriz de admitâncias nodais da rede de transmissão.

$$\overline{y}_{0,dss} = 2{,}15 + j1{,}04 \text{ [pu]} \tag{5.2}$$

5.2.2 Regime transitório com OpenDSS

Para a simulação no tempo do regime transitório do sistema com a presença de uma rede de distribuição, realiza-se novamente a simulação no tempo de 10 [s] da Subseção **5.1.2**, na qual um curto-circuito de impedância $\overline{Z}_{flt} = j0{,}001$ [pu] na base 100 [MVA] é aplicado à barra 8 da rede de transmissão no instante $t_{flt} = 0{,}2$ [s] e removido 100 [ms] depois, junto com a linha defeituosa entre as barras 8 e 9.

Para a modelagem das cargas da rede de transmissão de 11 barras, a carga L_9 é modelada como impedância constante, enquanto a carga L_7 é modelada parcialmente como impedância constante e como o sistema de distribuição de 38 barras. Como o sistema de distribuição consome $L_{dss} = 199 - j95{,}7$ [MVA] da carga $L_7 = 967 + j100$ [MVA] durante o regime permanente, modela-se a parcela de $L_7 - L_{dss} = 768 + j195{,}7$ [MVA] como impedância constante durante o regime transitório, enquanto a parcela L_{dss} se mantém

sendo modelada pelo sistema de 38 barras.

Os outros parâmetros adotados para a simulação no tempo são: passo de integração igual a 1 [ms]; modelagem dos geradores da transmissão através da FMU do tipo *Model Exchange*; tolerância absoluta de 10^{-4} [pu] e relativa de 10^{-2} % para as tensões nodais da rede de transmissão (no caso da solução do conjunto modelo-rede-dss) e para a tensão nodal do PAC da distribuição (no caso da solução do conjunto rede-dss).

Após a execução da simulação no tempo com um sistema de distribuição, são gerados os seguintes gráficos: número de iterações, ângulos internos relativos e tensões das barras dos geradores. Em seguida, mostra-se também o gráfico de tensão do PAC e realiza-se uma comparação dos resultados obtidos pelo *Python* em relação ao programa ANATEM.

Durante a simulação no tempo com a rede de distribuição, são contabilizados dois tipos de iterações: as iterações do conjunto "modelo-rede-dss" e do conjunto "rede-dss", sendo que o segundo tipo está contido dentro do primeiro. Os gráficos correspondentes estão apresentados na Figura 32.

O gráfico de iterações do conjunto "modelo-rede-dss" da Figura 32(a) é idêntico ao gráfico do conjunto "modelo-rede" da Subseção **5.1.2**. No entanto, o gráfico do conjunto "rede-dss" da Figura 32(b) indica que a presença da rede de distribuição de 38 barras acarreta uma maior quantidade de iterações, o que se justifica pela alternância entre a solução da distribuição pelo `OpenDSSDirect` e a solução da transmissão pelo método nodal.

Agora, apresentam-se graficamente os resultados dos geradores do sistema de transmissão de 11 barras com a rede de 38 barras. Os ângulos internos relativos estão mostrados na Figura 33 e as tensões terminais estão mostradas da Figura 34 até a Figura 37. É possível observar a proximidade gráfica entre os resultados do *Python* e os do ANATEM.

Agora, mostra-se também a tensão do PAC da rede de distribuição de 38 barras, conforme a Figura 38. Como esperado, é possível observar a proximidade gráfica entre os resultados do *Python* e do ANATEM.

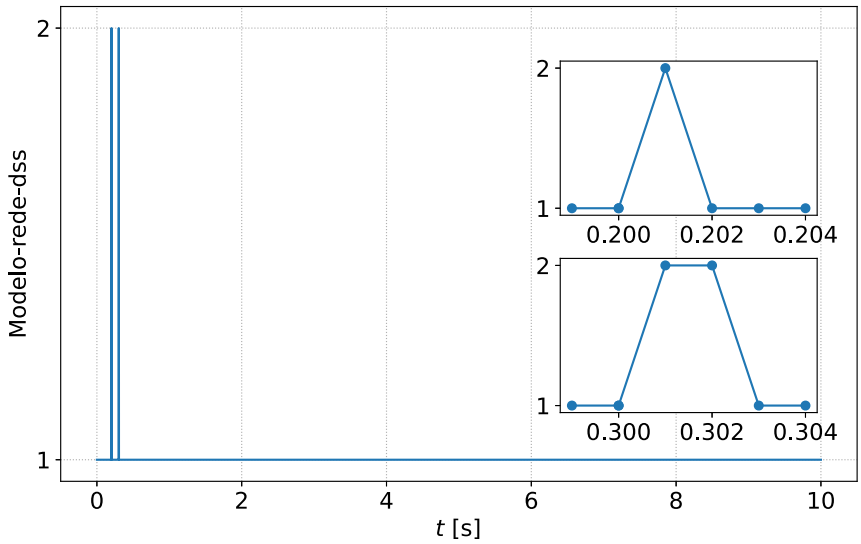
5.2.3 Comparação quantitativa dos resultados com OpenDSS

No entanto, apesar da proximidade gráfica, é necessário realizar uma verificação quantitativa da precisão dos resultados do protótipo em *Python* em relação aos resultados do ANAREDE e do ANATEM. Para isso, adota-se novamente o NIAE, mostrado anteriormente na equação (5.1).

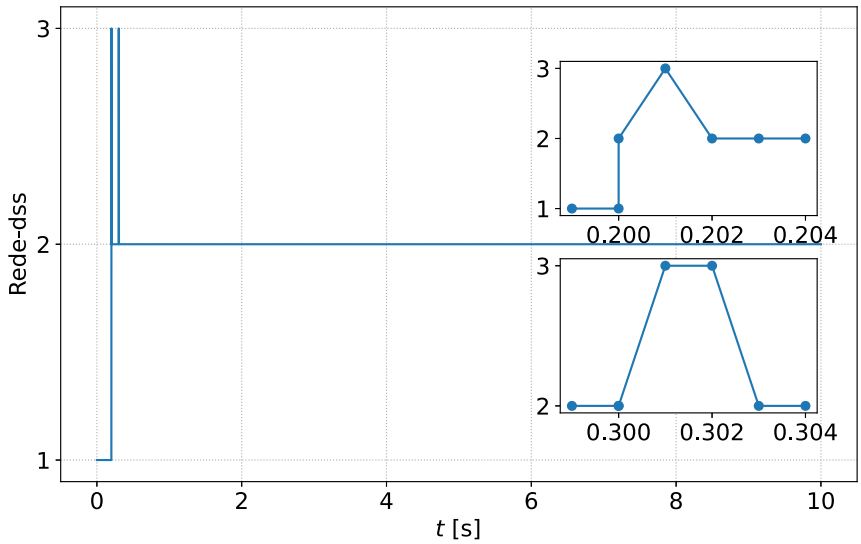
Depois de realizar os cálculos correspondentes, tem-se na Tabela 9 os valores de NIAE dos geradores, enquanto os valores de NIAE das tensões das barras da transmissão e de distribuição estão apresentados na Tabela 10 e na Tabela 11, respectivamente.

- Em comparação com os resultados do regime transitório sem o OpenDSS (Subse-

Figura 32 – Número de iterações da simulação no tempo com OpenDSS.



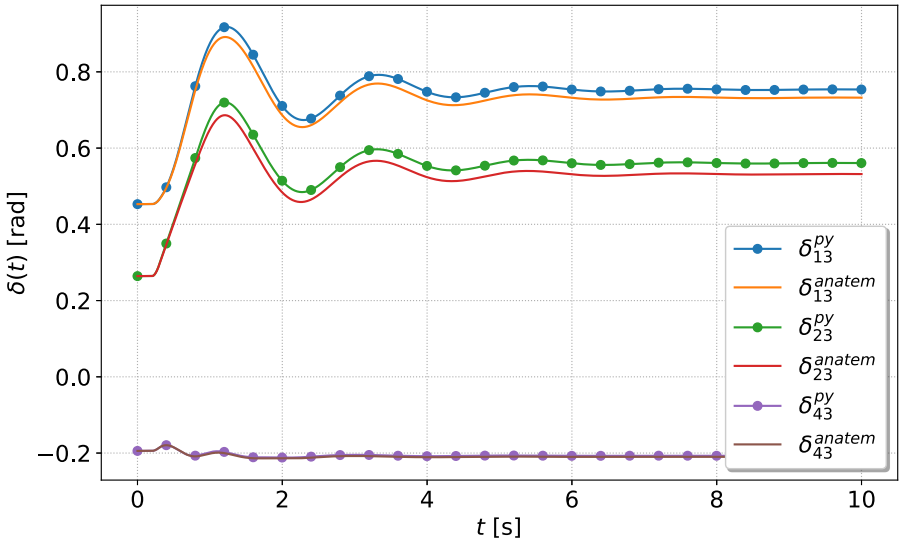
(a) Número de iterações “modelo-rede-dss”.



(b) Número de iterações “rede-dss”.

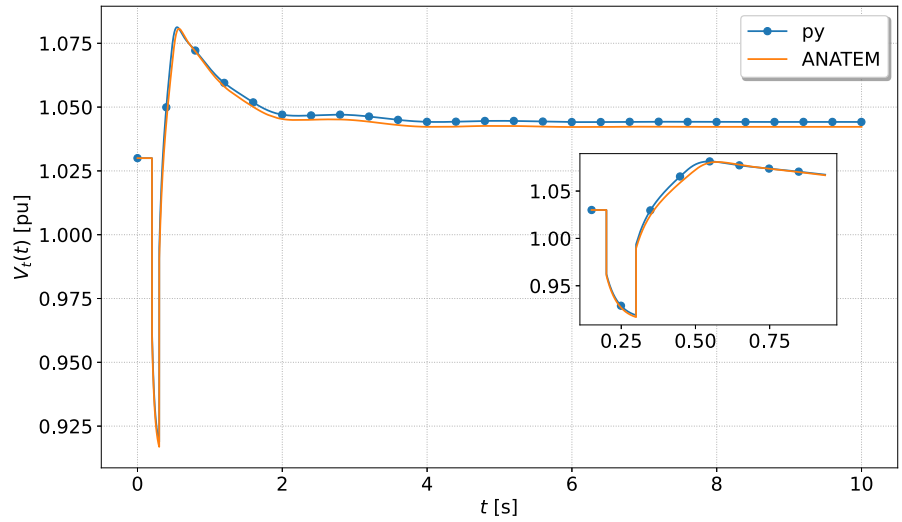
Fonte: Elaborada pelo autor (2024)

Figura 33 – Ângulos internos relativos dos geradores do sistema de transmissão de 11 barras com OpenDSS.



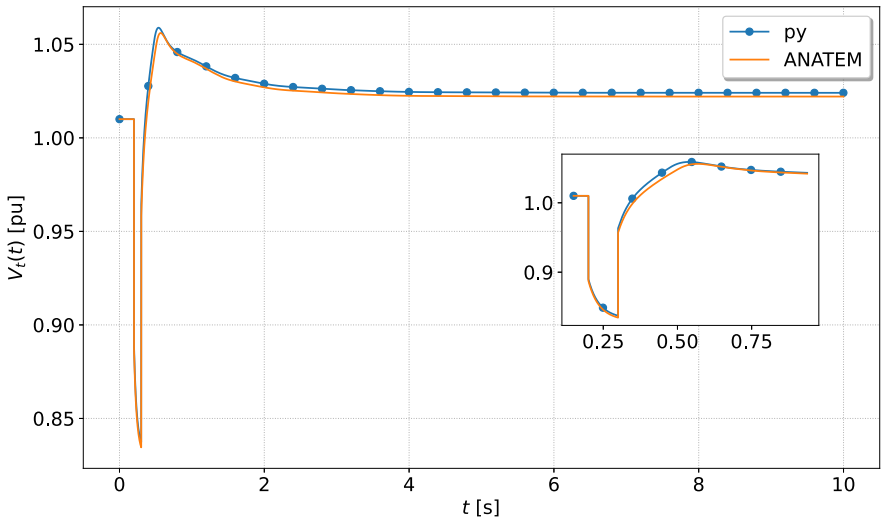
Fonte: Elaborada pelo autor (2024)

Figura 34 – Tensão terminal do gerador G_1 do sistema de transmissão de 11 barras com OpenDSS.



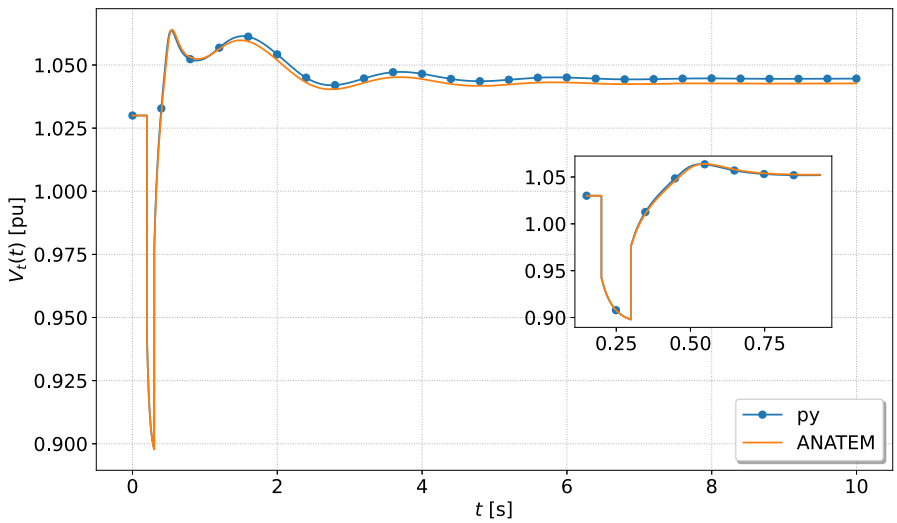
Fonte: Elaborada pelo autor (2024)

Figura 35 – Tensão terminal do gerador G_2 do sistema de transmissão de 11 barras com OpenDSS.



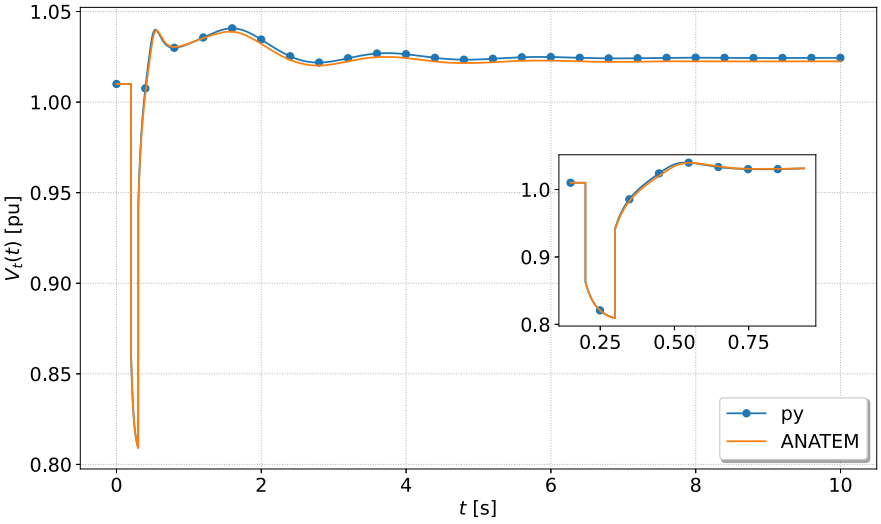
Fonte: Elaborada pelo autor (2024)

Figura 36 – Tensão terminal do gerador G_3 do sistema de transmissão de 11 barras com OpenDSS.



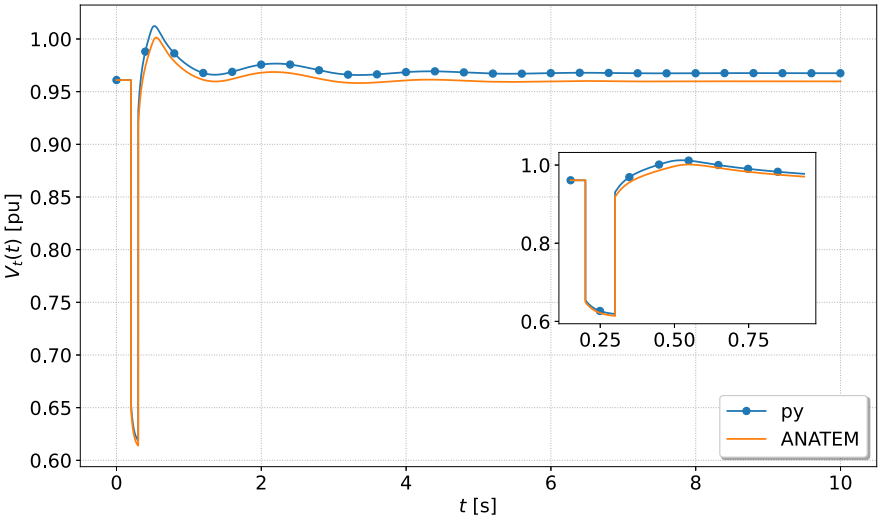
Fonte: Elaborada pelo autor (2024)

Figura 37 – Tensão terminal do gerador G_4 do sistema de transmissão de 11 barras com OpenDSS.



Fonte: Elaborada pelo autor (2024)

Figura 38 – Tensão nodal do PAC da rede de distribuição de 38 barras.



Fonte: Elaborada pelo autor (2024)

Tabela 9 – Valores de NIAE dos geradores do sistema de transmissão de 11 barras com OpenDSS.

Gerador	δ_{g3}	ω	P_e	I_a
G_1	0,9721	0,9998	0,9994	0,9947
G_2	0,9484	0,9998	0,9993	0,9886
G_3	-	0,9998	0,9992	0,9970
G_4	0,9876	0,9998	0,9992	0,9954

Fonte: Elaborada pelo autor (2024)

Tabela 10 – Valores de NIAE das tensões nodais do sistema de transmissão de 11 barras com OpenDSS.

Barra	V_t
1	0,9982
2	0,9980
3	0,9983
4	0,9983
5	0,9968
6	0,9947
7	0,9920
8	0,9931
9	0,9965
10	0,9972
11	0,9979

Fonte: Elaborada pelo autor (2024)

ção **5.2.2**), a inserção da rede de distribuição acarreta uma redução para os valores de NIAE dos geradores, principalmente os ângulos internos relativos. O exemplo mais evidente desse cenário na Tabela 9 é o coeficiente de δ_{23}, que é o único que se encontra abaixo do valor de referência de 0,95.

- Analisando a Tabela 10, tem-se que a presença do OpenDSS também acarreta uma redução para os coeficientes das tensões nodais da transmissão. Porém, apesar dessa redução, os coeficientes se mantêm acima de 0,9900. Além disso, o menor coeficiente da tabela é o da barra 7 (destacado em vermelho), que se trata do PAC da rede de transmissão e, portanto, é diretamente submetida aos resultados encontrados pelo `OpenDSSDirect` ao longo da simulação. Sabendo disso, observa-se que os coeficientes das tensões da transmissão seguem um padrão no qual as barras mais próximas ao PAC possuem valores menores em relação a barras mais distantes.

Tabela 11 – Valores de NIAE das tensões nodais do sistema de distribuição de 38 barras.

Barra	V_{dss}	**Barra**	V_{dss}	**Barra**	V_{dss}
1	0,9920	14	0,9861	27	0,9868
2	0,9917	15	0,9860	28	0,9853
3	0,9905	16	0,9860	29	0,9843
4	0,9898	17	0,9858	30	0,9838
5	0,9891	18	0,9858	31	0,9832
6	0,9874	19	0,9917	32	0,9831
7	0,9871	20	0,9914	33	0,9830
8	0,9868	21	0,9913	34	0,9862
9	0,9866	22	0,9912	35	0,9864
10	0,9864	23	0,9902	36	0,9864
11	0,9864	24	0,9897	37	0,9856
12	0,9864	25	0,9896	38	0,9901
13	0,9862	26	0,9871		

Fonte: Elaborada pelo autor (2024)

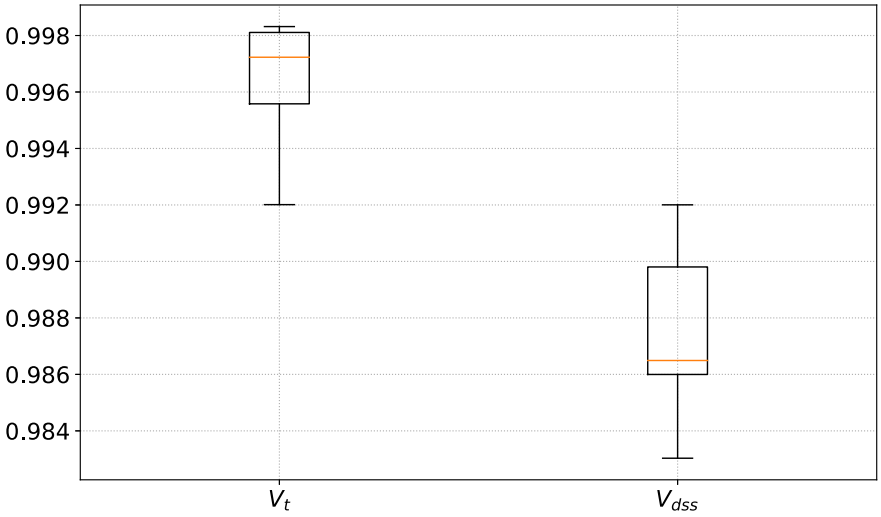
- Analisando a Tabela 11, o maior coeficiente é 0,9920, correspondente ao PAC da distribuição (barra 1). Conforme esperado, esse valor é igual ao NIAE do PAC da transmissão (barra 7 da Tabela 10), haja vista que a impedância adotada para o transformador de conexão ($\overline{Z}_{tr} = j0{,}001$ %) é consideravelmente menor em relação às outras impedâncias das linhas do sistema. Além disso, observa-se que os coeficientes das tensões da distribuição seguem um padrão no qual as barras mais próximas ao PAC possuem valores maiores em relação a barras mais distantes. Sabendo disso, tem-se que o menor valor encontrado de NIAE é 0,9830, correspondente à barra 33, que é a última barra do ramo no qual se encontra.

Com os valores de NIAE das tensões das barras das redes de transmissão (Tabela 10) e distribuição (Tabela 11), apresenta-se na Figura 39 o diagrama de caixa da dispersão desses coeficientes, onde se mostra que a mediana das tensões da transmissão ($\overline{V}_t$) é igual a 0,9972, enquanto a mediana das tensões da distribuição ($\overline{V}_{dss}$) é igual a 0,9865.

Além das tensões da distribuição, calcula-se também os valores de NIAE de outros parâmetros da rede de distribuição de 38 barras. A Tabela 12 apresenta os coeficientes para as cargas, enquanto a Figura 40 apresenta os diagramas de caixa correspondentes.

Com base na Tabela 12 e na Figura 40, os valores de NIAE de $P_{l,dss}$ e $Q_{l,dss}$ são semelhantes entre si, apresentando uma mediana igual a 0,9391. Além disso, dentre as 32 cargas da distribuição, 19 delas possuem coeficientes abaixo de 0,95, isto é, mais da metade das cargas possuem NIAE abaixo do valor de referência. Também é possível observar

Figura 39 – Diagrama de caixa dos valores de NIAE das tensões nodais do sistema de transmissão de 11 barras com OpenDSS.



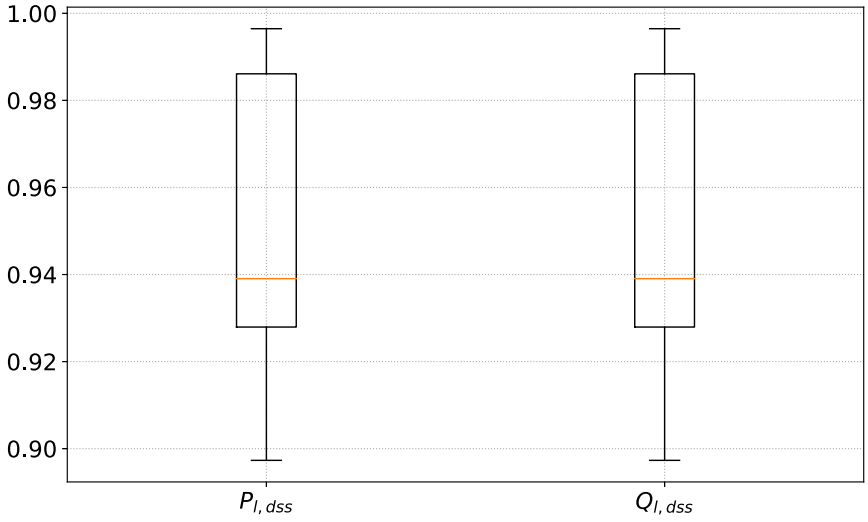
Fonte: Elaborada pelo autor (2024)

Tabela 12 – Valores de NIAE das cargas do sistema de distribuição de 38 barras.

Carga	$P_{l,dss}$	$Q_{l,dss}$	**Carga**	$P_{l,dss}$	$Q_{l,dss}$
2	0,9387	0,9387	18	0,9696	0,9696
3	0,9354	0,9354	19	0,9377	0,9377
4	0,9370	0,9370	20	0,9313	0,9313
5	0,9394	0,9394	21	0,9300	0,9300
6	0,9516	0,9516	22	0,9289	0,9289
7	0,9731	0,9731	23	0,9308	0,9308
8	0,9885	0,9885	24	0,9230	0,9230
9	0,9965	0,9965	25	0,9210	0,9210
10	0,9944	0,9944	26	0,9484	0,9465
11	0,9943	0,9943	27	0,9441	0,9422
12	0,9938	0,9952	28	0,9252	0,9252
13	0,9931	0,9945	29	0,9117	0,9117
14	0,9912	0,9912	30	0,9059	0,9059
15	0,9886	0,9886	31	0,8992	0,8992
16	0,9853	0,9853	32	0,8978	0,8978
17	0,9745	0,9745	33	0,8973	0,8973

Fonte: Elaborada pelo autor (2024)

Figura 40 – Diagramas de caixa dos valores de NIAE das cargas do sistema de distribuição de 38 barras.



Fonte: Elaborada pelo autor (2024)

Tabela 13 – Valores de NIAE dos geradores do sistema de distribuição de 38 barras.

Gerador	$P_{g,dss}$	$Q_{g,dss}$
34	0,9519	0,9516
35	0,9627	0,9627
36	0,9828	0,9828
37	0,9507	0,9515
38	0,9311	0,9311

Fonte: Elaborada pelo autor (2024)

um padrão no qual as barras mais radiais possuem coeficientes menores em relação às menos radiais. Sabendo disso, tem-se que o menor valor encontrado de NIAE é 0,8973, correspondente à barra 33, que é a última barra do ramo no qual se encontra. Esse cenário é semelhante ao observado na Tabela 11 mostrada anteriormente.

Agora, tem-se na Tabela 13 os coeficientes para os geradores da rede de distribuição. Os valores de NIAE de $P_{g,dss}$ e $Q_{g,dss}$ são semelhantes entre si, apresentando uma média de 0,9558. Além disso, dentre os 5 geradores da distribuição, apenas 1 deles (gerador 38) possui coeficientes abaixo de 0,95.

Analisando os resultados de regime transitório de forma geral, tem-se que todos os valores de NIAE da transmissão se mantém maiores do que o valor de referência de 0,95, com exceção de um único coeficiente (correspondente ao ângulo interno δ_{23}). Entretanto,

o mesmo não ocorre com a distribuição, porque embora os valores de NIAE de $\overline{V}_{dss}$ sejam todos acima de 0,9830, há coeficientes de $P_{l,dss}$ e $Q_{l,dss}$ em torno de 0,90.

Portanto, apesar dos resultados menos precisos em relação à simulação da Seção 5.1, tem-se que o acoplamento do sistema de distribuição à estratégia de simulação eletromecânica com a interface FMI pode ser considerada adequada.

5.3 CONCLUSÕES PARCIAIS

Neste capítulo, o método alternado apresentado nos capítulos anteriores foi implementado em ambiente *Python* para a simulação de um sistema de transmissão de 11 barras, cujas unidades geradoras foram modeladas através de uma FMU do tipo *Model Exchange*. Esse mesmo sistema elétrico foi montado e simulado nos programas ANAREDE e ANATEM, com o objetivo de comparar os resultados obtidos pelo protótipo em *Python*.

Foram mostradas duas simulações: a primeira sem e a segunda com a presença de uma rede de distribuição de 38 barras acoplada ao restante do sistema. Essas duas simulações realizaram comparações quantitativas dos resultados tanto de regime permanente (através dos erros absolutos) quanto de regime transitório (através do coeficiente NIAE).

Na primeira simulação (isto é, sem a rede de 38 barras), os valores de NIAE indicaram que a solução alternada e as soluções individuais dos subsistemas foram bem sucedidas. Portanto, foi possível considerar como adequada a estratégia de simulação eletromecânica com a interface FMI apresentada neste trabalho.

Na segunda simulação (isto é, com a rede de 38 barras), o cálculo do regime permanente passou a envolver uma alternância entre a solução da transmissão e da distribuição, mas os resultados sofreram pouca alteração em relação ao regime permanente da primeira simulação. No entanto, os resultados de regime transitório passaram a exigir mais iterações para a convergência e indicaram diferenças maiores entre os resultados do *Python* e do ANATEM. Embora sejam diferenças que não comprometam a validação da simulação, são indicativos de que o acoplamento de um sistema de distribuição à estratégia de simulação eletromecânica com a interface FMI pode ser melhorado futuramente.

6 CONCLUSÕES E DESENVOLVIMENTOS FUTUROS

Este trabalho propôs uma estratégia para a simulação de sistemas de potência visando a estabilidade transitória através de três ferramentas computacionais distintas: linguagem *Python* (rede de transmissão), interface FMI 2.0.3 do tipo *Model Exchange* (máquinas e equipamentos dinâmicos) e um módulo baseado no programa OpenDSS (rede de distribuição).

Antes de adotar a FMU do tipo *Model Exchange* para o modelo de máquinas elétricas, foi testada também a do tipo *Co-Simulation*, que compacta num único arquivo `.fmu` o seu solucionador numérico próprio em conjunto com o modelo a ser solucionado. No entanto, os resultados desse teste não foram satisfatórios, pois como a troca de dados da FMU do tipo CS é restrita a pontos de comunicação discretos, houve um atraso dos dados intercambiados e, consequentemente, uma defasagem dos resultados obtidos.

Para eliminar esse atraso, este trabalho adotou a FMU do tipo ME, o que acarretou a necessidade de criar um solucionador externo ao arquivo compactado. Com a propriedade de dividir a solução em diferentes modos (modo de inicialização, modo de tempo contínuo e modo de evento), foi possível realizar as soluções do modelo de máquinas de forma adequada.

Escolhido o tipo de FMU, o protótipo de simulação no tempo em *Python* foi criado suportando apenas a rede de transmissão e o modelo de máquinas. Com o acoplamento desses dois subsistemas, foram discutidas soluções consolidadas da área de engenharia que demonstraram potencial de trabalharem em conjunto numa simulação computacional. Para o regime permanente, as equações nodais de rede elétrica foram solucionadas por *Newton-Raphson*, enquanto o regime transitório foi simulado pela alternância entre duas soluções distintas: a primeira foi a solução iterativa das equações nodais da rede de transmissão, enquanto a segunda foi a solução das equações das FMUs através do método trapezoidal.

Dentro do código em *Python*, houve tentativas de melhorar o desempenho do método alternado. Por exemplo, em vez de utilizar as tensões nodais da rede de transmissão como critério de convergência, foram testadas as correntes de armadura e as correntes de Norton dos geradores. Com o protótipo alterado, o programa calculou os mesmos resultados, mas demandou mais iterações do que o protótipo original. Portanto, o critério de convergência do método alternado foi mantido como sendo apenas as tensões nodais da rede de transmissão.

Em seguida, foram discutidas algumas adaptações do protótipo de simulação do sistema elétrico com o objetivo de possibilitar o acoplamento de redes de distribuição, o que exigiu a extensão e criação de funções em *Python*. Sabendo disso, testes foram bem sucedidos somente em casos nos quais a distribuição era representada pelo seu equivalente

de Norton, que por sua vez possui a vantagem de não exigir um aumento das dimensões da matriz de admitâncias nodais da rede de transmissão.

O cálculo de regime permanente foi adaptado para um método alternado entre a solução da rede de transmissão (por *Newton-Raphson*) e a da rede de distribuição (pelo módulo `OpenDSSDirect`). Sabendo disso, foram discutidos métodos de sensibilidade para calcular a admitância de Norton da rede de distribuição. Esses métodos, por sua vez, se mostraram eficientes para acelerar a convergência de regime permanente. Para o cálculo de regime transitório, as soluções da transmissão continuaram sendo realizadas considerando a distribuição como um equivalente de Norton, cuja admitância em paralelo se manteve constante e cuja corrente era ajustada a cada iteração, da mesma forma que as cargas do modelo ZIP também precisaram ser ajustadas diante de eventos.

Por fim, o método alternado implícito foi implementado para a simulação no tempo de um sistema de transmissão contendo 11 barras e 4 unidades geradoras. Os resultados dessa primeira simulação foram satisfatoriamente próximos aos obtidos pelos programas ANAREDE e ANATEM. Para a segunda simulação no tempo, foi discutida a influência da rede de distribuição assumindo parte de uma carga de impedância constante. Foi possível observar que, apesar do aumento do número de iterações, do esforço computacional e da redução da precisão dos resultados, o acoplamento da distribuição ao restante do protótipo através da representação de Norton se provou bem sucedida.

Portanto, as metodologias apresentadas nesta dissertação atenderam aos objetivos desejados, isto é, a divisão de um sistema maior em subsistemas menores e de fácil solução se mostrou uma ferramenta com potencial de representação de sistemas de grande porte. Com isso, deseja-se que este trabalho contribua para simulações e estudos cada vez mais complexos, principalmente aqueles que envolvem as mais diversas fontes de energia e Recursos Energéticos Distribuídos (RED).

6.1 DESENVOLVIMENTOS FUTUROS

Vale ressaltar que o programa de estabilidade transitória descrito e testado neste trabalho se trata de um protótipo, cujo objetivo é possibilitar o desenvolvimento de equipamentos de maior porte e complexidade. Para dar prosseguimento a este trabalho, algumas possíveis alternativas são:

1. Investigação da comunicação entre o módulo do OpenDSS e o restante do protótipo, com o objetivo de aumentar a precisão dos resultados em relação aos do ANATEM.

2. Inclusão de modelagens dinâmicas de REDs às redes elétricas de distribuição contidas em arquivos de OpenDSS.

3. Análise do impacto gerado pela atribuição de ramais de redes de distribuição a múltiplas barras de uma rede elétrica de transmissão.

4. Implementação da interface FMI 3.0, que apresenta atualizações das FMUs do tipo CS e ME. Uma dessas atualizações é a possibilidade de intercâmbio de valores de entrada e saída entre pontos de comunicação, o que elimina a restrição de pontos de comunicação discretos do CS da versão 2.0.3. Além disso, a versão 3.0 introduz a FMU do tipo *Scheduled Execution* (SE).

5. Reestruturação do protótipo escrito em *Python*, com o objetivo de melhorar o desempenho computacional e possibilitar a execução de sistemas de maior porte de forma ágil.

REFERÊNCIAS

ANDERSSON, C. *Methods and Tools for Co-Simulation of Dynamic Systems with the Functional Mock-up Interface*. Mai. 2016. Tese (Doutorado) – Mathematics Faculty of Engineering, Lund University. Defence details Date: 2016-05-04 Time: 10:15 Place: Lecture hall MH:A, Centre for Mathematical Sciences, Sölvegatan 18, Lund University, Faculty of Engineering External reviewer(s) Name: Woodward, Carol Title: Dr. Affiliation: Lawrence Livermore National Laboratory, USA —. ISBN 978-91-7623-698-7. Disponível em: <https://portal.research.lu.se/en/publications/methods-and-tools-for-co-simulation-of-dynamic-systems-with-the-f>.

ANDERSSON, C.; ÅKESSON, J.; FÜHRER, C. *PyFMI: A Python Package for Simulation of Coupled Dynamic Models with the Functional Mock-up Interface*. [S.l.]: Centre for Mathematical Sciences, Lund University, 2016. LUTFNA-5008-2016. (Technical Report in Mathematical Sciences, 2). Disponível em: <https://portal.research.lu.se/en/publications/pyfmi-a-python-package-for-simulation-of-coupled-dynamic-models-w>.

CHAGAS, I. B. O. *Metodologias de co-simulação aplicadas a sistemas de potência*. Abr. 2022. Diss. (Mestrado) – Universidade Federal de Juiz de Fora. DOI: https://doi.org/10.34019/ufjf/di/2022/00118. Disponível em: <https://repositorio.ufjf.br/jspui/handle/ufjf/14295>.

CHAGAS, I. B. O.; TOMIM, M. A. Co-simulation applied to power systems with high penetration of distributed energy resources. *Electric Power Systems Research*, v. 212, p. 108413, 2022. ISSN 0378-7796. DOI: https://doi.org/10.1016/j.epsr.2022.108413. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0378779622005508>.

DAVIS, Timothy A.; PALAMADAI NATARAJAN, Ekanathan. Algorithm 907: KLU, A Direct Sparse Solver for Circuit Simulation Problems. *ACM Transactions on Mathematical Software*, Association for Computing Machinery (ACM), v. 37, n. 3, p. 1–17, set. 2010. ISSN 1557-7295. DOI: 10.1145/1824801.1824814.

DOMMEL, H. W.; SATO, N. Fast transient stability solutions. *IEEE Transactions on Power Apparatus and Systems*, PAS-91, n. 4, p. 1643–1650, jul. 1972. ISSN 0018-9510. DOI: 10.1109/TPAS.1972.293341. Disponível em: <https://ieeexplore.ieee.org/document/4074900>.

DUGAN, R. C.; MONTENEGRO, D. *Reference Guide: The Open Distribution System Simulator (OpenDSS)*. [S.l.], jun. 2021.

ELETROBRAS CEPEL. *Análise de Transitórios Eletromecânicos 11.10.00 - Manual do Usuário*. [S.l.], ago. 2020.

EMPRESA DE PESQUISA ENERGÉTICA. *Balanço Energético Nacional - 50 anos*. [S.l.: s.n.], 2023. Disponível em: <https://www.epe.gov.br/sites-pt/publicacoes-dados-abertos/publicacoes/Documents/BEN%2050%20anos.pdf>.

_____. *Balanço Energético Nacional - Relatório Final*. [S.l.: s.n.], 2023. Disponível em: <https://www.epe.gov.br/sites-pt/publicacoes-dados-abertos/publicacoes/PublicacoesArquivos/publicacao-748/topico-687/BEN2023.pdf>.

FREITAS, P. R. R. de. *Modelos avançados de análise de redes elétricas inteligentes utilizando o software OpenDSS*. [S.l.: s.n.], 2015.

_____. *Open Distribution System Simulator (OpenDSS)*. [S.l.: s.n.]. Disponível em: <https://www.pauloradatz.me/opendss>.

GRANADOS, J. F. L. *Modelagem de carga em sistemas de distribuição de energia elétrica*. Fev. 2018. Diss. (Mestrado) – Universidade Federal de Minas Gerais. Disponível em: <http://hdl.handle.net/1843/30194>.

HINDMARSH, Alan C. et al. *User Documentation for CVODE*. [S.l.: s.n.], 2024. 7.1.1. Disponível em: <https://sundials.readthedocs.io/en/latest/cvode>.

KERSTING, W. H.; DUGAN, R. C. Recommended Practices for Distribution System Analysis. In: 2006 IEEE PES Power Systems Conference and Exposition. [S.l.: s.n.], out. 2006. P. 499–504. DOI: 10.1109/PSCE.2006.296364. Disponível em: <https://ieeexplore.ieee.org/abstract/document/4075803>.

KRISHNAMURTHY, Dheepak. *OpenDSSDirect.py 0.5.0*. [S.l.: s.n.], 2017. Disponível em: <https://dss-extensions.org/OpenDSSDirect.py/index.html>.

KUNDUR, P. *Power system stability and control*. Edição: Neal J. Balu Mark G. Lauby. [S.l.]: McGraw-Hill, Inc., jan. 1994. ISBN 0-07-035958-X.

MODELICA ASSOCIATION. *Functional Mock-up Interface for Model Exchange and Co-Simulation*. [S.l.], nov. 2021. Disponível em: <https://fmi-standard.org/>.

MODELON. *JModelica.org User Guide, version 2.2*. [S.l.], 2018. Disponível em: <https://jmodelica.org/downloads/UsersGuide.pdf>.

MOHSENI-BONAB, Seyed Masoud et al. Optimization Application in Integrated Transmission and Distribution Operation: Co-Simulation Approach. In: 2020 IEEE Power and Energy Society General Meeting (PESGM). [S.l.: s.n.], 2020. P. 1–5. DOI: 10.1109/PESGM41954.2020.9281439.

OPEN SOURCE MODELICA CONSORTIUM. *Open Modelica User's Guide*. [S.l.], mai. 2023. Disponível em: <https://openmodelica.org/home/consortium/>.

OPERADOR NACIONAL DO SISTEMA ELÉTRICO. *Análise da Perturbação do Dia 15/08/2023 às 08h30min - Relatório de Análise de Perturbação - RAP*. [S.l.: s.n.], out. 2023. Disponível em: <https://www.ons.org.br/Paginas/Noticias/20231018_Ocorrencia_de_15_de_agosto_ONS_finaliza_Relatorio_de_Analise_de_Perturbacao_RAP.aspx>.

PALENSKY, P. et al. Cosimulation of Intelligent Power Systems: Fundamentals, Software Architecture, Numerics, and Coupling. *IEEE Industrial Electronics Magazine*, v. 11, n. 1, p. 34–50, mar. 2017. ISSN 1932-4529. DOI: 10.1109/MIE.2016.2639825. Disponível em: <https://ieeexplore.ieee.org/document/7883974>.

SCHINCARIOL, R. S.; BELIN, P. R. *Sistemas Elétricos de Potência II*. [S.l.]: Editora e Distribuidora Educacional S.A., 2019. ISBN 978-85-522-1467-0.

SEXAUER, J. *Introdução ao OpenDSS*. [S.l.], jul. 2016. Tradução e Edição por Paulo Radatz e Celso Rocha. Disponível em: <https://ifgjatai.eu5.org/OpenDSS.pdf>.

SILVA, J. P. A. *Metodologia para modelagem e simulação de curto e longo termo em sistemas elétricos de potência*. Set. 2016. Diss. (Mestrado) – Universidade Federal de Juiz de Fora. Disponível em: <https://repositorio.ufjf.br/jspui/handle/ufjf/3658>.

SILVA, L. T. F. W. da. *Modelagem e simulação de sistemas de geração de energia eólica através de co-simulação*. Mar. 2020. Diss. (Mestrado) – Universidade Federal de Juiz de Fora. Disponível em: <https://repositorio.ufjf.br/jspui/handle/ufjf/11593>.

SINGH, D.; MISRA, R. K.; SINGH, D. Effect of Load Models in Distributed Generation Planning. *IEEE Transactions on Power Systems*, v. 22, n. 4, p. 2204–2212, out. 2007. ISSN 0885-8950. DOI: 10.1109/TPWRS.2007.907582. Disponível em: <https://ieeexplore.ieee.org/document/4349072>.

STOTT, B. Power system dynamic response calculations. *Proceedings of the IEEE*, v. 67, n. 2, p. 219–241, fev. 1979. ISSN 0018-9219. DOI: 10.1109/PROC.1979.11233. Disponível em: <https://ieeexplore.ieee.org/document/1455502>.

THEODORO, T. S. et al. MatLab-OpenDSS co-simulation environment: An alternative tool to investigate DSG connection. In: 2018 Simposio Brasileiro de Sistemas Eletricos (SBSE). [S.l.: s.n.], jun. 2018. P. 1–7. DOI: 10.1109/SBSE.2018.8395643. Disponível em: <https://ieeexplore.ieee.org/document/8395643>.

TOMIM, M. A. *Parallel computation of large power system networks using the multi-area Thévenin equivalents*. Jul. 2009. Tese (Doutorado) – University of British Columbia. DOI: http://dx.doi.org/10.14288/1.0067477. Disponível em: <https://open.library.ubc.ca/collections/ubctheses/24/items/1.0067477>.

TOMIM, Marcelo A.; HENRIQUES, Ricardo M.; PASSOS FILHO, João A. Introduction to OmniPES: a Novel Modelica Library for Power Systems Modeling and Analysis. In: UNIVERSITÉ PARIS-SACLAY, 4 jun. 2024. XXIII Power Systems Computation Conference (PSCC'2024). Paris, France: [s.n.]. Aceito para publicação.

APÊNDICE A – COMPARAÇÃO ENTRE FMUS DO TIPO *CO-SIMULATION* E *MODEL EXCHANGE*

Neste dissertação, apresenta-se a interface FMI de versão 2.0.3, onde há dois tipos de modelos dinâmicos possíveis: *Co-Simulation* (CS, ou Cossimulação) e *Model Exchange* (ME, ou Intercâmbio de Modelos). Sabendo disso, este apêndice possui os seguintes objetivos: realizar a implementação de ambos os modelos a um sistema de potência simplificado, comparar os resultados obtidos e avaliar qual é o mais adequado para a simulação de sistemas elétricos de maior porte e complexidade.

A.1 SIMULAÇÃO DE UM SISTEMA RADIAL

Primeiramente, considera-se o sistema radial mostrado na Figura 41, onde se tem um turbogerador conectado a um sistema de grande porte, representado por um barramento infinito, através de um transformador elevador e duas linhas de transmissão em paralelo. As reatâncias mostradas no diagrama unifilar são dadas por unidade nas bases de 2220 [MVA] e 24 [kV].

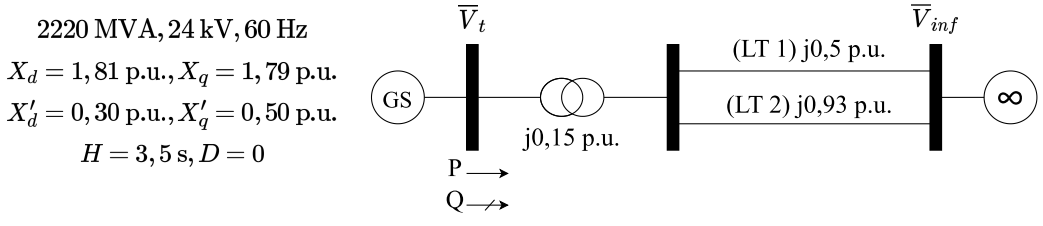
Além disso, utiliza-se o modelo clássico para o gerador síncrono e considera-se o sistema operando em regime permanente nas seguintes condições: $P = 1776$ [MW], $V_t = 1{,}0$ [pu] e $V_{inf} = 1{,}0$ [pu]. As equações de oscilação da máquina do modelo clássico estão mostradas em (A.1).

$$\frac{d\omega}{dt} = \frac{P_m - P_e - D_m\,\omega}{2H} \tag{A.1a}$$

$$\frac{d\delta}{dt} = \omega_b \cdot \omega \tag{A.1b}$$

Definidos os parâmetros do sistema radial, realiza-se uma simulação no tempo de 10 [s], na qual um curto-circuito trifásico de reatância $X_f = 0{,}01$ [pu] é aplicado à barra de alta tensão do transformador no instante $t_f = 0{,}1$ [s] e removido 100 [ms] depois, junto

Figura 41 – Diagrama unifilar do sistema radial máquina-barramento infinito.



Fonte: Elaborada pelo autor (2024).

com a linha LT2. Os outros parâmetros adotados para a simulação no tempo são: passo de integração igual a 1 [ms], tolerância absoluta de 10^{-4} [pu] e relativa de 10^{-2} %.

Com a modelagem clássica do gerador síncrono no programa OMEdit, são exportados os dois tipos de FMUs da interface FMI 2.0.3 para que sejam realizadas duas simulações, conforme mostrado nas subseções a seguir.

A.1.1 Modelagem clássica através da FMU do tipo CS

Para a primeira simulação no tempo, a ferramenta escolhida para a solução da FMU do tipo CS é o CVode (*C-language Variable-coefficients Ordinary Differential Equations*), que consiste em um solucionador de problemas de valores iniciais envolvendo sistemas de equações diferenciais ordinárias dadas na forma explícita, conforme mostrado em (A.2).

$$\begin{aligned} y' &= f(y,t) \\ y(t_0) &= y_0 \end{aligned} \tag{A.2}$$

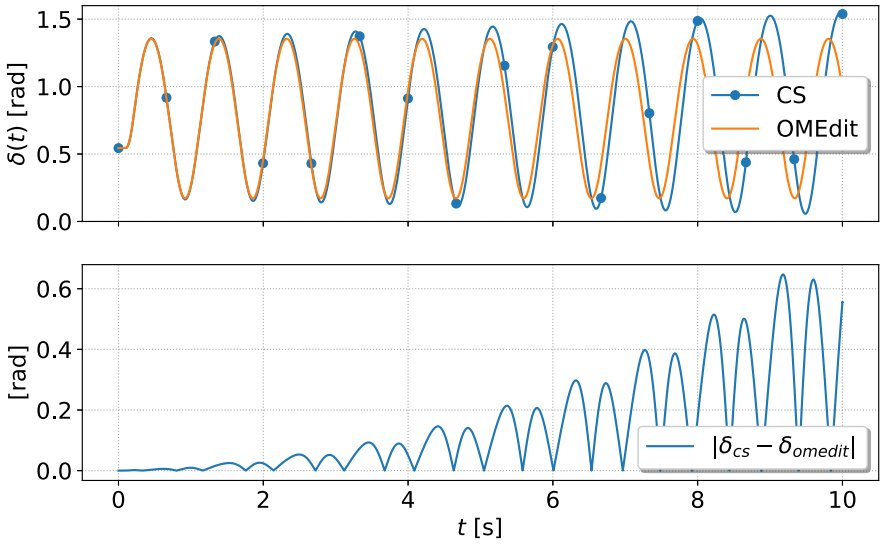
Esse solucionador pertence ao pacote SUNDIALS (*SUite of Nonlinear and DIfferential/ALgebraic equation Solvers*), que por sua vez tem como objetivo fornecer não só o CVode, como também outros solucionadores de sistemas não lineares que possam ser incorporados a programas de simulação preexistentes (HINDMARSH et al., 2024).

Com a FMU do tipo *Co-Simulation* definida e implementada à simulação do sistema radial dentro do ambiente *Python*, tem-se na Figura 42 a curva do ângulo interno do gerador síncrono, bem como a curva de referência obtida pelo OMEdit, onde o sistema radial foi criado originalmente. Com essas duas curvas, tem-se também o gráfico correspondente aos erros absolutos do ângulo interno.

Com base na Figura 42, é possível observar que os resultados da FMU do tipo CS (curva azul) a princípio coincidem com o comportamento puramente oscilatório do OMEdit (curva laranja). No entanto, à medida em que a simulação avança, é possível observar um aumento gradativo do período das oscilações da curva azul, bem como um aumento gradativo das amplitudes. Como consequência, os erros absolutos do ângulo do gerador apresentam picos em torno de 0,6 [rad], o que equivale a um erro relativo de mais de 150 %.

Portanto, considerando as análises gráficas e quantitativas realizadas, tem-se que a implementação da FMU do tipo CS não apresentou resultados satisfatórios. Uma possível explicação para esse cenário diz respeito à forma como a troca de informações da interface é realizada, isto é, por meio de pontos de comunicação discretos, o que acarreta atrasos dos resultados obtidos.

Figura 42 – Ângulos internos do gerador síncrono - FMU do tipo CS.



Fonte: Elaborada pelo autor (2024).

A.1.2 Modelagem clássica através da FMU do tipo ME

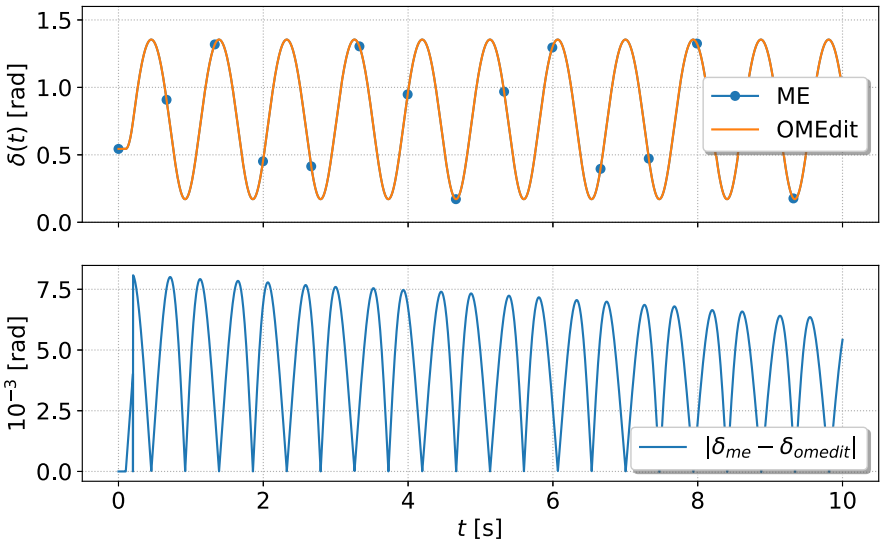
Agora, para a segunda simulação no tempo, considera-se a FMU do tipo ME, que ao contrário da FMU do tipo CS, não possui o solucionador embutido. Em vez disso, o solucionador externo escolhido corresponde à classe `TrapezoidalIntegrator`, criada para esta dissertação e mostrada no Apêndice C.

Com a FMU do tipo *Model Exchange*, realiza-se a mesma simulação da subseção anterior. Com isso, tem-se na Figura 43 a curva do ângulo interno do gerador síncrono, bem como a curva de referência do OMEdit e o gráfico de erros absolutos.

Com base na Figura 43, é possível observar que os resultados da FMU do tipo ME (curva azul) coincidem com o comportamento puramente oscilatório do OMEdit (curva laranja) durante toda a simulação. Os erros absolutos do ângulo do gerador apresentam picos em torno de $7{,}5 \cdot 10^{-3}$ [rad], o que equivale a um erro relativo de menos de 2 %.

Portanto, considerando as análises gráficas e quantitativas, tem-se que a implementação da FMU do tipo ME apresentou resultados significativamente satisfatórios em relação aos do tipo CS. Com isso, define-se que a solução do modelo de máquinas desta dissertação deve adotar a FMU do tipo *Model Exchange*, com o objetivo de garantir a precisão dos resultados de simulação.

Figura 43 – Ângulos internos do gerador síncrono - FMU do tipo ME.



Fonte: Elaborada pelo autor (2024).

APÊNDICE B – MÉTODO DE INTEGRAÇÃO TRAPEZOIDAL

Métodos de integração numérica consistem em formas de encontrar a solução de equações diferenciais com condições iniciais conhecidas. Essas soluções são encontradas através da conversão de equações diferenciais (de natureza contínua) em equações de diferenças (de natureza discreta), que por sua vez são utilizadas para calcular valores da solução.

O método trapezoidal consiste num método de integração numérica que leva em conta o valor médio das derivadas dos instantes t e $t + \Delta t$. Pode ser interpretado como um aperfeiçoamento do método de Euler, que leva em conta apenas a derivada do instante t.

B.1 DEDUÇÃO MATEMÁTICA

Sendo $\mathbf{x}$ o vetor de variáveis de estado e $\mathbf{v}$ o vetor de variáveis algébricas, considera-se uma equação diferencial da forma mostrada em (B.1).

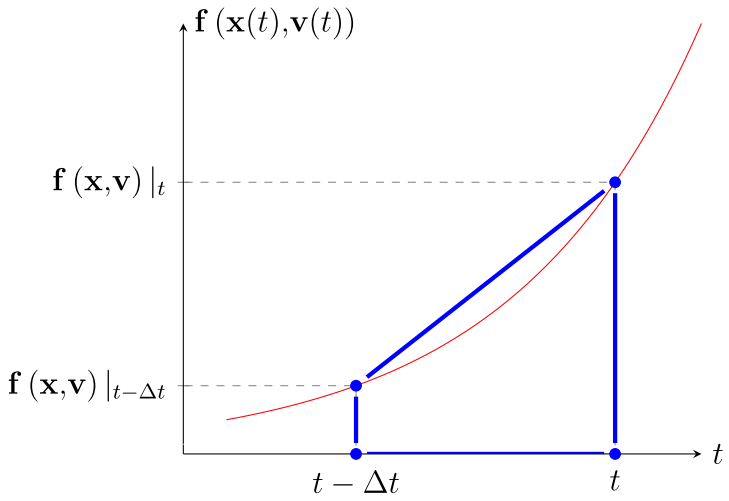
$$\dot{\mathbf{x}}(t) = \mathbf{f}\left(\mathbf{x}(t), \mathbf{v}(t)\right) \tag{B.1}$$

No método de integração trapezoidal, os passos de integração são executados com base na área de um trapézio, conforme a Figura 44. Dessa forma, os pontos correspondentes a cada instante de tempo são encontrados através da discretização da função $\mathbf{f}$.

Considerando a Figura 44, faz-se o seguinte:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}\left(\mathbf{x}(t), \mathbf{v}(t)\right) \\ \frac{d\mathbf{x}(t)}{dt} &= \mathbf{f}\left(\mathbf{x}(t), \mathbf{v}(t)\right) \\ d\mathbf{x}(t) &= \mathbf{f}\left(\mathbf{x}(t), \mathbf{v}(t)\right)\, dt \end{aligned} \tag{B.2}$$

Figura 44 – Demonstração gráfica do método trapezoidal.



Fonte: Elaborada pelo autor (2023).

Aplicando a ambos os lados uma integral no intervalo de $t - \Delta t$ até t, pegam-se dois pontos da função $\mathbf{f}$ a ser integrada (em vermelho) e dois pontos do eixo das abscissas para servirem como vértices desse trapézio (em azul). Com isso, as equações diferenciais de $\mathbf{f}$ são discretizadas conforme mostrado em (B.3).

$$\begin{aligned} \int_{t-\Delta t}^{t} d\mathbf{x}(\tau) &= \int_{t-\Delta t}^{t} \mathbf{f}\left(\mathbf{x}(\tau), \mathbf{v}(\tau)\right)\, d\tau \\ \mathbf{x}(t) - \mathbf{x}(t-\Delta t) &= \int_{t-\Delta t}^{t} \mathbf{f}\left(\mathbf{x}(\tau), \mathbf{v}(\tau)\right)\, d\tau \\ &\approx \left[\mathbf{f}(\mathbf{x},\mathbf{v})\Big|_{t} + \mathbf{f}(\mathbf{x},\mathbf{v})\Big|_{t-\Delta t}\right]\, \frac{\Delta t}{2} \end{aligned} \tag{B.3}$$

Portanto, a equação do método trapezoidal fica da forma apresentada em (B.4).

$$\mathbf{x}(t) = \mathbf{x}(t-\Delta t) + \frac{1}{2}\Big[\mathbf{f}(t) + \mathbf{f}(t-\Delta t)\Big]\Delta t \tag{B.4}$$

Conforme a equação (B.4), tem-se que o método de integração trapezoidal consiste num método implícito, porque além de depender do valor da derivada da função no passo anterior $(t - \Delta t)$, também depende do valor da derivada no passo atual (t).

Separando os termos presentes e os termos passados da equação (B.4), tem-se a nova equação de $\mathbf{x}(t)$ apresentada em (B.5). Nessa equação, tem-se o vetor de termos históricos $\mathbf{x}_h(t)$, correspondentes às funções das variáveis de estado do instante imediatamente anterior $t - \Delta t$. Portanto, o vetor $\mathbf{x}_h(t)$ é mantido fixo durante a solução do instante t e precisa ser atualizado quando a solução convergente é encontrada.

$$\mathbf{x}(t) = \frac{\Delta t}{2}\,\mathbf{f}(t) + \mathbf{x}_h(t) \tag{B.5a}$$

$$\mathbf{x}_h(t) = \mathbf{x}(t-\Delta t) + \frac{\Delta t}{2}\,\mathbf{f}(t-\Delta t) \tag{B.5b}$$

B.2 EXEMPLO

Para exemplificar a aplicação do método de integração trapezoidal, considera-se a equação diferencial mostrada em (B.6).

$$\begin{cases} 0{,}5\,\dfrac{dx}{dt} = -x + u \\ \left.\dfrac{dx}{dt}\right|_{t=0} = 0 \end{cases} \tag{B.6}$$

onde:

$$u = \begin{cases} 1, & t < 1{,}0 \text{ [s]} \\ -1, & t \geq 1{,}0 \text{ [s]} \end{cases}$$

Reescrevendo a equação diferencial de (B.6), tem-se a equação de $f(t)$ mostrada em (B.7).

$$\begin{aligned} 0{,}5\, f(t) &= -x(t) + u(t) \\ f(t) &= 2\Big(- x(t) + u(t)\Big) \end{aligned} \tag{B.7}$$

Reunindo a equação diferencial de (B.7) com a equação do método trapezoidal de (B.5a), tem-se o sistema de equações mostrado em (B.8).

$$\begin{cases} f(t) = 2\Big(- x(t) + u(t)\Big) \\ x(t) = \dfrac{\Delta t}{2}\, f(t) + x_h(t) \end{cases} \tag{B.8}$$

Como os termos $x_h(t)$ e $u(t)$ são conhecidos a cada instante de tempo, o sistema mostrado em (B.8) possui duas equações e duas incógnitas, que são o estado $x(t)$ e a derivada $f(t)$.

Para a solução desse sistema de equações, pode-se adotar o método de iteração funcional. Portanto, para cada instante de tempo, testam-se diferentes valores de $x(t)$ de forma iterativa até atender uma determinada tolerância de convergência. Quando o sistema converge, a solução correspondente é utilizada para atualizar o termo histórico $x_h(t)$, conforme mostrado em (B.9).

$$x_h(t) = x(t - \Delta t) + \frac{\Delta t}{2}\, f(t - \Delta t) \tag{B.9}$$

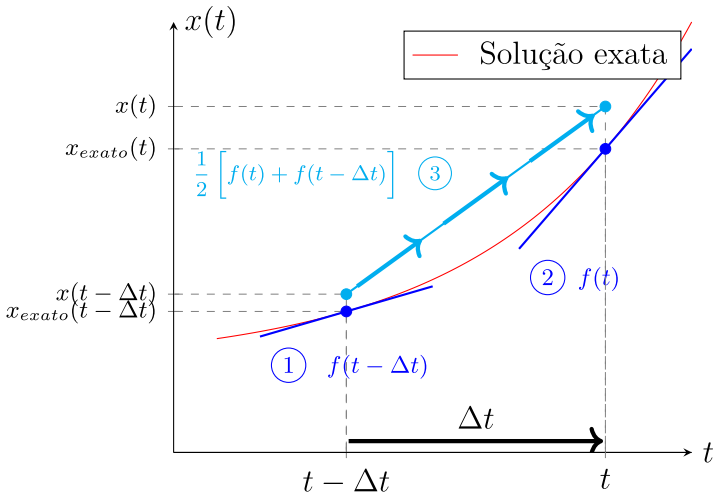
Como a entrada $u(t)$ se altera no instante de tempo $1{,}0^+$ [s], o cálculo de $x(t)$ é realizado da seguinte forma:

- Para $t_e \neq 1{,}0^+$ [s]: calcula-se a solução convergente das equações de (B.8) e, após a convergência, a equação (B.9).
- Para $t_e = 1{,}0^+$ [s]: tem-se $x(t_e^+) = x(t_e^-)$ para garantir a natureza contínua dos estados. Com isso, substituem-se os valores de $x(t_e^+)$ e $u(t_e^+)$ nas equações mostradas em (B.10), atualizando os valores de $f(t_e^+)$ e $x_h(t_e^+)$.

$$\begin{cases} f(t_e^+) = 2\Big(- x(t_e^+) + u(t_e^+)\Big) \\ x_h(t_e^+) = x(t_e^+) + \dfrac{\Delta t}{2}\, f(t_e^+) \end{cases} \tag{B.10}$$

Portanto, a partir de um passo de integração Δt e de um estado inicial $x(0)$, a solução convergente de cada $x(t)$ corresponde ao avanço de um passo de integração trapezoidal da Figura 45. Isto é, com os coeficientes angulares das retas ① e ②, o coeficiente angular da reta ③ é encontrado através de uma média aritmética das duas retas.

Figura 45 – Avanço de um passo de integração trapezoidal.



Fonte: Elaborada pelo autor (2023).

Retas da Figura 45:

①: reta de coeficiente angular $f(t-\Delta t)$.

②: reta de coeficiente angular $f(t)$.

③: reta de coeficiente angular $\dfrac{1}{2}\Big[f(t)+f(t-\Delta t)\Big]$.

Conhecidos os valores do instante anterior $t-\Delta t$, o método trapezoidal encontra o estado atual $x(t)$ (bem como a derivada $f(t)$) de forma implícita. A diferença entre ele e o estado exato é $|x-x_{exato}|=\varepsilon$. Para manter o valor de ε o mais próximo possível do zero ao longo da integração numérica, é necessário escolher um valor de Δt apropriado.

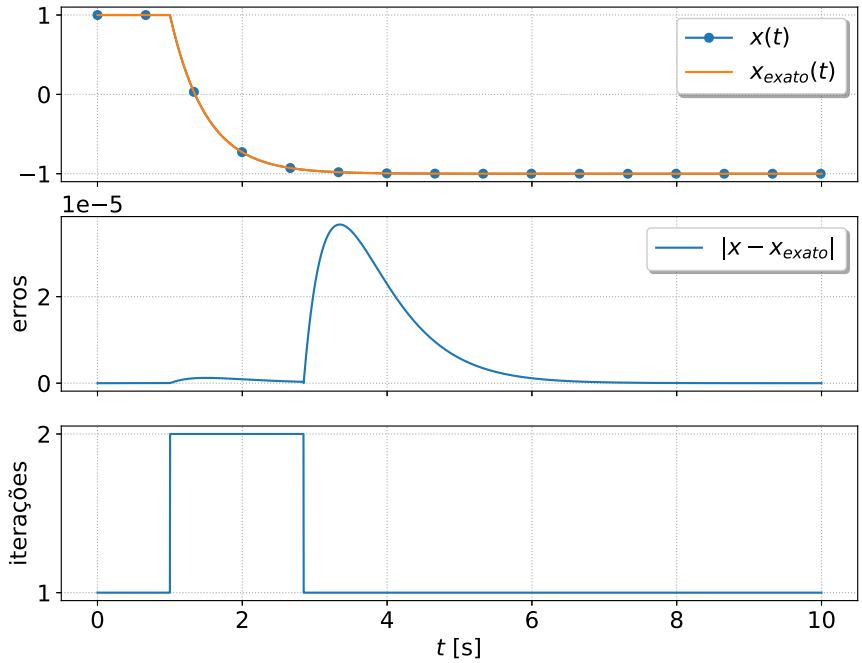
No caso do sistema de equações de (B.6), o estado inicial correspondente é $x(0)=1$. Além disso, a simulação no tempo correspondente adota o passo de integração igual a $\Delta t = 1$ [ms] e o instante final igual a $t_f = 10{,}0$ [s]. Com isso, comparam-se os resultados obtidos com a solução exata mostrada em (B.11). A comparação está mostrada na Figura 46.

$$x_{exato}(t)=\begin{cases}1, & t<1{,}0\ \text{[s]}\\ 2e^{-2(t-1)}-1, & t\geq 1{,}0\ \text{[s]}\end{cases} \tag{B.11}$$

Com base na Figura 46, o erro absoluto máximo entre $x(t)$ e $x_{exato}(t)$ ficou na ordem de 10^{-5}, validando a aplicação do método trapezoidal para a discretização de equações diferenciais com condições iniciais conhecidas. Além disso, alguns instantes de tempo exigiram 2 iteração para a convergência.

Agora, com o objetivo de acelerar a convergência do método trapezoidal, utiliza-se um passo preditor no início de cada instante de tempo. Para isso, adota-se o método de

Figura 46 – Comparação dos resultados do método trapezoidal.



Fonte: Elaborada pelo autor (2023).

Euler explícito, conforme a equação mostrada em (B.12).

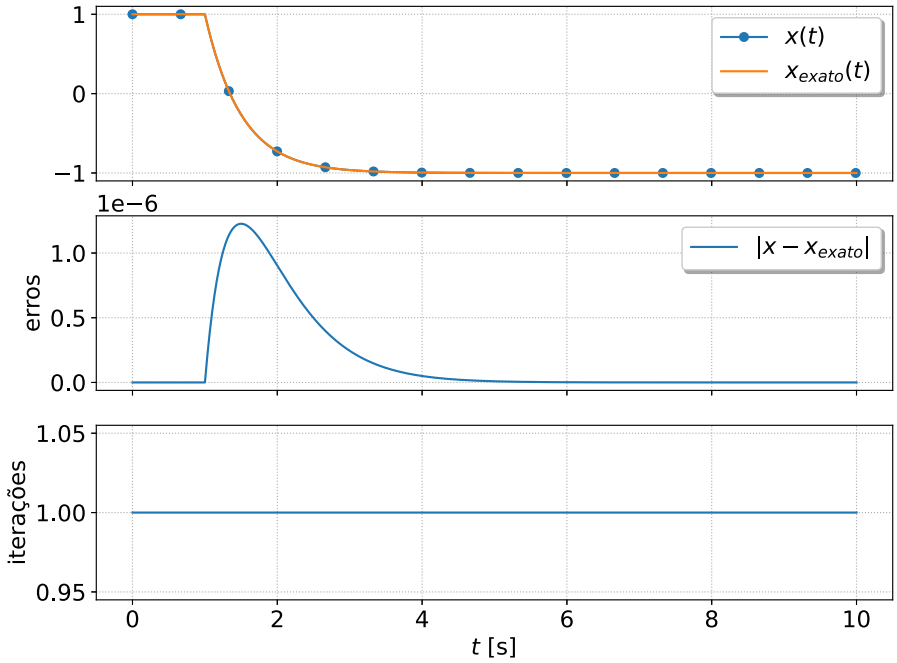
$$x^{(0)}(t) = x(t - \Delta t) + \Delta t \cdot f(t - \Delta t) \tag{B.12}$$

Utilizando o passo preditor para solucionar novamente o sistema mostrado em (B.8), tem-se os novos resultados mostrados na Figura 47.

Em comparação com a solução anterior da Figura 46, os novos resultados da Figura 47 mostram que o erro absoluto máximo entre $x(t)$ e $x_{exato}(t)$ agora diminuiu para a ordem de 10^{-6}. Além disso, todos os instantes de tempo agora exigiram apenas 1 iteração para a convergência.

Com isso, define-se que a solução do modelo de máquinas desta dissertação deve adotar o passo preditor através do método de Euler explícito, com o objetivo de melhorar o desempenho computacional.

Figura 47 – Comparação dos resultados do método trapezoidal com passo preditor.



Fonte: Elaborada pelo autor (2023).

APÊNDICE C – ARQUIVOS DO SOLUCIONADOR EXTERNO À FMU

Nesta dissertação, a modelagem de máquinas elétricas é realizada por uma ferramenta computacional denominada *Functional Mock-up Interface* (FMU). Conforme descrito no Capítulo 2, esse tipo de ferramenta compacta as equações de modelos matemáticos, que por sua vez podem ser montados em diferentes programas. Como o tipo adotado de FMU é o *Model Exchange*, a solução dessas equações compactadas deve ser realizada por um solucionador externo à FMU.

Através da linguagem de programação *Python*, a montagem desse solucionador externo implementa funções baseadas no método de integração trapezoidal. Os arquivos em `.py` correspondentes estão apresentadas neste apêndice.

C.1 ARQUIVO `trapezoidal.py`

Primeiro, tem-se o Algoritmo 15, correspondente ao arquivo `trapezoidal.py`, onde se adota o método de integração trapezoidal.

O arquivo `trapezoidal.py` possui a classe chamada `TrapezoidalIntegrator`, cuja entrada é a classe `Integrator` (importada do `ode_solver.py`, apresentada na próxima seção).

As funções da classe `TrapezoidalIntegrator` são:

Algoritmo 15 – Arquivo `trapezoidal.py`.

```
1  from ode_solver import Integrator

2  class TrapezoidalIntegrator(Integrator):

3      def __init__(self, initialized_fmu, h, tend, atol, rtol):
4          super().__init__(initialized_fmu, h, tend, atol, rtol)
5          self.hist = self.x_prev + 0.5 * self.dt * self.der_x_prev

6      def predictor(self):
7          # Estimate states by means of the explicit euler method
8          self.x = self.x_prev + self.dt * self.der_x_prev

9      def step(self):
10         # Estimate states by means of the trapezoidal rule
11         self.der_x = self.fmu.get_derivatives()
12         self.x = self.hist + 0.5 * self.dt * self.der_x

13     def update_history(self):
14         # Update history terms of the states
15         self.x_prev = self.x[:]
16         self.der_x_prev = self.der_x[:]
17         self.hist = self.x_prev + 0.5 * self.dt * self.der_x_prev
```

Fonte: Elaborado pelo autor (2024)

- Função `__init__` (linha 3): consiste na inicialização dos seguintes parâmetros de simulação no tempo:

  - Objeto que contém o modelo inicializado da FMU (`initialized_fmu`), passo de integração (`h`), instante de tempo final (`tend`), tolerância absoluta (`atol`) e tolerância relativa (`rtol`): parâmetros definidos pelo usuário no ambiente de simulação.
  - Termo histórico inicial (`hist`): calculado pelo método trapezoidal, através do estado inicial (`x_prev`) e da derivada inicial desse estado (`der_x_prev`), conforme mostrado em (C.1).

$$x_{h,0} = x(0) + \frac{\Delta t}{2} f(0) \tag{C.1}$$

- Função `predictor` (linha 8): consiste na estimativa inicial do estado através do método de Euler explícito, conforme mostrado em (C.2). Tem como objetivo acelerar a convergência do processo de integração.

$$x^{(0)} = x_{prev} + \Delta t \cdot f_{prev} \tag{C.2}$$

- Função `step` (linha 9): consiste em extrair do modelo da FMU a derivada atual (`der_x`) pelo comando `get_derivatives` e utilizá-la para a estimativa do estado por meio do método trapezoidal, conforme mostrado em (C.3).

$$\begin{cases} f \leftarrow \texttt{fmu.get\_derivatives} \\ x = \dfrac{\Delta t}{2} f + x_h \end{cases} \tag{C.3}$$

- Função `update_history` (linha 13): consiste em atualizar o termo histórico (`hist`) a ser utilizado pelo instante de tempo seguinte. Essa atualização está mostrada em (C.4).

$$\begin{cases} x_{prev} = x \\ f_{prev} = f \\ x_h = x_{prev} + \dfrac{\Delta t}{2} f_{prev} \end{cases} \tag{C.4}$$

C.2 ARQUIVO `ode_solver.py`

Agora, tem-se o Algoritmo 16, correspondente ao arquivo `ode_solver.py`.

O arquivo `ode_solver.py` possui a classe chamada `Integrator`, cujas funções são:

- Função `__init__` (linha 4): consiste na inicialização dos seguintes parâmetros de simulação no tempo:

Algoritmo 16 – Arquivo `ode_solver.py`.

```
1  import numpy as np
2  from abc import ABC, abstractmethod


3  class Integrator(ABC):

4      def __init__(self, initialized_fmu, h, tend, atol, rtol):
5          self.fmu = initialized_fmu
6          self.dt = h
7          self.tf = tend
8          self.rtol = rtol
9          self.atol = atol

10         self.x_prev = self.fmu.continuous_states[:]
11         self.der_x_prev = self.fmu.get_derivatives()[:]

12         self.x = np.zeros_like(self.fmu.continuous_states)
13         self.der_x = np.zeros_like(self.fmu.continuous_states)

14     @abstractmethod
15     def step(self):
16         pass

17     @abstractmethod
18     def update_history(self):
19         pass

20     def check_convergence(self):
21         # Check absolute error
22         abs_error = np.abs(self.fmu.continuous_states - self.x)
23         abs_convergence = (np.max(abs_error) < self.atol)

24         # Check relative error
25         rel_error = abs_error - self.rtol * np.abs(self.fmu.continuous_states)
26         rel_convergence = (np.max(rel_error) < 1e-10)

27         # Check model convergence:
28         model_convergence = (abs_convergence and rel_convergence)

29         return model_convergence
```

Fonte: Elaborado pelo autor (2024)

  - Objeto que contém o modelo inicializado da FMU (`initialized_fmu`), passo de integração (`h`), instante de tempo final (`tend`), tolerância absoluta (`atol`) e tolerância relativa (`rtol`): parâmetros definidos pelo usuário no ambiente de simulação.
  - Estado inicial (`x_prev`) e derivada inicial do estado (`der_x_prev`): parâmetros extraídos a partir das condições iniciais do modelo da FMU.
  - Estado atual (`x`) e derivada atual do estado (`der_x`): parâmetros inicializados como valores nulos para que posteriormente sejam calculados pela função `step`.

- Funções `step` (linha 15) e `update_history` (linha 18): como dependem do método de integração escolhido, escolhe-se realizar ambas as funções num arquivo separado chamado `trapezoidal.py`, conforme mostrado anteriormente na Seção C.1.

- Função `check_convergence` (linha 20): consiste em verificar se os erros absolutos e relativos (`abs_error` e `rel_error`) do estado atendem dadas tolerâncias de convergência (`atol` e `rtol`). Para isso, utiliza-se a variável binária `model_convergence` (linha 28), conforme mostrado em (C.5).

$$\texttt{model\_convergence} = (\texttt{abs\_convergence}\ \texttt{and}\ \texttt{rel\_convergence}) \tag{C.5}$$

  Em caso de convergência simultânea dos erros absolutos e relativos, tem-se a variável binária `model_convergence` igual a `True`. Caso contrário, tem-se a variável `model_convergence` igual a `False`.

APÊNDICE D – DADOS DOS SISTEMAS TESTE

Neste trabalho, utilizam-se como exemplo um sistema de transmissão de 11 barras e um sistema de distribuição de 38 barras durante a simulação no tempo, com o objetivo de validar os métodos implementados. Os dados desses dois sistemas estão apresentados neste apêndice.

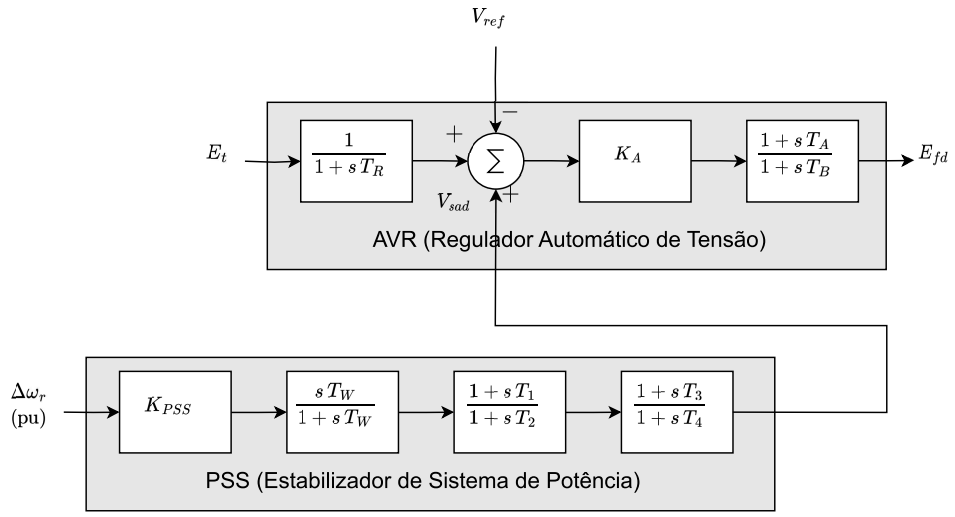
D.1 SISTEMA DE TRANSMISSÃO DE 11 BARRAS

O sistema de transmissão de 11 barras, apresentado em (KUNDUR, 1994), consiste em duas áreas similares interconectadas por duas linhas de transmissão. Na barra 7, há um banco de capacitores C_7 de 200 [Mvar] e uma carga L_7 de potências 967 [MW] e 100 [Mvar]. Na barra 9, há um banco de capacitores C_9 de 350 [Mvar] e uma carga L_9 de potências 1767 [MW] e 100 [Mvar].

Cada área do sistema de 11 barras possui duas unidades geradoras, onde as unidades G_1 e G_2 possuem constante de inércia $H = 6{,}5$ [s], e as unidades G_3 e G_4 possuem constante de inércia $H = 6{,}175$ [s]. As 4 unidades geradoras são conectadas através de transformadores de reatância de 0,15 [pu] nas bases 900 [MVA] e 20/230 [kV]. Além disso, o modelo de máquina síncrona das 4 unidades geradoras possui um regulador automático de tensão (AVR) e um estabilizador de sistema de potência (PSS). O diagrama de blocos correspondente a esses equipamentos está mostrado na Figura 48, e os seus coeficientes estão mostrados na Tabela 14.

Os dados elétricos das unidades geradoras estão mostrados na Tabela 15, cujos

Figura 48 – Diagrama de blocos do AVR e do PSS.



Fonte: (KUNDUR, 1994)

Tabela 14 – Dados do AVR e do PSS do sistema de 11 barras.

Parâmetro	**Valor**
K_A	200,0
T_R	0,01
K_{PSS}	20,0
T_W	10,0
T_1	0,05
T_2	0,02
T_3	3,0
T_4	5,4

Fonte: (KUNDUR, 1994)

Tabela 15 – Dados elétricos das unidades geradoras do sistema de 11 barras.

Parâmetro	**Valor**
X_d	1,8 [pu]
X_q	1,7 [pu]
X_l	0,2 [pu]
X'_d	0,3 [pu]
X'_q	0,55 [pu]
X''_d	0,25 [pu]
X''_q	0,25 [pu]
R_a	0,0025 [pu]
T'_{d0}	8,0 [s]
T'_{q0}	0,4 [s]
T''_{d0}	0,03 [s]
T''_{q0}	0,05 [s]

Fonte: (KUNDUR, 1994)

valores base são 900 [MVA] e 20 [kV].

Os dados elétricos das linhas de transmissão do sistema de 11 barras estão apresentados na Tabela 16, cujos valores base são 100 [MVA] e 230 [kV]. Os comprimentos dessas linhas estão apresentados na Tabela 17.

D.2 SISTEMA DE DISTRIBUIÇÃO DE 38 BARRAS

O sistema de distribuição de 38 barras, apresentado em (SINGH; MISRA; SINGH, 2007), consiste em um sistema de 69 [kV] composto por 1 barra de conexão ao sistema de transmissão, 32 barras de carga (barras 2 a 33) e 5 geradores (barras 34 a 38). Nesse

Tabela 16 – Dados elétricos das linhas de transmissão do sistema de 11 barras.

Parâmetro	**Valor [pu/km]**
r	0,0001
x_l	0,001
b_c	0,00175

Fonte: (KUNDUR, 1994)

Tabela 17 – Comprimentos das linhas de transmissão do sistema de 11 barras.

Linha	**Comprimento [km]**
5-6	25
6-7	10
7-8	110
8-9	110
9-10	10
10-11	25

Fonte: (KUNDUR, 1994)

sistema, a barra 1 é o PAC, isto é, a barra 1 é conectada ao PAC do sistema de transmissão. Além disso, neste trabalho a sua representação é da forma trifásica equilibrada.

Os dados das linhas e dos transformadores de distribuição estão apresentados na Tabela 18 e na Tabela 19, respectivamente, nas bases 100 [MVA] e 69 [kV]. Os dados das cargas estão apresentados na Tabela 20. Os dados dos geradores estão apresentados na Tabela 21.

Tabela 18 – Dados das linhas de distribuição do sistema de 38 barras.

De	Para	r (%)	x (%)	De	Para	r (%)	x (%)
1	2	0,0574	0,0293	17	18	0,4558	0,3574
2	3	0,307	0,1564	2	19	0,1021	0,0974
3	4	0,2279	0,1161	19	20	0,9366	0,844
4	5	0,2373	0,1209	20	21	0,255	0,2979
5	6	0,51	0,4402	21	22	0,4414	0,5836
6	7	0,1166	0,3853	3	23	0,2809	0,192
7	8	0,443	0,1464	23	24	0,5592	0,4415
8	9	0,6413	0,4608	24	25	0,5579	0,4366
9	10	0,6501	0,4608	6	26	0,1264	0,0644
10	11	0,1224	0,0405	26	27	0,177	0,0901
11	12	0,2331	0,0771	27	28	0,6594	0,5814
12	13	0,9141	0,7192	28	29	0,5007	0,4362
13	14	0,3372	0,4439	29	30	0,316	0,161
14	15	0,368	0,3275	30	31	0,6067	0,5996
15	16	0,4647	0,3394	31	32	0,1933	0,2253
16	17	0,8026	1,0716	32	33	0,2123	0,3301

Fonte: (SINGH; MISRA; SINGH, 2007)

Tabela 19 – Dados dos transformadores do sistema de 38 barras.

De	Para	r (%)	x (%)	Tape
8	34	1,2453	1,2453	1,000
9	35	1,2453	1,2453	1,000
12	36	1,2453	1,2453	1,000
18	37	1,2453	1,2453	1,000
25	38	1,2453	1,2453	1,000

Fonte: (SINGH; MISRA; SINGH, 2007)

Tabela 20 – Dados das cargas do sistema de 38 barras, onde R=Residencial, I=Industrial e C=Comercial.

Carga	$\mathbf{P}_l$ **[MW]**	$\mathbf{Q}_l$ **[Mvar]**	**Carga**	$\mathbf{P}_l$ **[MW]**	$\mathbf{Q}_l$ **[Mvar]**
2	10,0	6,0	18	9,0	4,0
3	9,0	4,0	19	6,0	2,0
4	12,0	8,0	20	9,0	4,0
5	6,0	3,0	21	9,0	4,0
6	6,0	2,0	22	9,0	4,0
7	20,0	10,0	23	9,0	5,0
8	20,0	10,0	24	42,0	20,0
9	6,0	2,0	25	42,0	20,0
10	6,0	2,0	26	6,0	2,5
11	4,5	3,0	27	6,0	2,5
12	6,0	3,5	28	6,0	2,0
13	6,0	3,5	29	12,0	7,0
14	12,0	8,0	30	20,0	60,0
15	6,0	1,0	31	15,0	7,0
16	6,0	2,0	32	21,0	10,0
17	6,0	2,0	33	6,0	4,0

Fonte: (SINGH; MISRA; SINGH, 2007)

Tabela 21 – Dados dos geradores do sistema de 38 barras.

Gerador	$\mathbf{P}_g$ **[MW]**	$\mathbf{Q}_g$ **[Mvar]**
34	88,75	137,88
35	44,40	90,00
36	14,80	30,00
37	29,59	57,09
38	14,40	30,00

Fonte: Elaborada pelo autor (2023)

APÊNDICE E – MÉTODO ITERATIVO DE *NEWTON-RAPHSON*

O método iterativo de *Newton-Raphson* consiste num método de solução de equações algébricas não lineares do tipo $f(x) = 0$. Para isso, são realizadas linearizações sucessivas de $f(x)$ a partir de uma estimativa inicial $x^{(0)}$.

Algumas propriedades desse método são:

- Sensibilidade às estimativas iniciais: dependendo do $x^{(0)}$ escolhido, a convergência pode exigir muitas ou poucas iterações, ou talvez não convirja.
- Independência da quantidade de equações: o número necessário de iterações para a convergência independe da dimensão de $f(x)$.

E.1 DEDUÇÃO MATEMÁTICA

Considerando uma única função $y = f(x)$, o método iterativo de *Newton-Raphson* procura encontrar a solução da equação $y = y_{esp}$ (por exemplo, $y_{esp} = 0$). Para isso, a função é linearizada em torno de um ponto inicial $\left(x^{(0)}, f(x^{(0)})\right)$, através da expansão de primeira ordem da Série de *Taylor*, conforme a equação (E.1).

$$\begin{aligned} y &= f(x) \\ y &= f(x^{(0)}) + \left.\frac{df(x)}{dx}\right|_{x^{(0)}} \cdot (x - x^{(0)}) \\ y - f(x^{(0)}) &= \left.\frac{df(x)}{dx}\right|_{x^{(0)}} \cdot (x - x^{(0)}) \end{aligned}$$

$$\Delta y^{(0)} = \left.\frac{df(x)}{dx}\right|_{x^{(0)}} \cdot \Delta x^{(0)} \tag{E.1}$$

O termo derivativo também recebe o nome de jacobiano. Portanto, pode ser substituído por um J. Com isso, a equação fica da forma apresentada em (E.2).

$$\Delta y^{(0)} = J^{(0)} \cdot \Delta x^{(0)} \tag{E.2}$$

Conhecidos os termos $\Delta y^{(0)}$ e $J^{(0)}$, o termo $\Delta x^{(0)}$ é calculado pela equação (E.3).

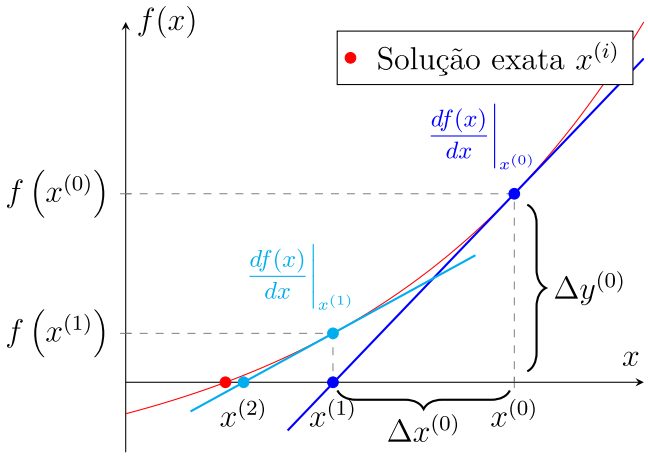
$$\Delta x^{(0)} = \left[J^{(0)}\right]^{-1} \cdot \Delta y \tag{E.3}$$

Com o valor de $\Delta x^{(0)}$, o termo $x^{(1)}$ da próxima iteração é calculado em (E.4).

$$x^{(1)} = x^{(0)} + \Delta x^{(0)} \tag{E.4}$$

Com o valor de $x^{(1)}$, calcula-se $\Delta y^{(1)} = y - f(x^{(1)})$. O método iterativo converge se forem atendidas as tolerâncias de convergência absoluta e relativa, mostradas em (E.5).

Figura 49 – Solução gráfica do método de *Newton-Raphson*.



Fonte: Elaborada pelo autor (2023).

$$\left|\Delta y^{(1)}\right| \leq \varepsilon_{nr}^{abs} \tag{E.5a}$$

$$\left|\Delta y^{(1)}\right| \leq \left|f(x^{(1)})\right| \varepsilon_{nr}^{rel} \tag{E.5b}$$

Caso não forem atendidas as relações de (E.5), retomam-se os cálculos até atingir a convergência na i-ésima iteração, conforme a solução gráfica da Figura 49.

Agora, considerando um conjunto de n equações e n incógnitas, tem-se os vetores $\mathbf{f}(\mathbf{x})$ e $\mathbf{x}$ apresentados em (E.6).

$$\mathbf{f}(\mathbf{x}) = \begin{cases} f_1\left(x_1, x_2, \ldots, x_n\right) = 0 \\ f_2\left(x_1, x_2, \ldots, x_n\right) = 0 \\ \vdots \\ f_n\left(x_1, x_2, \ldots, x_n\right) = 0 \end{cases} \tag{E.6}$$

$$\mathbf{x} = [x_1, x_2, \ldots, x_n]$$

Generalizando a equação $y = f(x)$ para um sistema n-dimensional, tem-se o seguinte:

$$\mathbf{y} = \mathbf{f}(\mathbf{x})$$

$$\mathbf{y} = \mathbf{f}\left(\mathbf{x}^{(0)}\right) + \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_1}\bigg|_{\mathbf{x}^{(0)}} (x - x_1^{(0)}) + \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_2}\bigg|_{\mathbf{x}^{(0)}} (x - x_2^{(0)}) + \ldots + \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_n}\bigg|_{\mathbf{x}^{(0)}} (x - x_n^{(0)})$$

$$\mathbf{y} - \mathbf{f}\left(\mathbf{x}^{(0)}\right) = \left.\frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_1}\right|_{\mathbf{x}^{(0)}} (x - x_1^{(0)}) + \left.\frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_2}\right|_{\mathbf{x}^{(0)}} (x - x_2^{(0)}) + \ldots + \left.\frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_n}\right|_{\mathbf{x}^{(0)}} (x - x_n^{(0)})$$

$$\Delta\mathbf{y}^{(0)} = \left.\frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_1}\right|_{\mathbf{x}^{(0)}} \Delta x_1^{(0)} + \left.\frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_2}\right|_{\mathbf{x}^{(0)}} \Delta x_2^{(0)} + \ldots + \left.\frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_n}\right|_{\mathbf{x}^{(0)}} \Delta x_n^{(0)}$$

Expandindo os vetores da equação anterior, tem-se o sistema de equações em (E.7).

$$\begin{cases} \Delta y_1^{(0)} = \left.\dfrac{\partial f_1(\mathbf{x})}{\partial x_1}\right|_{\mathbf{x}^{(0)}} \Delta x_1^{(0)} + \left.\dfrac{\partial f_1(\mathbf{x})}{\partial x_2}\right|_{\mathbf{x}^{(0)}} \Delta x_2^{(0)} + \ldots + \left.\dfrac{\partial f_1(\mathbf{x})}{\partial x_n}\right|_{\mathbf{x}^{(0)}} \Delta x_n^{(0)} \\ \Delta y_2^{(0)} = \left.\dfrac{\partial f_2(\mathbf{x})}{\partial x_1}\right|_{\mathbf{x}^{(0)}} \Delta x_1^{(0)} + \left.\dfrac{\partial f_2(\mathbf{x})}{\partial x_2}\right|_{\mathbf{x}^{(0)}} \Delta x_2^{(0)} + \ldots + \left.\dfrac{\partial f_2(\mathbf{x})}{\partial x_n}\right|_{\mathbf{x}^{(0)}} \Delta x_n^{(0)} \\ \quad\vdots \qquad\quad \vdots \qquad\qquad\quad \vdots \qquad\qquad\quad \ddots \qquad\quad \vdots \\ \Delta y_n^{(0)} = \left.\dfrac{\partial f_n(\mathbf{x})}{\partial x_1}\right|_{\mathbf{x}^{(0)}} \Delta x_1^{(0)} + \left.\dfrac{\partial f_n(\mathbf{x})}{\partial x_2}\right|_{\mathbf{x}^{(0)}} \Delta x_2^{(0)} + \ldots + \left.\dfrac{\partial f_n(\mathbf{x})}{\partial x_n}\right|_{\mathbf{x}^{(0)}} \Delta x_n^{(0)} \end{cases} \tag{E.7}$$

Escrevendo as equações de (E.7) na forma matricial, tem-se a equação matricial em (E.8).

$$\begin{bmatrix} \Delta y_1^{(0)} \\ \Delta y_2^{(0)} \\ \vdots \\ \Delta y_n^{(0)} \end{bmatrix} = \left[\begin{matrix} \dfrac{\partial f_1(\mathbf{x})}{\partial x_1} & \dfrac{\partial f_1(\mathbf{x})}{\partial x_2} & \cdots & \dfrac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \dfrac{\partial f_2(\mathbf{x})}{\partial x_1} & \dfrac{\partial f_2(\mathbf{x})}{\partial x_2} & \cdots & \dfrac{\partial f_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial f_n(\mathbf{x})}{\partial x_1} & \dfrac{\partial f_n(\mathbf{x})}{\partial x_2} & \cdots & \dfrac{\partial f_n(\mathbf{x})}{\partial x_n} \end{matrix}\right]_{\mathbf{x}^{(0)}} \cdot \begin{bmatrix} \Delta x_1^{(0)} \\ \Delta x_2^{(0)} \\ \vdots \\ \Delta x_n^{(0)} \end{bmatrix}$$

$$\Delta\mathbf{y}^{(0)} = \mathbf{J}^{(0)} \cdot \Delta\mathbf{x}^{(0)} \tag{E.8}$$

Conhecidos os termos $\Delta\mathbf{y}^{(0)}$ e $\mathbf{J}^{(0)}$, deve-se calcular o termo $\Delta\mathbf{x}^{(0)}$. Com isso, o vetor $\mathbf{x}^{(1)}$ da próxima iteração é calculado pela equação (E.9).

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \Delta\mathbf{x}^{(0)} \tag{E.9}$$

Com o valor de $\mathbf{x}^{(1)}$, calcula-se $\Delta\mathbf{y}^{(1)} = \mathbf{y} - \mathbf{f}\left(\mathbf{x}^{(1)}\right)$. Se forem atendidas as tolerâncias de convergência absoluta e relativa, o método iterativo converge. Caso contrário, calcula-se a matriz $\mathbf{J}^{(1)}$ e retomam-se os cálculos até atingir a convergência na i-ésima iteração, conforme a equação (E.10).

$$\max \left|\Delta \mathbf{y}^{(i)}\right| \leq \varepsilon_{nr}^{abs} \tag{E.10a}$$

$$\max \left|\Delta \mathbf{y}^{(i)}\right| \leq \max \left|\mathbf{f}\left(\mathbf{x}^{(i)}\right)\right| \varepsilon_{nr}^{rel} \tag{E.10b}$$

E.2 APLICAÇÃO

Nesta dissertação, o método de *Newton-Raphson* é aplicado ao cálculo do ponto inicial de operação em regime permanente da rede elétrica de transmissão. As variáveis de controle contidas no vetor $\mathbf{y}$ correspondem às potências ativas e reativas líquidas das barras da rede ($\Delta\mathbf{P}$ e $\Delta\mathbf{Q}$), e as variáveis de estado contidas no vetor $\mathbf{x}$ correspondem às magnitudes e ângulos das tensões das barras da rede ($\Delta\mathbf{V}$ e $\Delta\theta$), conforme a equação matricial em (E.11).

$$\begin{bmatrix} \Delta\mathbf{P} \\ \\ \Delta\mathbf{Q} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial\mathbf{P}}{\partial\theta} & \dfrac{\partial\mathbf{P}}{\partial\mathbf{V}} \\ \\ \dfrac{\partial\mathbf{Q}}{\partial\theta} & \dfrac{\partial\mathbf{Q}}{\partial\mathbf{V}} \end{bmatrix} \cdot \begin{bmatrix} \Delta\theta \\ \\ \Delta\mathbf{V} \end{bmatrix} \tag{E.11}$$

APÊNDICE F – MODELAGEM DE CARGA ZIP

Para representar as cargas presentes no sistema elétrico durante o regime transitório, adota-se uma modelagem estática, formada por equações algébricas de potência em função da magnitude das tensões terminais da rede do instante atual de tempo. No caso deste trabalho, adota-se a modelagem de cargas ZIP.

As cargas ZIP consistem em cargas que possuem parcela de impedância constante (Z), corrente constante (I) e potência constante (P). Suas equações de potência podem ser expressas da forma apresentada em (F.1).

$$P = \frac{P_0}{100}\left[(100 - A - B) + A\left(\frac{V_t}{V_0}\right) + B\left(\frac{V_t}{V_0}\right)^2\right] \tag{F.1a}$$

$$Q = \frac{Q_0}{100}\left[(100 - C - D) + C\left(\frac{V_t}{V_0}\right) + D\left(\frac{V_t}{V_0}\right)^2\right] \tag{F.1b}$$

onde:

$$\begin{cases} A, C: & \text{Coeficientes da parcela de corrente constante.} \\ B, D: & \text{Coeficientes da parcela de impedância constante.} \\ P_0, Q_0: & \text{Potências geradas de regime permanente.} \\ \overline{V}_0: & \text{Tensão nodal de regime permanente.} \\ \overline{V}_t: & \text{Tensão nodal, variável ao longo do tempo.} \end{cases}$$

Deve-se também se atentar para algumas restrições que devem ser impostas aos coeficientes A, B, C e D, como descritas abaixo.

$$\begin{cases} 0 \leq A,\, B,\, C,\, D \leq 100 \\ 0 \leq A + B \leq 100 \\ 0 \leq C + D \leq 100 \end{cases}$$

F.1 DEDUÇÃO MATEMÁTICA

Considerando as equações de P e Q dadas em (F.1), pode-se obter a expressão da potência aparente complexa $\overline{S} = P + jQ$ da forma apresentada em (F.2).

$$\begin{aligned} P + jQ \; = & \frac{(100 - A - B)}{100} \cdot P_0 + \frac{A}{100} \cdot P_0 \left(\frac{V_t}{V_0}\right) + \frac{B}{100} \cdot P_0 \left(\frac{V_t}{V_0}\right)^2 + \\ & + \frac{(100 - C - D)}{100} \cdot jQ_0 + \frac{C}{100} \cdot jQ_0 \left(\frac{V_t}{V_0}\right) + \frac{D}{100} \cdot jQ_0 \left(\frac{V_t}{V_0}\right)^2 \end{aligned} \tag{F.2}$$

Reescrevendo a equação (F.2), tem-se a equação resultante em (F.3), onde o termo θ corresponde à fase da tensão nodal $\overline{V}_t$.

$$\begin{aligned}\overline{S} = &\frac{(100-A-B)}{100}\cdot P_0 + j\,\frac{(100-C-D)}{100}\cdot Q_0 + \\ &+\left(\frac{A\cdot P_0 + j\,C\cdot Q_0}{100\,V_0\underline{/\theta}}\right)\overline{V}_t + \\ &+\left(\frac{B\cdot P_0 + j\,D\cdot Q_0}{100\,V_0^2}\right)V_t^2\end{aligned} \tag{F.3}$$

A partir da equação (F.3), criam-se as variáveis de corrente $\overline{I}_c$ e de impedância $\overline{y}_i$, conforme mostrado em (F.4), onde $\overline{I}_{c,aux}$ é um termo fasorial constante.

$$\overline{I}_c = \left(\frac{A\cdot P_0 - j\,C\cdot Q_0}{100\,V_0}\right)\underline{/\theta} = \overline{I}_{c,aux}\underline{/\theta} \tag{F.4a}$$

$$\overline{y}_i = \left(\frac{B\cdot P_0 - j\,D\cdot Q_0}{100\,V_0^2}\right) \tag{F.4b}$$

Com isso, pode-se escrever a potência aparente complexa $\overline{S}$ conforme mostrado em (F.5).

$$\overline{S} = \underbrace{\frac{(100-A-B)}{100}\cdot P_0 + j\,\frac{(100-C-D)}{100}\cdot Q_0}_{\text{P}} + \underbrace{\left(\overline{I}_c^*\right)\overline{V}_t}_{\text{I}} + \underbrace{(\overline{y}_i^*)\,V_t^2}_{\text{Z}} \tag{F.5}$$

Observando a equação (F.5), pode-se identificar as parcelas de impedância (Z), corrente (I) e potência (P) constante, conforme a equação (F.6).

$$\overline{S} = \overline{S}_{z\_cte} + \overline{S}_{i\_cte} + \overline{S}_{p\_cte} \tag{F.6}$$

onde:

$$\overline{S}_{z\_cte} = (\overline{y}_i^*)\,V_t^2 \tag{F.7}$$

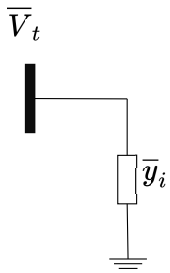
$$\overline{S}_{i\_cte} = \left(\overline{I}_c^*\right)\overline{V}_t \tag{F.8}$$

$$\overline{S}_{p\_cte} = \frac{(100-A-B)}{100}\cdot P_0 + j\,\frac{(100-C-D)}{100}\cdot Q_0 \tag{F.9}$$

F.1.1 Parcelas de impedância e de corrente constante

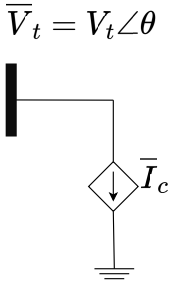
As parcelas de impedância e de corrente constante estão representadas na Figura 50 e na Figura 51.

Figura 50 – Modelagem de carga do tipo impedância constante.



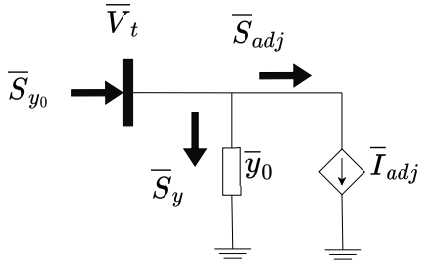
Fonte: Elaborada pelo autor (2023).

Figura 51 – Modelagem de carga do tipo corrente constante.



Fonte: Elaborada pelo autor (2023).

Figura 52 – Modelagem de carga do tipo potência constante.



Fonte: Elaborada pelo autor (2023).

F.1.2 Parcela de potência constante

A parcela de potência constante está representada na Figura 52. Sua representação consiste em uma parcela de impedância constante ($\overline{y}_0$), correspondente ao seu valor de regime permanente, e um ajuste proporcionado por uma fonte de corrente ($\overline{I}_{adj}$), para as situações em que a tensão na barra de carga apresentar variações. Essas parcelas são definidas nas expressões de (F.10) até (F.15).

Conforme a (F.9), a potência aparente complexa de regime permanente $\overline{S}_{y_0}$ é

definida pela equação (F.10).

$$\overline{S}_{y_0} = \frac{(100 - A - B)}{100} \cdot P_0 + j\,\frac{(100 - C - D)}{100} \cdot Q_0 \tag{F.10}$$

Dessa forma, a admitância $\overline{y}_0$ é calculada a partir da tensão $\overline{V}_t = \overline{V}_0$ e da potência $\overline{S}_{y_0}$, ambas de regime permanente, conforme a equação (F.11). Portanto, para $\overline{V}_t = \overline{V}_0$, a potência de ajuste torna-se identicamente nula, ou seja, $\overline{S}_{adj} = 0$.

$$\overline{y}_0 = \frac{\overline{S}^*_{y_0}}{V_0^2} \tag{F.11}$$

Ao longo de uma simulação no tempo, quando se tem $\overline{V}_t \neq \overline{V}_0$, a potência de admitância torna-se $\overline{S}_y$, conforme a equação (F.12).

$$\overline{S}_y = \overline{y}^*_0 \cdot V_t^2 \tag{F.12}$$

Assim, a potência de ajuste $\overline{S}_{adj}$ pode ser controlada de forma a manter a potência $\overline{S}_{y_0}$ entrando na barra de tensão $\overline{V}_t$ ao longo do tempo, conforme a equação (F.13).

$$\overline{S}_{adj} = \overline{S}_{y_0} - \overline{S}_y \tag{F.13}$$

Associada à potência $\overline{S}_{adj}$, calcula-se a corrente de ajuste $\overline{I}_{adj}$ mostrada na equação (F.14). Essa corrente deve ser controlada de forma a fazer com que o valor de $\overline{S}_{adj}$ sempre mantenha a potência $\overline{S}_{y_0}$ entrando na barra de tensão $\overline{V}_t$.

$$\overline{I}_{adj} = \left[\frac{\overline{S}_{adj}}{\overline{V}_t}\right]^* \tag{F.14}$$

Substituindo as equações anteriores em (F.14), tem-se a nova equação de $\overline{I}_{adj}$ apresentada em (F.15).

$$\overline{I}_{adj} = \left[\frac{\overline{S}_{y_0}}{\overline{V}_t}\right]^* \cdot \left[1 - \frac{V_t^2}{V_0^2}\right] \tag{F.15}$$

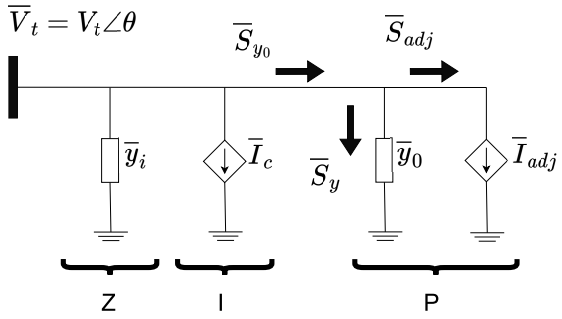
F.2 CARGA ZIP COMPLETA

Com a representação individual de cada parcela de carga, tem-se a carga ZIP completa mostrada na Figura 53. Essa modelagem visa representar a carga para uma faixa de tensões próximas à nominal, isto é, maiores ou iguais a um determinado valor V_{mn} (por exemplo, 70 %).

F.3 IMPEDÂNCIA CONSTANTE PARA BAIXAS TENSÕES

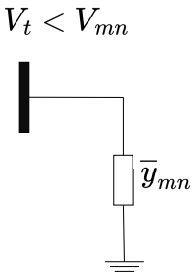
No entanto, para tensões significativamente baixas (isto é, abaixo de V_{mn}), despreza-se a modelagem completa da Figura 53 e considera-se apenas uma impedância constante $\overline{y}_{mn}$, conforme mostrado na Figura 54.

Figura 53 – Modelagem de carga ZIP completa.



Fonte: Elaborada pelo autor (2023).

Figura 54 – Modelagem de carga ZIP do tipo impedância constante para baixas tensões.



Fonte: Elaborada pelo autor (2023).

Considerando essa admitância na forma $\overline{y}_{mn} = g_{mn} - j \cdot b_{mn}$, o cálculo de g_{mn} e b_{mn} está mostrado nas equações de (F.16).

$$g_{mn} = \frac{P_0}{100} \left[\frac{(100 - A - B)}{V_{mn}^2} + \frac{A}{V_0 \cdot V_{mn}} + \frac{B}{V_0^2} \right] \tag{F.16a}$$

$$b_{mn} = \frac{Q_0}{100} \left[\frac{(100 - C - D)}{V_{mn}^2} + \frac{C}{V_0 \cdot V_{mn}} + \frac{D}{V_0^2} \right] \tag{F.16b}$$